

Міністерство освіти і науки України
Державний заклад
«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут фізики, математики та інформаційних
технологій

Кафедра інформаційних технологій та систем

Смагіна Ольга Олександрівна

**ДОСЛІДЖЕННЯ АДАПТИВНИХ МЕТОДІВ СКЛАДАННЯ РОЗКЛАДУ
НАВЧАЛЬНОГО ПРОЦЕСУ У ЗАКЛАДАХ ВИЩОЇ ОСВІТИ**

**кваліфікаційна робота
здобувача вищої освіти другого (магістерського) рівня
освітньої програми «Мультимедійні системи»
за спеціальністю 121 „Інженерія програмного забезпечення”**

Особистий підпис _____  _____ Ольга СМАГІНА

Науковий керівник _____ Галина КОЗУБ,
кандидат технічних наук, доцент
кафедри інформаційних
технологій та систем

Завідувач кафедри _____ Микола СЕМЕНОВ,
кандидат педагогічних наук,
доцент кафедри інформаційних
технологій та систем

АНОТАЦІЯ

Смагіна О.О.

Тема: дослідження адаптивних методів складання розкладу навчального процесу у закладах вищої освіти

Спеціальність: 121 "Інженерія програмного забезпечення".

Установа: ДЗ ЛНУ імені Тараса Шевченка, 2023 р.

Кваліфікаційна робота містить: 71 стор., 15 рис., 31 джерело, 5 додатків.

Об'єкт дослідження – організація та планування навчального процесу.

Предмет дослідження – алгоритмічний підхід до автоматичного та напіваавтоматичного складання розкладу занять.

Мета роботи – дослідження адаптивних методів складання розкладу та розробка програмного модуля автоматичної генерації розкладу занять закладу вищої освіти.

Методи дослідження: *Теоретичні методи:* аналіз науково-технічних джерел з проблем дослідження. *Емпіричні методи:* аналіз проблеми складання розкладу у закладі вищої освіти. *Експериментальні методи:* тестування розробленого додатку.

Результати роботи. Розроблений адаптивний метод є основою програмної реалізації системи складання розкладу навчального процесу в ЗВО. Використання даної системи в ЗВО дозволить полегшити та підвищити ефективність роботи диспетчерів, які займаються складанням розкладу, та якість навчального процесу за рахунок максимального врахування побажань учасників навчального процесу при складанні розкладу.

Ключові слова: система, програмний модуль, розклад занять, предметна область, математична модель, генетичний алгоритм, Java, SpringBoot, Hibernate.

ABSTRACT

Smahina O.O.

Topic: Research of adaptive methods of drawing up the schedule of the educational process in institutions of higher education

Specialty: 121 "Software engineering".

Institution: Luhansk Taras Shevchenko National University, 2023

The qualification work contains: 71 pages, 15 figures, 31 sources, 5 app.

The object of research is organization and planning of the educational process.

The subject of the research is an algorithmic approach to automatic and semi-automatic preparation of the class schedule.

The purpose of the work is research of adaptive methods of drawing up a schedule and development of a software module for automatic generation of a schedule of classes of a higher education institution.

Research methods: *Theoretical methods:* analysis of scientific and technical sources on research problems. *Empirical methods:* analysis of the problem of scheduling in a higher education institution. *Experimental methods:* testing the developed application.

Work results. The developed adaptive method is the basis of the software implementation of the educational process scheduling system in higher education institutions. The use of this system in higher education institutions will facilitate and increase the efficiency of the work of dispatchers who are engaged in drawing up the schedule, and the quality of the educational process due to the maximum consideration of the wishes of the participants in the educational process when drawing up the schedule.

Keywords: system, program module, class schedule, subject area, mathematical model, genetic algorithm, Java, SpringBoot, Hibernate.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ	5
ВСТУП	6
РОЗДІЛ 1 Аналіз проблеми складання розкладу занять та завдання дослідження	9
1.1. Розклад навчальних занять та його роль у життєдіяльності ЗВО	9
1.2. Опис предметної області	14
1.3. Аналіз існуючих рішень автоматизації процесу складання розкладу	16
1.4. Постановка завдання дослідження	19
Висновки до розділу 1	20
РОЗДІЛ 2. Дослідження алгоритмів автоматичної генерації розкладу занять	21
2.1. Математичне моделювання задачі складання розкладу	21
2.2. Основні проблеми автоматизації	24
2.3. Моделі та алгоритми складання розкладу занять	27
Висновки по розділу 2	39
РОЗДІЛ 3. Розробка програмного модуля автоматичного складання розкладу занять	40
3.1. Вибір засобів програмування розробки програми	40
3.2. Опис кодів програми модуля генерації розкладу занять з урахуванням генетичного алгоритму	45
3.3. Аналіз якості автоматично згенерованого розкладу	51
Висновки до розділу 3	52
ВИСНОВКИ	54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	56
ДОДАТКИ	59

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ

API	–	Application Programming Interface
АСУ	–	Автоматизовані системи керування
БД	–	База даних
ГА	–	Генетичний алгоритм
UML	–	Unified Modeling Language
REST	–	Representational State Transfer
HTTP	–	HyperText Transfer Protocol
ПЗ	–	Програмне забезпечення
SOAP	–	Simple Object Access Protocol
MVC	–	Model-View-Controller

ВСТУП

У навчальних закладах існує безліч проблем, пов'язаних із організацією навчального процесу. Однією є проблема складання розкладу занять. Процес складання та редагування розкладу складний через велику кількість факторів, що впливають на навчальний процес. Для кожного факультету, інституту, університету вони індивідуальні, тому немає універсального підходу до складання розкладу занять.

Редагування розкладу в ході навчального процесу нерідко призводить до дисбалансу в навчальному, викладацькому та аудиторному навантаженні, відповідно знижується якість навчального процесу і, як наслідок, професіоналізм фахівців, що випускаються.

Якість та швидкість складання розкладу в першу чергу залежить від досвіду укладача та зручності програмного забезпечення. Збір і підготовка вихідних даних для розкладу ускладнюється прийнятною компанією, що не закінчилася, відсутністю викладачів, великою кількістю студентів та іншими факторами, які має враховувати відповідальна за складання розкладу людина.

Використання єдиного інформаційного простору для зберігання даних про студентів, абітурієнтів, викладачів, дисциплін, аудиторій дозволяє скоротити час пошуку потрібних для складання розкладу відомостей.

Впровадження програмного модуля в єдиний інформаційний простір дозволить прискорити та спростити процес складання розкладу: складати розклад для студентів та викладачів з урахуванням вимог дисциплін, побажань викладачів, завантаженості аудиторій та типів занять; вносити зміни до розкладу та вести облік проведених занять.

Завдання зі складання розкладу занять не така проста, якою здається на перший погляд. Як відомо, було зроблено величезну кількість спроб її вирішення. На жаль, жодна з цих програм не має необхідної універсальності та не задовольняє потреб усіх ЗВО, або має надто високу вартість.

Отже, дослідження, створені задля розробку алгоритмів автоматизації побудови навчального розкладу, є актуальними.

Мета роботи – дослідження адаптивних методів складання розкладу та розробка програмного модуля автоматичної генерації розкладу занять закладу вищої освіти.

Об’єкт дослідження: організація та планування навчального процесу.

Предмет дослідження – алгоритмічний підхід до автоматичного та напівавтоматичного складання розкладу занять.

Для досягнення даної мети повинні бути вирішені наступні **завдання**:

- описати предметну область розкладу занять;
- визначити вимоги та механізм складання розкладу;
- дослідити існуючі методи та алгоритми автоматичної генерації розкладу;
- розробити модуль автоматичної системи складання розкладу.

Методи дослідження:

Теоретичні методи: аналіз науково-технічних джерел з проблем дослідження. *Емпіричні методи:* аналіз проблеми складання розкладу у закладі вищої освіти. *Експериментальні методи:* тестування розробленого додатку.

Практичне завдання. Розглянуті у роботі підходи може бути покладено основою подальших досліджень, а отриманий результаті розробки модуль автоматичної генерації може бути фундаментом єдиної автоматизованої інформаційної системи університету.

Структура дипломної роботи. Робота складається з пояснювальної записки, списку використаних джерел, додатків. Обсяг роботи становить 71 сторінки, обсяг використаної літератури - 31 джерело.

В пояснювальній записці у першому розділі шляхом аналізу проблем, що виникають у процесі складання розкладу, обґрунтовується актуальність проведення досліджень. Для оцінки існуючих програмних рішень

автоматизації процесу проводиться опис предметної області та вимог, що висуваються до розкладу. За підсумками результатів огляду існуючих програмних рішень автоматизації процесу виконується постановка завдань дослідження.

У другому розділі дослідження алгоритмів автоматичної генерації розкладу занять виконується математичне моделювання завдання та виявляються основні проблеми автоматизації. Для якісного вирішення даних проблем досліджуються моделі та алгоритми складання розкладу. З проведеного дослідження здійснюється вибір методу автоматизації процесу.

У третьому розділі здійснюється вибір методів та засобів програмування. Наводиться опис роботи основних програмних вузлів модуль автоматичної генерації розкладу. Також оцінюється якість розкладу, що згенерував програмний модуль. Розробляються практичні рекомендації щодо модернізації, оптимізації модуля і, отже, підвищення якості складання розкладу.

Додаток містить в собі код програми.

РОЗДІЛ 1.

АНАЛІЗ ПРОБЛЕМИ СКЛАДАННЯ РОЗКЛАДУ ЗАНЯТЬ ТА ЗАВДАННЯ ДОСЛІДЖЕННЯ

1.1. Розклад навчальних занять та його роль у життєдіяльності ЗВО

Завдання складання розкладів є предметом наукових досліджень із середини минулого століття. Область їх застосування включає різні сфери людської діяльності, такі як: транспортні перевезення, масове обслуговування, промисловість, освіта тощо. Практика висуває безліч завдань, які неможливо ефективно вирішити шляхом повного перебору. Для більшості моделей теорії розкладів знаходження оптимального розкладу є важкою задачею, а рішення наближених до реальних умов задач має ще більшу складність, оскільки дані рішення повинні задовольняти численним, найчастіше конфліктуючим між собою обмеженням виробничого, організаційного та психофізіологічного характеру. Виходом із цього положення є відмова від підходу, коли придатним вважається лише найкраще рішення.

Кількісне та якісне зростання вищої школи потребує нового підходу до вирішення завдань управління навчальною, науковою та господарською діяльністю ЗВО. Цей підхід останніми роками знаходить своє втілення у застосуванні сучасних засобів обчислювальної техніки та математичних методів в управлінні закладами вищої освіти. У сучасному світі все більшого поширення набувають різноманітні системи автоматизації технічних процесів, які завжди виконувались вручну. Наприклад, системи прийняття рішення в маркетингу, експертні системи, що замінюють досвідчених фахівців, прогнозують системи в різних галузях науки і техніки. До таких самих процесів відноситься і складання розкладу, який досі у багатьох навчальних закладах створюється вручну на основі багаторічного досвіду.

Сучасні ІТ-технології мають у своєму розпорядженні засоби, що дозволяють найкраще організувати будь-який процес, зокрема і навчальний.

Завдання планування розкладу навчальних занять – це завдання складання розкладу комбінаторного типу, характерною особливістю якої є величезна розмірність і наявність великої кількості обмежень складної форми. Фактично, нині, немає універсальних методів розв’язання таких завдань. Пряме застосування математичної (класичної) теорії розкладу до завдання складання навчальних занять неможливо [1, 2, 3]. Тим не менш, є низка евристичних та перебірних методів, які цілком піддаються програмуванню.

Є думка, що досвідчений диспетчер зможе скласти розклад так, що він відповідатиме інтересам навчального процесу та суспільного життя ЗВО. Однак із цим не можна погодитися. Ручне вирішення завдання складання розкладу занять потребує великих витрат часу, кваліфікованих спеціалістів, водночас результат такого рішення часто виходить далеко не оптимальним. Після введення вихідної інформації потрібно її узгодження, тоді як неможливість отримання необхідного розкладу можна визначити ще етапі аналізу.

Під час складання розкладу можливе виникнення складних ситуацій. Все це вимагає зміни вихідних даних та послаблення обмежень, і тут без людини не обійтися. Без внесення цих змін розклад не матиме практичної цінності. Також слід враховувати той фактор, що розклад може змінюватися і під час його використання, тобто після складання, і тут дуже важливим є людський фактор. У цьому плані важлива підтримка цього процесу автоматизованими методами та процедурами. Основна перевага полягає в тому, що автоматизоване складання усуває масу рутинної роботи, такої як: пошук можливих варіантів внесення чергових елементів до розкладу, перевірку виконання вимог, пошук випадкових помилок у готовому розкладі, оформлення розкладу на папері у вигляді різних таблиць (для викладачів, груп, аудиторного фонду), залишаючи людині більше часу більш

інтелектуальні дії. Комп'ютер також є інструментом, що значно посилює здібності людини, бо людина не спроможна перебрати і проаналізувати таку кількість варіантів розкладів, як комп'ютер.

В останні роки робляться множинні спроби вдосконалення планування навчального процесу шляхом побудови алгоритмів оптимізації завдань планування навчальної роботи ЗВО з використанням обчислювальної техніки та програмного забезпечення Microsoft Excel. Практичне впровадження планування навчального процесу з використанням вебтехнологій має місце лише в небагатьох ЗВО. Аналіз стану цих розробок дозволяє зробити такі висновки: - Розробка та впровадження ЗВО завдань АСУ здійснюється в ініціативному порядку і ці роботи, як правило, спрямовані на вирішення окремих проблем. Роз'єднаність груп дослідників та розробників призвела до створення безлічі систем, спрямованих на розробку алгоритмів та програм, розрахованих на обслуговування лише конкретного вишу. - багато систем покладають на розробника розкладу всю відповідальність за облік реальних вимог. Зокрема, облік вимог викладачів, обмежень на кількість занять, що проводяться на день, на тиждень — усі ці та багато інших рутинні завдання в таких системах доводиться вирішувати людині найчастіше методами перебору. - наявні програми не передбачають розрахований на багато користувачів режим роботи і не підтримують весь необхідний електронний документообіг. - не впроваджується розробка типових уніфікованих елементів створення єдиної автоматизованої системи управління вищою школою. - Наявні програми мають дуже незручний інтерфейс для введення вихідних даних та редагування отриманого розкладу. У зв'язку з розширенням робіт з удосконалення системи управління вищою школою шляхом створення та впровадження у ЗВО різних автоматизованих систем управління виникає потреба уніфікувати засоби складання навчального розкладу на обчислювальній техніці. Для цього необхідно чітко формалізувати вимоги до розкладу та розробити відповідне алгоритмічне забезпечення.

При розробці алгоритмів автоматизованого складання розкладу занять гостро постає проблема створення універсальних алгоритмів, що враховують специфіку умов кожної конкретної задачі. Такі алгоритми мали бути досить «гнучкими», тобто без істотної їх зміни можна було б включати та виключати вимоги із системи вимог до розкладу. Проте спроба вирішувати завдання якимось одним єдиним універсальним алгоритмом на даний момент неможливо. Алгоритми, що дозволяють вирішувати широкий клас завдань, не дають ефективності, яку забезпечують більш конкретні, адаптовані з урахуванням конкретних умов алгоритми [1].

Для систем складання розкладу занять характерна сильна залежність від специфіки конкретних навчальних закладів вже на рівні математичних моделей та подання даних, що ускладнює використання типових систем. Систему, створену в одному ЗВО, зазвичай без зміни та доопрацювання неможливо ефективно використовувати в іншому. До того ж багато хто з них створювався досить давно і за їх допомогою неможливо ефективно вирішувати поставлене завдання.

Для вирішення існуючих проблем потрібна побудова гнучкої системи, що легко адаптується, на основі нових принципів, з використанням сучасних веб-технологій. Необхідна система, що становить розклад відповідно до вибраних критеріїв та заданих вимог [18]. Ці можливості повинні здійснюватися також без зміни вихідного коду системи. Для покриття найбільш типових випадків необхідно створення кількох типових алгоритмів, що реалізують складання розкладів. Дана система повинна мати можливість доповнення та зміни існуючої бази даних та інтерфейсу користувача. Все це давало б можливість задавати в кожному ЗВО вимоги, що відповідають його умовам, та за допомогою підбору та налаштування відповідного алгоритму отримувати необхідний розклад.

1.2 Опис предметної області

Розклад є інструментом керування навчанням студентів. Він складається на семестр чи рік для групи чи підгрупи студентів залежно від навчального закладу.

Розклад має містити таку інформацію:

- номер навчального тижня;
- день тижня;
- номер пари;
- час початку та закінчення пари;
- номер аудиторії;
- освітній компонент;
- тип заняття;
- викладач;
- група, підгрупа чи потік.

Вхідними даними для складання розкладу занять є: графік навчального процесу; навчальний план; відомості про аудиторний фонд; відомості про навчальні групи та кількість учнів у них; розподіл учнів з лекційних потоків, груп, підгруп; відомості кафедр про закріплення освітніх компонентів за викладачами.

При складанні розкладу диспетчер має враховувати такі вимоги:

- розклад по можливості складається на семестр і має бути рівномірним протягом тижня;
- навчальні заняття мають здійснюватись з дотриманням Правил внутрішнього розпорядку ЗВО;
- за заявками кафедр слід передбачати час для кураторської години;
- освітні компоненти необхідно розміщувати так, щоб забезпечувалася логічна послідовність вивчення, враховувалося чергування типів занять;

- щоденне навантаження учнів не повинно перевищувати восьми академічних годин (чотири пари) без урахування факультативних занять та іншої діяльності;
- розміщення групи на заняття у різні аудиторії не допускається, якщо на групу призначено одного викладача;
- читання кількох освітніх компонентів групи на одному занятті не допускається;
- не допускається читання дисциплін одним викладачем для двох і більше груп, якщо вони не об'єднані у потік та не розміщені в одній аудиторії;
- викладачі з інших ЗВО можуть читати дисципліни у певні дні та години;
- кількість місць в аудиторії має відповідати кількості осіб у групі чи потоці;
- в одній аудиторії не допускається проведення кількох лекцій одночасно;
- аудиторія має відповідати типу заняття.

Порядок складання розкладу у навчальних закладах [4]:

- розклад занять складає співробітник, відповідальний за складання розкладу, призначений деканом факультету;
- термін складання розкладу занять та проведення екзаменаційних сесій визначається календарним графіком планування та організації навчального процесу;
- розклад розміщується на інтернет-ресурсах та на інформаційних стендах ЗВО.
- затверджений розклад має виконуватись усіма працівниками університету, які мають відношення до проведення навчального процесу;
- зрив занять не допускається;

– зміна розкладу допускається рішенням декана.

Незважаючи на вищеописані вимоги система повинна мати певну гнучкість, бо умови у навчальних закладах можуть відрізнятись. При необхідності система повинна дозволяти не враховувати ці вимоги, давати повну свободу диспетчеру, але при цьому попереджаючи про можливі наслідки.

Складений розклад занять має зберігатися у базі даних.

За запитами користувачів ІС має виводити розклад студентів та викладачів.

1.3 Аналіз існуючих рішень автоматизації процесу складання розкладу

Проблемі автоматизованого формування розкладу навчального процесу протягом останніх десятиліть приділяється значна увага.

Публікації з даного питання можна поділити на декілька категорій:

– формалізація предметної області:

Так, в [3] розглянуто математичну постановку задачі складання розкладу занять у ЗВО в умовах розбіжності вимог і побажань викладачів і студентів, введено поняття віртуальних та узагальнених груп і підгруп, проаналізовано жорсткі та м'які обмеження в задачі складання розкладу.

Робота [4] присвячена стандартизації даних для складання розкладу у навчальних закладах. Автори описують формат представлення даних у частині опису навчального плану і розкладу занять на основі XML-технологій.

В роботі [5] розглядається питання формування цільової функції, за допомогою якої оцінюється якість складеного розкладу навчального процесу, з подальшою оптимізацією цільової функції на основі використання еволюційних технологій і матричного подання розкладу; аналіз існуючих та розробка нових алгоритмів, моделей та методів складання розкладу:

У роботі [6] наведено огляд методів складання розкладу ЗВО, проаналізовані їхні переваги та недоліки. Опис основних методів, що використовуються для розв'язання даної задачі, також представлено у [7].

Для автоматичного складання розкладу широко використовуються як класичні методи такі, як лінійне цілочисельне програмування, метод розфарбування графу, метод імітаційного моделювання, так і метаевристичні методи.

Так, застосування методу розфарбування графу для складання розкладу розглядається у роботах, починаючи з 1967 р. (Welsh) по сьогоднішній день. У роботі [8] розглянуті варіанти підходів до складання розкладу за допомогою даного методу, запропоновані різними авторами, та описане використання методу розфарбування графу для складання розкладу екзаменів.

Але оскільки дану задачу можна віднести до класу NP-важких задач багатокритеріальної комбінаторної оптимізації зі значною кількістю обмежень та складністю побудови математичної моделі, для яких не доведено існування ефективних методів та алгоритмів, які знаходять точне рішення за поліноміальний час, використання класичних методів обмежене. Тому сьогодні для розв'язання задачі складання розкладу ЗВО дуже поширене застосування метаевристичних методів. Застосовують такі метаевристичні методи як метод імітації відпалу, генетичні алгоритми, метод мурашиних колоній тощо. Ці алгоритми пошуку можуть забезпечувати високоякісні рішення, але часто мають значну обчислювальну вартість.

У роботі [7] розроблено алгоритм побудови розкладу для дистанційного навчання на основі генетичного алгоритму, реалізований у якості підсистеми для системи дистанційного навчання «Віртуальний Університет». Генетичний алгоритм для побудови розкладу також розглядається у роботах [9, 10] для побудови розкладу іспитів. При цьому особлива увага приділяється послідовності іспитів. Також генетичні

алгоритми застосовують в автоматизованій системі розподілу навантаження викладачів і студентів в роботі [11].

У роботі [12] запропоновано новий варіант алгоритму імітації відпалу, який називається FastSA, для автоматичної побудови розкладу іспитів в ЗВО. В FastSA кожен іспит, обраний для планування, переміщається (і даний рух оцінюється) лише в тому випадку, якщо цей іспит мав якісь прийняті рухи на попередньому відрізку. Для побудови початкового рішення використовується евристика на основі ступеня насичення, поєднана зі статистикою на основі конфліктів, щоб уникнути зациклення.

У [13] розглядається побудова нейронної мережі для розв'язання даної задачі.

Робота [14] описує підхід до складання розкладу за допомогою максимінного методу мурашиного алгоритму (Best-WorstAntSystem) та максимінного методу мурашиних колоній (Best-WorstAntColonySystem), а також у комбінації з 2 типами локального пошуку. Доведено, що комбінація методу мурашиних колоній та локального пошуку значно підвищує ефективність алгоритму. Також результати експериментів показали перевагу методу максимінного мурашиних колоній над максимінним мурашиним алгоритмом.

Також в останній час з'являються роботи, присвячені гібридним методам складання розкладу. Так, у [15] автор запропонував метод складання розкладу, використовуючи комбінований підхід методу підживлення бактерій та генетичного алгоритму. Але обчислювальна вартість такого підходу порівняно висока;

– аналіз існуючих та розробка нових програмних засобів генерації комп'ютерного розкладу: технології, що використовуються, та їх програмна реалізація [16 – 19]:

Задача генерації розкладу відноситься до задач автоматизації діяльності ЗВО, тому деякі навчальні заклади використовують засоби складання розкладу, які є складовою системи документообігу.

Розглянемо особливості деяких систем автоматизованого формування розкладу навчального процесу.

Комплексна система складання університетського розкладу UniTime [16] розроблена зусиллями групи викладачів, студентів і співробітників університетів Північної Америки та Європи.

UniTime підтримує складання розкладів курсів та іспитів, управління змінами в цих розкладах, спільне використання аудиторій з іншими заходами і розподіл студентів за окремими курсами (див. рис. 1.1).



Рисунок 1.1 – Структура системи UniTime

Це розподілена система, яка дозволяє декільком відповідальним особам координувати зусилля зі створення і зміни розкладу, що відповідає їхнім організаційним потребам, мінімізуючи конфлікти в ході роботи.

Систему можна використовувати окремо для створення і ведення розкладу занять і / або іспитів або для взаємодії з існуючою АІС ЗВО. Дані для розкладу можуть вводитися безпосередньо в системі або імпортуватися із xml-файлів (див. рис. 1.2).

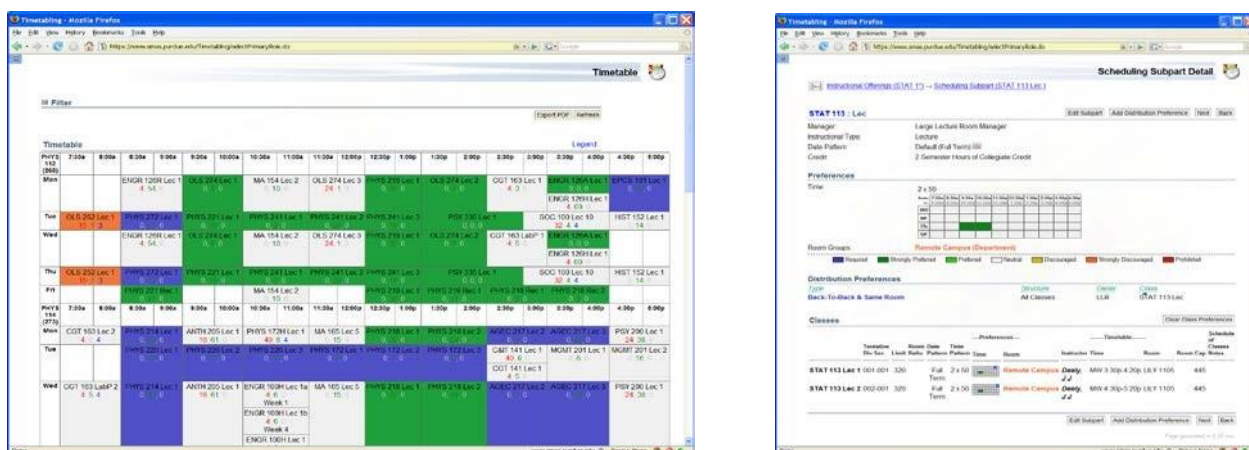


Рисунок 1.2 – Екранні форми системи UniTime

Алгоритм пошуку в UniTime ґрунтується на ітеративному алгоритмі прямого пошуку (iterative forward search algorithm). Цей алгоритм схожий з локальними методами пошуку; однак, на відміну від класичних локальних методів пошуку, він працює з допустимими, але не обов'язково повними рішеннями. У цих рішеннях деякі заняття можуть залишатися не розміщеними. Однак усі жорсткі обмеження щодо призначених занять повинні виконуватися. Такі рішення простіші для візуалізації та більш значущі для користувачів, ніж повні, але нездійсненні рішення. Через ітеративний характер алгоритму алгоритм може запускати, зупиняти або продовжувати обробляти будь-яке допустиме рішення – повне або неповне.

Даний додаток поширюється безкоштовно за ліцензією з відкритим вихідним кодом для використання іншими коледжами та університетами, а також для сторонніх дослідників, які захочуть внести вклад в дослідження в цій області. Недоліком цієї системи є орієнтація на англомовні країни.

На основі аналізу особливостей, переваг та недоліків представлених систем, була поставлена задача розроблення гібридного алгоритму створення розкладу навчального процесу та реалізації системи складання розкладу навчального процесу в університеті з використанням розробленого методу.

1.4 Постановка завдання дослідження

Однією з цілей переслідуваних під час проведення автоматизації процесу складання розкладу занять є створення єдиної інформаційної системи. Наявність єдиного центру зберігання інформації (бази даних) з метою мінімізувати функції окремих користувачів – не менш важливе завдання створення автоматизованих систем. У зв'язку з наявністю у ЗВО єдиної бази даних, однією з вимог до системи, що розробляється, є можливість інтеграції в цю базу, для використання інформації, що зберігається в ній, про здобувачів освіти, кафедри та навчальні плани. Виконання цієї вимоги для збереження універсальності розроблюваного рішення вкрай важливо.

Швидкість складання та якість автоматичного розкладу безпосередньо залежить від алгоритму, що застосовується у програмному модулі. Тому особливу увагу слід приділити дослідженню алгоритмів автоматичного складання розкладу занять. Для вибору оптимального алгоритму необхідно провести математичне моделювання цієї задачі та визначити основні проблеми автоматизації.

На основі отриманих у результаті дослідження даних та вимог до розкладу необхідно розробити модуль автоматичної генерації розкладу занять.

Висновки до розділу 1

У ході дослідження процесу складання розкладу у ЗВО було вивчено предметну область, проаналізовано існуючі програмні засоби автоматизації процесу складання розкладу, визначено вимоги до розкладу. Виконано постановку завдання дослідження.

Внаслідок чого можна зробити висновок, що завдання створення власної автоматизованої інформаційної системи складання розкладу занять є актуальним.

РОЗДІЛ 2.

ДОСЛІДЖЕННЯ АЛГОРИТМІВ АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ РОЗКЛАДУ ЗАНЯТЬ

2.1 Математичне моделювання задачі складання розкладу

При складанні розкладу існують обов'язкові та додаткові, так звані бажані вимоги [8, 9, 10], які можуть змінюватися в залежності від навчального закладу.

Обов'язкові вимоги:

- аудиторія повинна вміщати всіх здобувачів освіти, які навчаються у поточний період;
- аудиторія має відповідати типу заняття;
- відсутність накладок у розкладі;
- повне виконання навчального плану;
- виконання кількісних норм занять щодня.

Додаткові вимоги:

- мінімізація кількості вікон у здобувачів освіти;
- дотримання логічної послідовності розміщення занять освітніх компонентів;
- побажання викладачів.

Для якісного складання розкладу необхідно визначити вихідні дані та сформулювати математичну модель. Математична модель дозволить уникнути накладок у розкладі.

Вихідні дані, як початкова інформація, представимо у вигляді множин:

- викладачі; $P = \{p_1, p_2, \dots, p_n\}$
- освітні компоненти; $D = \{d_1, d_2, \dots, d_n\}$
- групи; $G = \{g_1, g_2, \dots, g_n\}$

- аудиторії; $A = \{a_1, a_2, \dots, a_n\}$
- часові інтервали. $T = \{t_1, t_2, \dots, t_n\}$

Як теоретико-множинну модель можна розглядати функцію, що відображає декартовий добуток множин на булеву множину.

$R = D \times G \times A \times P \times T \rightarrow \{0,1\}$, де , означатиме, що з освітнього компонента для групи викладачем проводиться заняття в аудиторії під час пари . Для планування розкладу необхідно розглянути основні його сутності.

$$R(d_i, g_l, p_m, a_j, t_k) = 1 d_i g_l p_m a_j t_k$$

1. аудиторії. У всіх ЗВО аудиторії діляться на лекційні та лабораторні. У вихідній інформації необхідно враховувати ці фактори, тому безліч аудиторій розпишемо як $A_{lec} A_{lab} A = A_{lec} + A_{lab}$

Розподіл аудиторій на 2 підмножини умовно, оскільки через брак аудиторного фонду на одній аудиторії може бути кілька груп: одна група виконує лабораторні роботи, інша слухає лекцію чи кілька груп на потокової дисципліні. У зв'язку з цим особливостей слід перетворити нашу множину аудиторій в такий спосіб (2.1):

$$A = A_{lec} + A_{lab} + A_{leclab} \quad (2.1)$$

Кожна аудиторія розташовується у певному корпусі (за наявності кількох), поверсі та під певним номером. Кожен університет має свої особливості нумерації, але в загальному випадку номер аудиторії має такий вигляд 1-123, де 1 номер корпусу, 1 номер поверху, 23 номер кабінету. Відповідно для формального опису аудиторії введемо тернарний кортеж (2.2):

$$A = \{a_j\}, a_j = (a_v^j, a_s^j, a_{type}^j), \quad (2.2)$$

де v – корпус, s – номер аудиторії, $type$ – тип аудиторії.

2. Часовий інтервал. Розклад зазвичай складається на один семестр, в якому відома кількість тижнів навчання, навчальних днів та пар на день. Позначимо кожен складову множини T у вигляді тернарного кортежу (2.3):

$$T = \{t_k\}, t_k = (t_w^k, t_d^k, t_p^k),$$

де – номер тижня, – номер дня тижня, – номер пари у навчальному дні. $t_w^k t_d^k t_p^k$

3. Множина, що пов'язує між собою викладачів, дисципліни та групи, що читаються. При створенні розкладу у нашому розпорядженні є список викладачів, навчальних груп та занять, що проводяться для них. Завдяки цим даним ми можемо зробити висновок у яких групах які предмети мають бути проведені та якими викладачами. З усіх цих зв'язків можна назвати нову структуру – заняття, що є множина Z (2.4).

$$Z = \{z_i\}, z_i = (z_p^i, z_d^i, z_g^i, z_s^i, z_e^i, z_h^i, z_w^i, z_c^i, z_l^i, z_a^i), \quad (2.4)$$

z_p^i – викладач, z_d^i – дисципліна, z_g^i – група, z_s^i – ознака потокового заняття, z_e^i – ознака підгрупи, z_h^i – кількість занять, z_w^i – кількість пар на тиждень, z_c^i – кількість пар одного заняття, z_l^i – тип заняття, z_a^i – допустиме підмножина аудиторій.

Таким чином, при складанні розкладу нам необхідно три множини Z , A , T , які можна визначити такими векторами:

$\alpha = (\alpha_i) \in A$ - Номер аудиторії для циклу занять. z_i

$\tau = (\tau_i) \in T$ - Код пари з циклу. z_i

Математична модель створення розкладу для навчального закладу виявляється у задачі нелінійного булева програмування з умовами, що обмежують. Така форма сприяє прийняттю до уваги всіх умов, що висуваються до цієї моделі.

Обмежувальні умови:

- відсутність накладок у викладачів та груп у допустимих аудиторіях;
- максимальна кількість занять на день не повинна перевищувати зазначений поріг;
- аудиторія має відповідати типу заняття;
- мінімізація «вікон» у навчальних днях для здобувачів освіти;

2.2 Основні проблеми автоматизації

Очевидний підхід скласти розклад, розглядаючи всі можливі варіанти (метод повного перебору), можливий, коли математичне завдання, одержане після формалізації поставленого завдання, має порівняно невелику розмірність. Як правило, застосування методу повного перебору до завдань складання розкладу занять у ЗВО важку, скоріше взагалі неможливу. Завдання знаходження оптимального розкладу занять здебільшого належить до класу важковирішуваних завдань. Якщо враховувати реальні умови, то завдання ще більше ускладнюється, оскільки шукані рішення повинні задовольняти великій кількості обмежень виробничого, організаційного та психофізіологічного характеру, що суперечать один одному. У цьому оптимальне рішення може існувати. Для подолання цієї ситуації потрібно знайти компроміс між критеріями, тобто скласти згортку критеріїв в один узагальнений критерій. Якщо такий підхід не дає оптимального рішення, слід відмовитись від пошуку оптимального рішення. Як розв'язання задачі в цьому випадку слід виділити не єдине рішення, а кілька прийнятних варіантів, з яких вибрати остаточний варіант рішення [12]. Подібний підхід доцільно застосувати до складання розкладів навчальних занять у ЗВО. Розглянута задача відноситься до завдань комбінаторного типу з великою кількістю змінних та обмежень, тобто. має велику розмірність, причому обмеження часто мають складну форму. Універсальних методів на вирішення подібних завдань практично немає. Розроблено досить універсальні методи та алгоритми для завдань, що добре формалізуються. До таких завдань відносяться, наприклад, завдання завантаження обладнання, задачі про призначення, завдання транспортного типу та ін [13 - 15]. Застосувати методи вирішення подібних завдань для розв'язання задачі складання розкладу навчальних занять, як правило, не вдається. Насправді досвідчений

диспетчер зможе скласти розклад, не намагаючись формалізувати цей процес. При цьому складений вручну розклад враховуватиме інтереси як навчального процесу, так і суспільного життя освітньої установи. Очевидно, це можливо лише у невеликих навчальних закладах (типу філій ЗВО).

Слід зазначити, що повністю виключати людину із процесу складання розкладу не можна. Оперативне прийняття рішення на етапі аналізу вихідної інформації, вирішення можливих тупикових ситуацій, що потребують зміни вихідних даних або послаблення обмежень, неможливе без участі людини. З іншого боку, розклад може змінюватися під час його використання, тобто. після складання, без можливості оперативної зміни воно не матиме практичної цінності. Здійснити такі зміни може лише людина.

Однак для ефективного управління цим процесом (розробкою розкладу), для усунення великого обсягу рутинної роботи необхідно мати різні методи та процедури, що автоматизують такі види робіт, як:

- пошук можливих варіантів внесення чергових елементів у розклад;
- перевірка виконання вимог;
- пошук випадкових помилок у готовому розкладі;
- оформлення розкладу на папері як таблиць кожної групи користувачів (груп, викладачів).

Використання комп'ютера дозволяє розглянути та проаналізувати дуже багато варіантів розкладу, що не може зробити людина. З метою підвищення ефективності навчального процесу у деяких навчальних закладах розроблялися та продовжують розроблятися алгоритми для вирішення локальних завдань автоматизації процесів в окремих підрозділах. Зокрема, було розроблено та реалізовано на обчислювальній техніці алгоритми планування безпосередньо навчальної роботи ЗВО [16, 17]. Проте кількість ЗВО, які зуміли запровадити планування навчального процесу з використанням комп'ютерної техніки, невелика. Аналізуючи наявні розробки

з проблем автоматизації процесів, що мають місце у ЗВО, можна зробити такі висновки.

Розробка та впровадження завдань АСУ у вишах здійснюється в ініціативному порядку роз'єднаними дослідницькими групами. Їхніми зусиллями створено локальні системи, розроблено алгоритми та програми для використання в конкретному навчальному закладі. Математичні моделі системи складання розкладу занять та подання даних розроблені для конкретних навчальних закладів та враховують лише їх специфічні особливості. Тому використання типових розробок важке. Навіть локальну систему, розроблену та впроваджену в одному ЗВО, неможливо ефективно використовувати в іншому без істотних змін і доопрацювань. За облік реальних вимог у багатьох системах відповідає розробник розкладу. Розрахований на багато користувачів режим роботи в програмах відсутній, погано підтримується необхідний електронний документообіг. Розробки типових уніфікованих елементів для створення єдиної автоматизованої системи керування вищою школою практично відсутні. Інтерфейс багатьох програм недружній. Він незручний, наприклад, для введення вихідних даних та редагування отриманого розкладу. Для вдосконалення системи керування вищою школою необхідно розробити уніфіковані засоби складання навчального розкладу з використанням обчислювальної техніки. Щоб вирішити цю проблему, насамперед необхідно чітко сформулювати вимоги до розкладу, дати їх опис у формалізованому вигляді та розробити відповідні універсальні алгоритми. Обов'язковими вимогами до універсальних алгоритмів є їх гнучкість, тобто необхідно, щоб універсальні алгоритми враховували специфіку умов конкретних завдань,

Наразі такого універсального алгоритму немає. Слід зазначити, що алгоритми, призначені на вирішення широкого класу завдань, часто менш ефективні, ніж алгоритми, які враховують конкретні умови вузького класу завдань [15, 18]. Щоб вирішити проблеми автоматизації навчального процесу, необхідно розробити та побудувати гнучку систему, засновану на

нових принципах. Система має легко адаптуватися до конкретного навчального закладу. Природно, що з досягнення ефективних результатів необхідно використовувати сучасні комп'ютерні технології. Розклад, що складається системою, повинен відповідати обраним критеріям та заданим вимогам. Система повинна звільнити людину від рутинної роботи, взяти на себе, по можливості, більшу частину її функцій, щоб ручне коригування розкладу, якщо таке буде потрібно, скоротилася якнайбільше. Адаптація системи до конкретного навчального закладу та інші її можливості повинні здійснюватися без зміни вихідного коду цієї системи. Необхідно мати певний набір стандартних алгоритмів, які дозволять створювати розклад у типових випадках. У системі повинна бути передбачена можливість доповнення та зміни існуючої бази даних та інтерфейсу користувача. Така система, задавши умови та вимоги конкретного вишу, обравши та налаштувавши відповідний алгоритм із набору стандартних алгоритмів, дозволить отримати навчальний розклад для цього навчального закладу.

2.3 Моделі та алгоритми складання розкладу занять

Завдання складання розкладів є одним із найпоширеніших завдань, що вирішуються кожною людиною (усвідомлено чи ні) практично щодня. У загальній постановці завдання складання розкладів є процесом розподілу деякого кінцевого набору подій у часі в умовах ресурсних та інших обмежень. Таким чином, людина, плануючи робочий день, та диспетчер, складаючи розклад занять у ЗВО або графік робіт на підприємстві, вирішують завдання складання розкладу. Але якщо у першому випадку завдання може вирішуватися інтуїтивно на основі життєвого досвіду, то у другому вона може виявитися непосильно складною навіть для групи фахівців. Така ситуація виникає через залучення до розкладу великої кількості людей зі своїми інтересами та вимогами, задоволення яких часто призводить до конфліктних ситуацій [19]. Тому з розвитком обчислювальних

технологій ведуться розробки автоматизованих систем складання розкладу. У деяких випадках вдалося розробити алгоритми, здатні знайти рішення за прийнятний час. У той же час більшість реальних завдань складання розкладу відносяться до класу задач, що важко вирішувати. Це робить розробку алгоритму, здатного вирішити їх за допустимий час, справді складним завданням, навіть якщо відповідне предметне завдання можна поставити як однокритеріальне. Ситуація суттєво посилюється тим, що більшість реальних завдань складання розкладів є багатокритеріальними. Різні завдання складання розкладів може мати багато спільного. Наприклад, у завданнях складання розкладу занять у ЗВО та графіка робіт на підприємстві можна провести такі аналогії між ресурсами: групи студентів та службовці, викладачі та зміни, аудиторія та кваліфікація службовців, предмети та роботодавці [13]. Тому методи, розроблені одного підкласу завдань, часто можна перенести інші.

Модель лінійного цілісного програмування

Застосування комбінаторних методів до завдання складання розкладу обмежується розмірністю завдання, що призводить до обмеження часового періоду розкладу. Для розкладів ЗВО цей період зазвичай становить два тижні, тобто розклад циклічно повторюється кожні два тижні весь семестр. Але у навчальних планах деяких ЗВО освітні компоненти можуть мати будь-яку кількість годин, і тоді ця вимога не виконується. Якщо навчальні плани складено таким чином, що освітні компоненти можуть читатися лише один раз на тиждень або через тиждень та суперечливість вимог усунута, то завдання складання оптимального розкладу може бути поставлене як завдання цілісного цілісного програмування. За такого підходу інтереси суб'єктів навчального процесу враховуються як обмежень чи редукованих критеріїв оптимальності [19].

Отже, нехай у ЗВО є N навчальних груп, об'єднаних в R потоків, r – номер потоку, $r = 1, \dots, R$; k_r – номер навчальної групи в потоці r , $k_r = 1, \dots, G_r$. Для кожної групи k_r визначається безліч номерів робочих днів цієї

групи T_{kr} , яка є підмножиною всіх робочих днів навчального закладу. Кожен робочий день розбивається на періоди навчання – пари, загальна кількість яких J , а $j = 1, \dots, J$, де J – номер конкретної пари.

Далі, з урахуванням навчального плану кожному за потоку складається список з S_r лекційних занять, де $s_r = 1, \dots, S_r$, де S_r – номер конкретної дисципліни у списку. Для кожної групи k_r складається список із Q_{kr} запланованих практичних занять, $q_{kr} = 1, \dots, Q_{kr}$, де Q_{kr} – номер освітнього компонента у списку. Список усіх занять кожної конкретної групи буде складатися з усіх занять, присутніх у списках. При цьому, якщо з якоїсь дисципліни протягом тижня проводиться більше одного заняття, вона згадується у списку лекцій чи практичних занять стільки разів, скільки це передбачено навчальним планом.

Для кожного викладача з безлічі всіх викладачів введемо на розгляд булеві значення:

$$\delta_{rS_r}^p = \begin{cases} 1, & \text{якщо на потоці } r \text{ лекцію } S_r \text{ читає викладач } p, \\ 0, & \text{в іншому випадку;} \end{cases}$$

$$\delta_{rk_rq_{kr}}^p = \begin{cases} 1, & \text{якщо в групі } k_r \text{ заняття } q_{kr} \text{ читає викладач } p, \\ 0, & \text{в іншому випадку.} \end{cases}$$

Оскільки навчальне навантаження викладача планується до складання розкладу занять, то дані величини вважаються заданими, як і аудиторне навантаження кожного викладача.

Нехай A_1 – кількість аудиторій для лекцій, A_2 – для практичних занять. У загальному випадку класифікація аудиторій може бути набагато складнішою, тоді приналежність аудиторії до певного типу визначається за якими ознаками (кількість посадочних місць, вид лабораторного обладнання, приналежність факультету або кафедри). При цьому можлива ситуація, коли та сама аудиторія входить у різні типи. У цьому випадку виникає додаткове оптимізаційне завдання рівномірного розподілу навантаження на аудиторний фонд, яке також зводиться до задачі лінійного цілісного програмування. В рамках цього завдання на основі даних про аудиторний фонд, типи аудиторій

та навантаження на них визначають частку часу, протягом якого кожному дану аудиторію слід використовувати за конкретним призначенням. При такому поданні початкових даних завдання складання розкладу полягатиме у визначенні для кожного заняття з потоком або групою дня тижня та пари в ньому з урахуванням обмежень, що накладаються на розклад. Для завдання обмежень введемо такі булеві змінні:

$$\begin{aligned}\varepsilon_j^{tk_r} &= \begin{cases} 1, & \text{якщо в день } t \text{ група } k_r \text{ починає заняття з пари } j, \\ 0, & \text{в іншому випадку;} \end{cases} \\ \eta_j^{tk_r} &= \begin{cases} 1, & \text{якщо в день } t \text{ група } k_r \text{ закінчує заняття парою } j, \\ 0, & \text{в іншому випадку;} \end{cases} \\ y_{rtj}^{tS_r} &= \begin{cases} 1, & \text{якщо в потоці } r \text{ в день } t \text{ на парі } j \text{ читається лекція } S_r, \\ 0, & \text{в іншому випадку;} \end{cases} \\ \chi_{rk_r,tj}^{qk_r} &= \begin{cases} 1, & \text{якщо в групі } k_r \text{ в день } t \text{ на парі } j \text{ проводиться} \\ & \text{практичне заняття } q_{k_r}, \\ 0, & \text{в іншому випадку.} \end{cases}\end{aligned}$$

Тоді обмеження «кожного дня на кожній парі для кожної групи може проводитися не більше одного заняття» набуде вигляду (2.5):

$$\sum_{q_{k_r}=1}^{Q_{k_r}} \chi_{rk_r,tj}^{qk_r} - \sum_{S_r=1}^{S_r} r_{rtj}^{S_r} \leq 1, \forall r = 1, \dots, R, \forall k_r = 1, \dots, G_r, \forall t \in T_{k_r}, \forall j = 1, \dots, J \quad (2.5)$$

Інша обов'язкова умова складання розкладу для навчальних закладів (2.6): для кожної групи k_r повинні виконуватися всі види аудиторної роботи протягом тижня виглядає так: k_r

$$\sum_{t \in T_{k_r}} \sum_{j=1}^J \left(\sum_{q_{k_r}=1}^{Q_{k_r}} \chi_{rk_r,tj}^{qk_r} + \sum_{S_r=1}^{S_r} y_{rtj}^{S_r} \right) = W_{k_r}, \forall r = 1, \dots, R, \forall k_r = 1, \dots, G_r \quad (2.6)$$

У конкретних завданнях список обмежень може бути продовжено. Після складання обмежень необхідно вибрати критерій оптимальності розкладу. Це досить складне питання через принципову багатокритеріальність завдання складання розкладів. Наприклад, можна розглядати згортку критеріїв, зокрема, критерій максимізації зваженої кількості вільних від аудиторної роботи днів всім викладачів. Це при

фіксованій довжині робочого тижня еквівалентно максимальному сукупному ущільненню аудиторного навантаження. Тоді критерій набуде вигляду (2.7):

$$\sum_{t=1}^T \sum_{p=1}^P \varepsilon_p z_t^p \rightarrow \max, \quad (2.7)$$

де T і P – кількість робочих днів ЗВО та кількість викладачів відповідно; ε_p – ваговий коефіцієнт, який визначається статусом викладача; z_t^p – булева змінна, що приймає значення «0», якщо викладач має заняття в даний день і «1» в іншому випадку.

Побудована таким чином модель відображатиме основні фактори, що враховуються при складанні розкладу, і відноситься до класу завдань лінійного булева програмування, рішення яких може бути знайдено, наприклад, методом гілок та границь. Вирішення таких завдань навіть щодо малої розмірності, характерної для невеликих навчальних закладів, як правило, потребує величезних витрат часу. Однак у деяких випадках це може бути необхідною платою за дотримання всіх обмежень.

У деяких випадках можливе розбиття задачі на систему підзавдань суттєво меншої розмірності. Такий підхід був використаний у роботі [20]. Тут завдання складання розкладу було розбито на систему з п'яти підзадач, які в сітці розкладу занять відображають розміщення:

- дня самостійної позааудиторної роботи студентів;
- занять з фізичного виховання або інших видів занять, які проводять позааудиторний фонд ЗВО;
- навантаження кожного потоку днями тижня;
- навантаження викладачів по робочих днях з урахуванням видів занять для потоків та груп;
- взаємозв'язків «навчальна дисципліна – викладач» за номерами пар занять кожного дня.

У машинному варіанті вдалося отримати досить компактний робочий тиждень. Однак, ця модель містить ряд недоліків, пов'язаних як з неповною адекватністю представленого рішення досліджуваної предметної задачі, так і

значною трудомісткістю використання запропонованого комплексу програм, що потребує кваліфікованого користувача [19].

Алгоритм методу імітації відпалу

Ідея алгоритму імітації відпалу запозичена з досліджень поведінки атомів металу у процесі його відпалу. У математичній моделі завдання складання розкладу під станом розуміється певний поточний варіант розкладу. Роль енергії може грати цільова функція, заснована на штрафах, що додаються до поточного розкладу за кожен незручний момент, а як низькоенергетичний стан – коректний (хоч і невідомий) розклад. Наведемо схему укрупненого алгоритму методу відпалу. Позначимо через X – поточне розв’язання задачі. Алгоритм імітації відпалу для завдання складання розкладу можна надати наступній послідовності етапів:

Етап 1. На першій ітерації генерується деяке коректне початкове розклад X^0 , тобто X^0 вважається поточним розв’язанням задачі.

Етап 2. Задаються початкове, високе значення температури T_0 та операції мутації (зміни) розкладу. Мутація розкладу може бути забезпечена такими діями:

- зміна часу проведення заняття;
- зміна аудиторії проведення заняття;
- обмін місцями у розкладі двох занять.

Етап 3. На основі введених операцій мутації та поточного рішення X генерується новий коректний розклад X' , який трохи відрізняється від попереднього.

Етап 4. Обчислюється зміна цільової функції: $\Delta f = f(X') - f(X)$

- якщо $\Delta f < 0$ (рішення не погіршилося), то $X = X'$, тобто. новий варіант розкладу стає поточним;
- якщо $\Delta f > 0$ (рішення погіршилося), то новий розклад стає поточним лише з ймовірністю $p = e^{-\Delta f / T} (1 - p)$. А з ймовірністю $(1 - p)$ попередній розклад зберігається як поточний.

На цьому етапі допускається збільшення цільової функції, але ймовірність цього зменшується зі зростанням Δf . Це дозволяє долати локальні екстремуми.

Етап 5. Обчислюється функція зміни температури. Температура i , відповідно, ймовірність прийняти як поточний розклад X розклад з великим значенням цільової функції зменшується з кожною ітерацією. Величину зменшення температури можна пов'язати як із числом виконаних ітерацій, і з величиною зміни цільової функції, тобто. сформулювати критерій зупинки. При високих температурах спостерігається стрибкоподібна зміна цільової функції. Поступово із зменшенням температури її значення має прагнути до мінімуму.

Етап 6. Якщо не виконано критерій зупинки, то перейти до п. 3. Як критерій зупинки можуть бути прийняті такі правила:

- виконання заданої кількості ітерацій;
- виконання заданого числа ітерацій без покращення значення цільової функції на задане значення.

Збереження m кращих рішень, а також n останніх розкладів, що згенерували, підвищить ефективність алгоритму і дозволить запобігти зацикленню процесу складання розкладу. Точність одержуваного алгоритмом рішення можна збільшити рахунок повільнішого зміни температури. При цьому алгоритм ретельніше досліджує простір пошуку, але час його роботи істотно збільшується. Тому функцію температури доводиться підбирати кожної конкретної завдання індивідуально.

Незважаючи на зовнішню простоту, такий підхід може бути цілком ефективним для складання невеликих розкладів.

Алгоритм методу розмальовки графа

Завдання складання розкладу можна як завдання розмальовки графа. Завданням розмальовки графа називають пошук мінімального числа кольорів (хроматичного числа графа), необхідні розмальовки вершин деякого графа те щоб кожна пара сусідніх вершин була пофарбована у різні кольори. Сама

задача пошуку хроматичного числа є важким завданням, для вирішення якої в більшості випадків використовуються різні жадібні алгоритми. Для встановлення завдання складання розкладу як завдання розмальовки графа будується граф. Кожна вершина графа – це заплановане навчальним планом заняття. Якщо між якимись двома вершинами можливі конфлікти, наприклад, обидва заняття проводяться в одній аудиторії або з одним викладачем, то вони з'єднуються ребром, що еквівалентно забороні одночасного проведення цих занять. Завдання складання розкладу може бути сформульована як завдання мінімізації числа кольорів, необхідні розмальовки графа. Кожен колір відповідає одному періоду розкладу. Завдання розмальовки графа як самостійна мало ефективна при складанні розкладів, але може бути корисним у разі її комбінації з іншими алгоритмами [19].

Алгоритм імітаційного моделювання

Для вирішення завдання складання розкладу можна спробувати імітувати дії диспетчера під час складання розкладу. У цьому випадку алгоритм оперує безпосередньо розкладом та списком занять, які необхідно включити до розкладу відповідно до навчального плану. Починається процес складання розкладу з порожнього розкладу. Це початковий незакінчений розклад. Усі заняття знаходяться у списку неврахованих занять. Далі алгоритм переходить від одного незакінченого розкладу до іншого, прагнучи найкраще розставити всі заняття, включені до списку. Процес триває доти, доки не буде сформовано повний розклад або виконається фіксована кількість ітерацій. Наведемо схему цього алгоритму:

Етап 1. Генерується незакінчений розклад (спочатку це порожній розклад).

Етап 2. Вибирається ще не включене до розкладу заняття з урахуванням аналізу «вузьких місць». Тут «вузькими місцями» є найдефіцитніші ресурси: студенти, викладачі та аудиторії. Насамперед складаються розклади для найбільш дефіцитних ресурсів. Це можуть бути

заняття, що використовують дефіцитний аудиторний фонд, заняття, що проводяться викладачами, які ставлять жорсткі умови за часом та місцем їх проведення тощо [23].

Етап 3. Для обраного заняття визначаються всі можливі варіанти його розміщення у розкладі, що задовольняють усім жорстким обмеженням. Потім кожна позиція оцінюється за допомогою спеціальної евристичної цільової функції і заняття поміщається в кращу з можливих позицій.

Етап 4. Якщо у випадку етапу 3 у розкладі виникла конфліктна ситуація, то «заняття, що конфліктують», видаляються з розкладу і поміщаються назад до списку неврахованих занять. При реалізації алгоритму імітаційного моделювання особливу увагу слід приділити розробці евристичних правил вибору чергового заняття зі списку, визначення найкращої йому позиції у розкладі та оцінці одержуваного розкладу. Позитивною особливістю такого підходу є можливість детального обліку специфіки завдання, що розв'язується, у разі складання розкладу для конкретного ЗВО. Проте можливість застосування розробленої системи інших навчальних закладах у своїй дуже обмежена. Також за незначних внутрішніх змін у ЗВО можливо треба буде вносити суттєві зміни до алгоритму.

Використовуючи алгоритм, що ґрунтується на діях диспетчера при складанні розкладу, можна організувати діяльний діалог між користувачем та системою при пошуку оптимального розкладу. Це може бути вірним лише для відносно невеликих завдань, тому що в іншому випадку значна залежність алгоритму від користувача може зробити такий діалог малоефективним.

Метод логічного програмування в обмеженнях

Складання розкладу можна як завдання задоволення обмежень, на вирішення яких розроблено безліч алгоритмів. Виник цілий напрямок у програмуванні - програмування в обмеженнях (constraint programming). Програмування в обмеженнях тісно пов'язане з традиційним логічним

програмуванням, у якого воно і сформувалося. Більшість систем програмування в обмеженнях є звичайним інтерпретатором Прологу з вбудованим механізмом для вирішення певного класу завдань задоволення обмежень. Програмування таких системах називають логічним програмуванням в обмеженнях (Constraint Logic Programming або CLP). Основна ідея вирішення завдань така: програміст визначає кілька змінних x_1, \dots, x_n і області їх значень X_1, \dots, X_n , описує додаткові обмеження, яким повинні задовольняти змінні, а система знаходить відповідні значення змінних, що задовольняють одночасно всім заданим обмеженням. Для ілюстрації наведемо невеликий приклад, який належить до досліджуваної предметної області. Розглянемо вищ, у якому працює M викладачів, у навчальному плані зафіксовано N предметів, аудиторний фонд складається з L аудиторій. Визначимо множину $P = 1, 2, \dots, DD$, елементами якого є всі періоди навчання (навчальні пари) у ЗВО протягом тижня, а кількість усіх періодів навчання протягом тижня. аудиторний фонд складається із аудиторій.

Позначимо y_{ij} – період навчання, i – викладач веде j предмет, $y_{ij} \in P$. Тоді обмеження «кожен викладач у кожний момент часу може вести не більше одного заняття» (2.8) набуде вигляду:

$$y_{ij} \neq y_{ij'}, 1 \leq i \leq M, 1 \leq j \leq N, 1 \leq j' \leq N, j \neq j'. \quad (2.8)$$

Нехай z_i - аудиторія, де проводить заняття i викладач, $1 \leq z_i \leq L$. Тоді обмеження «у кожній аудиторії в кожний момент часу може проводитися не більше одного заняття» (2.9) набуде вигляду:

$$z_i \neq z_{i'}, 1 \leq i \leq M, 1 \leq i' \leq M, i \neq i'. \quad (2.9)$$

Аналогічно складаються інші обмеження.

В результаті роботи алгоритму буде отримано множину значень кожної змінної, що задовольняють задані обмеження. При цьому область визначення змінних, що беруть участь у строгих обмеженнях, може суттєво скоротитися або навіть містити єдине значення.

Основна перевага, що отримується при використанні CLP, – це скорочення простору пошуку, що досягається не шляхом оцінки кожного варіанта розкладу, а за рахунок того, що система сама виключає з розгляду «дороги, які ведуть у глухий кут».

Генетичні алгоритми

Наведені вище методи переважно використовують ітераційну техніку поліпшення результатів. При виконанні однієї ітерації вони шукають рішення, найкраще на околицях поточного рішення, отриманого на попередній ітерації. Якщо таке рішення знайдено, воно стає поточним та починається нова ітерація. Це триває до того часу, доки виконається правило зупинки: приріст цільової функції не зменшиться майже до нуля чи виконається задану кількість ітерацій. Звісно, такі методи здійснюють пошук лише локальних оптимумів, причому становище знайденого оптимуму залежить від стартової точки, а глобальний оптимум можна знайти лише випадково. Щоб підвищити можливість перебування глобального оптимуму, пошук проводиться багаторазово з різними початковими точками. Час пошуку при цьому значно збільшується.

Тому цікавить розробка алгоритмів, що зберігають переваги описаних методів і вільних від зазначеного недоліку. До таких алгоритмів належать генетичні алгоритми.

Генетичні алгоритми – це стохастичні евристичні оптимізаційні методи, основну ідею яких взято з теорії еволюційного розвитку видів [20]. Основним механізмом еволюції є природний відбір.

Першим кроком розробки математичної моделі, заснованої на генетичному алгоритмі, є розробка структури хромосоми, у якій зберігатиметься рішення. У нашому випадку такою "хромосомою" є розклад. Вибрана структура повинна враховувати всі особливості та обмеження, що пред'являються до шуканого рішення. У кінцевому рахунку, вибір «хромосоми» впливає як на швидкість, а й збіжність алгоритму взагалі.

Одним з найбільш зручних уявлень розв'язання задачі, що розглядається, є тривимірна матриця, по осях i, j, k якої відкладаються відповідно групи, навчальний годинник і аудиторії. Елементом матриці є запит на проведення заняття з i -ю групою даним викладачем з даної дисципліни в j -ту годину в k -й аудиторії. Змістовно запит на проведення заняття є наступним: на етапі завдання вихідних даних користувач для кожної групи (підгрупи, потоку) вказує дисципліни та викладачів, які будуть вести їх у рамках передбаченого навчального плану. Після цього кожна зв'язка дисципліна-група-викладач розглядається як єдине ціле за умови, що в алгоритм введено інформацію: які дисципліни, в якій групі, який викладач і який тип аудиторій для цього підходить [20].

Така структура хромосоми зручна тим, що на етапі завдання початкових даних можна виключити свідомо невдалі рішення, заблокувавши відповідні осередки.

На наступному етапі алгоритму створюється початкова популяція, розмір якої залежить від розмірності завдання і зазвичай становить кілька сотень рішень.

Для організації процесу, що оптимізує, необхідно створити спрямовуючу силу розвитку популяції. Як така сила виступає вимога мінімізації цільової функції (у термінах генетичних алгоритмів, – фітнес-функції). Як фітнес-функція можна використовувати адитивний показник оптимальності, заснований на штрафах, що встановлюються кожному рішенню за якийсь незручний момент у розкладі. Важлива особливість такого вибору – можливість налаштувати алгоритм вирішення конкретного завдання. Це досягається варіюванням коефіцієнтів, що призводить до зміни пріоритетів при пошуку оптимального розкладу.

Наведемо алгоритм кожної ітерації. Етапи ітерації генетичного алгоритму:

Етап 1. Кожна особина популяції оцінюється з допомогою цільової функції (фітнес-функції).

Етап 2. Найкращі рішення (зазвичай близько 5%) копіюються у нову популяцію без зміни. Такий принцип, званий принципом елітизму, запобігає втратам кращих рішень і забезпечує підвищену збіжність алгоритму.

Етап 3. Проводиться пропорційний відбір двох рішень з поточної популяції, які піддаються рекомбінації. Для цього хромосоми батьків обмінюються відповідними ділянками.

Етап 4. Отриманий у попередньому етапі розклад може бути некоректним, наприклад, не відповідати навчальному плану. Тоді можна повторювати операцію рекомбінації доти, доки не буде отримано коректний розклад, але бажано передбачити евристичні механізми виправлення розкладу.

Етап 5. Якщо нова популяція сформована, то стара видаляється, після чого переходимо до етапу 1. Інакше переходимо до етапу 3.

Наведений алгоритм є стійким до локальних мінімумів. Завдяки внутрішньому паралелізму (робота не з окремими рішеннями, а з цілими класами рішень) він забезпечує швидкий пошук оптимального рішення.

Висновки до розділу 2

Математичне моделювання завдання складання розкладу дозволило виділити обов'язкові та необов'язкові вимоги, що пред'являються до розкладу, що генерується. З виконаного аналізу основних проблем автоматизації сформульовані методи, необхідні автоматизації процесу складання розкладу.

Дослідження основних алгоритмів, що застосовуються у задачі, виявило їх сильні та слабкі сторони. На основі проведеного аналізу результату даного дослідження розробки автоматичного модуля генерації розкладу обраний генетичний алгоритм.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ АВТОМАТИЧНОГО СКЛАДАННЯ РОЗКЛАДУ ЗАНЯТЬ

3.1 Вибір засобів програмування розробки програми

На підставі вибору архітектури програмного комплексу, завдань, які має вирішувати інформаційна система складання розкладу занять, необхідно вибрати мову програмування та технології для створення веб-системи.

Основними вимогами до програмного модуля можна назвати можливість автоматичної генерації розкладу занять з урахуванням жорстких та м'яких обмежень та можливість виведення даних з БД для надання відомостей студентам та викладачам про розклад занять.

Для реалізації модуля автоматичного складання розкладу було обрано мову програмування JAVA з використанням технологій Maven [20], Spring [25], Hibernate [26] та Angular [27]. Як середовище розробки вибрано IntelliJ IDEA. До роботи з БД підключений плагін Flyway [24]. Документування коду виконано засобами бібліотеки Swagger [28].

Maven

Maven – це інструмент для складання Java проєкту: компіляції, створення jar та war файлів, створення дистрибутива програми, генерації документації.

Основними перевагами Maven є те, що він не залежить від ОС. Складання проєкту відбувається у будь-якій операційній системі. Також він дозволяє використовувати у проєкті сторонні бібліотеки й у разі конфліктів версій виправляти їх, тобто. здійснювати управління залежностями. А використання командного рядка для автоматичного складання проєктів спрощує розгортання програми на сервері. Maven за допомогою конфігураційного файлу дозволяє працювати з проєктом у будь-якому середовищі розробки. Єдиний настроювальний файл для середовища

розробки та збирання дозволяє уникати дублювання даних та зменшувати кількість помилок під час компіляції.

Вся структура проєкту Maven описується у файлі: `pom.xml` (POM – Project Object Model), який знаходиться у кореневій папці проєкту.

Ключовим поняттям Maven є артефакт - це, по суті, будь-яка бібліотека, що зберігається в репозиторії (місце зберігання).

Залежності - це бібліотеки, які безпосередньо використовуються у вашому проєкті для компіляції коду або його тестування.

Плагіни використовуються самим Maven при складанні проєкту або для якихось інших цілей (деплоймент, створення файлів проєкту для Eclipse та ін).

Архетип — це стандартне компонування файлів і каталогів у проєктах різного роду (веб, swing-проєкти та інші). Іншими словами, Maven знає, як зазвичай будуються проєкти і відповідно до архетипу створює структуру каталогів.

Як правило, назва артефакту складається з назви групи, власної назви та версії. Прописана бібліотека із зазначенням артефакту у файлі `pom.xml` автоматично підвантажується Maven у репозиторій і підключається до проєкту. Конфігураційний файл проєкту представлено у додатку А.

Hibernate

Hibernate є інструментом об'єктно-реляційного відображення Java з відкритим вихідним кодом. Він надає фреймворк для відображення об'єктно-орієнтованої моделі даних традиційні реляційні бази даних. Hibernate перетворює класи Java в таблиці БД (і типи даних Java в типи даних SQL), а також дозволяє здійснювати запити і пошук БД, скорочуючи час розробки.

Зіставлення Java-класів з таблицями БД здійснюється з допомогою конфігураційних XML файлів, або з допомогою Java анотацій. Забезпечуються можливості з організації відносини між класами один-багатьом і багатьом-багатьом. На додаток до управління асоціації між

об'єктами, Hibernate може також керувати рефлексивними відносинами, де об'єкт має зв'язок один з багатьма іншими екземплярами свого типу.

Hibernate має інтуїтивно зрозумілий API до виконання запитів до об'єктів, які у базі даних. Взаємодія здійснюється за допомогою анотацій. Приклад реалізації Hibernate-моделі представлено у додатку Б.

Spring framework

Spring framework є універсальною платформою з відкритим вихідним кодом для Java.

Spring забезпечує комплексну модель розробки та конфігурації для сучасних бізнес-додатків на Java – на будь-яких платформах. Ключовий елемент Spring - підтримка інфраструктури на рівні програми: основна увага приділяється "водопроводу" бізнес-додатків, тому розробники можуть зосередитися на бізнес-логіці без зайвих налаштувань залежно від середовища виконання.

Spring Framework має модульну структуру і представлений у вигляді фреймворку у фреймворку. При цьому практично всі фреймворки можуть працювати самостійно незалежно один від одного. Але найбільшу функціональність вони демонструють при їх спільному використанні.

Основні модулі Spring Framework:

- Inversion of Control-контейнер конфігурує компоненти програми та керує життєвим циклом Java-об'єктів.
- Модуль аспектно-орієнтованого програмування працює з функціональністю, реалізація якої неможлива засобами об'єктно-орієнтованого програмування Java без втрат.
- Модуль доступу до даних працює із системами управління реляційними базами даних, використовує JDBC та ORM.
- Фреймворк керування транзакціями відповідає за роботу API керування транзакціями та інструментарію керування транзакціями для об'єктів Java.

- Фреймворк MVC є каркасом для Spring додатків, заснований на HTTP та сервлетах, багатофункціональний та розширюємо залежно від вимог розробника.

- Модуль дистанційного доступу відповідає за передачу Java-об'єктів через мережу в стилі RPC і підтримує RMI, CORBA, HTTP-based протоколи, включаючи web-сервіси (SOAP).

- Модуль аутентифікації та авторизації, що налаштовується та підтримує різні стандарти протоколів, інструментів, практик через дочірній проєкт Spring Security.

- Фреймворк дистанційного керування надає можливість керування Java-об'єктами для локальної або віддаленої конфігурації за допомогою JMX.

- Модуль роботи з повідомленнями здійснює реєстрацію об'єктів-слухачів повідомлень для прозорості обробки повідомлень із черги повідомлень за допомогою JMS за допомогою JMS API.

- Модуль тестування є каркасом, що підтримує класи для написання модульних та інтеграційних тестів.

Spring містить безліч додаткових модулів, кожен із яких можна підключити до проєкту за допомогою конфігураційного файлу.

Для реалізації програмного модуля ручного складання розкладу вибрано конфігурацію Spring Boot.

Spring Boot – проєкт Spring Framework, який спрямований на спрощення створення корпоративних додатків. Він є плагіном для системи автоматизації складання проєктів (підтримує Maven і Gradle). Основна вигода Spring Boot - конфігурування ресурсів виходячи із змісту classpath. Якщо pom.xml файл Maven проєкту містить JPA залежності, то Spring Boot підвантажить їх у проєкт. Проєкт необхідної конфігурації також можна зібрати за допомогою Spring Initializr – веб-конфігуратора проєктів.

Angular

Angular - це платформа, фреймворк для створення клієнтських програм на HTML і TypeScript. Angular написано на TypeScript. Він реалізує основну та необов'язкову функціональність у вигляді набору бібліотек TypeScript, які ви імпортуєте до своїх програм.

Основними модулями Angular є NgModules, які відповідають за компіляцію компонентів. NgModules збирають пов'язаний код у функціональні набори; Angular додаток визначається набором NgModules. У додатку завжди є принаймні кореневий модуль, який включає автозавантаження, і зазвичай має набагато більше функціональних модулів:

- компоненти визначають уявлення (views), які є набори елементів екрана, які Angular може вибирати і змінювати відповідно до логікою та даними вашої програми;
- компоненти використовують послуги (services), які надають певні функціональні можливості, не пов'язані безпосередньо з уявленнями. Постачальники послуг можуть бути впроваджені в компоненти як залежності, що робить ваш код модульним, повторно використовуваним та ефективним.

І компоненти, і сервіси – це просто класи з декораторами, які відзначають їх тип та надають метадані, які повідомляють Angular, як їх використовувати.

Метадані для класу компонента пов'язують його із шаблоном, який визначає уявлення. Шаблон поєднує звичайний HTML з директивами Angular і розміткою прив'язки, які дозволяють Angular змінювати HTML перед його відображенням.

Метадані для класу обслуговування надають інформацію, необхідну Angular, щоб зробити її доступною для компонентів за допомогою залежностей (DI).

Компоненти програми зазвичай визначають безліч уявлень, розташованих ієрархічно. Angular пропонує сервіс Router, який допоможе

вам визначити шляхи навігації між уявленнями. Маршрутизатор забезпечує складні навігаційні можливості у браузері.

Angular NgModule відрізняються від модулів JavaScript (ES2015) та доповнюють їх. NgModule оголошує контекст компіляції для набору компонентів, виділений для домену програми, робочого процесу або тісно пов'язаного набору можливостей. NgModule може зв'язувати свої компоненти зі зв'язаним кодом, таким як послуги, для формування функціональних блоків.

Кожна програма Angular має кореневий модуль, умовно званий AppModule, який забезпечує механізм початкового завантаження, що запускає програму. Програма зазвичай містить багато функціональних модулів.

Як і модулі JavaScript, NgModules можуть імпортувати функціональність з інших NgModules, і дозволяють експортувати та використовувати їх власні функції іншими NgModules. Наприклад, щоб використовувати службу роутера у вашій програмі, ви імпортуєте Router NgModule.

Організація коду в окремі функціональні модулі допомагає в управлінні розробкою складних програм та розробці для повторного використання. Крім того, цей метод дозволяє використовувати переваги відкладеного завантаження, тобто завантаження модулів на вимогу, щоб мінімізувати обсяг коду, який необхідно завантажити під час запуску.

3.2 Опис кодів програми модуля генерації розкладу занять з урахуванням генетичного алгоритму

На основі вихідних даних, вимог до процесу складання розкладу, вимог до програмного забезпечення для реалізації проєкту обрано архітектуру клієнт-сервер.

Структурно проєкт розділений на дві частини: серверна частина на SpringBoot, що відповідає за логіку програми; клієнтська частина на Angular відповідає за інтерфейс користувача.

Проєкт серверної частини розроблено у концепції MVC (Model-View-Controller). Для взаємодії з клієнтським проєктом та для зручності розробки реалізовано REST API, що збільшує можливості для frontend-розробки: за потреби можна використовувати будь-які технології для взаємодії з користувачем та виведення інформації.

Зберігання інформації реалізовано з використанням реляційної моделі даних під керуванням СУБД PostgreSQL. Розроблена база даних дозволяє зберігати основні відомості із навчальних планів, навантажень, списків груп, кафедр, аудиторій тощо. необхідні автоматичної генерації розкладу занять.

Взаємодія серверного проєкту з БД здійснюється засобами бібліотек: Flyway - для управління безпосередньо структурою БД і таблиць (додаток В), Spring Data JPA - для управління даними програми, Hibernate - для перетворення таблиць БД Java класи і навпаки (додаток Г).

Для зручності розробки та супроводу введено автоматичне документування коду за допомогою бібліотеки Swagger.

Алгоритм роботи модуля автоматичної генерації розкладу

На запит користувача система складання розкладу на підставі даних з таблиць навчального плану групи та навантаження викладачів генерує необхідну кількість занять, функцію генерації занять наведено у лістингу 3.1.

Лістинг 3.1 - Функція генерування занять

Листинг 3.1 – Функция генерирования занятий

```
@Override
public List<Pair> generate(Groups groups) {

    init(groups);

    for (Subject subject : this.subjectList) {
```

```

        List<LecturerLoads> lecturerLoads =
lecturerLoadsService.getBySubject(subject);

for (LecturerLoads lecturerLoad : lecturerLoads) {

int pairsCount = lecturerLoad.getHours() / 2;

for (int i = 0; i < pairsCount; i++) {

        Pair pair = new Pair();
        pair.setGroups(this.groups);
        pair.setSubject(subject);

        pair.setLoadTypes(lecturerLoad.getLoadTypes());
        pair.setLecturer(lecturerLoad.getLecturer());

pairService.create(pair);
    }

}

this.pairsList = pairService.getByGroups(groups);

return this.pairsList;
}

```

Згенерований на даному етапі список занять дозволяє Користувачеві почати розміщувати пари самостійно або скористатися функцією автоматичного розміщення занять (листинг 3.2).

Листинг 3.2 – Функція розміщення занять

```

@Override
public List<Pair> locate(Groups groups) {

    init(groups);

    for (Subject subject : this.subjectList) {
        this.weeksList = weeksService.getWeeksByGraphCourses(graphCourses);
        int pairsCountOnWeek =
calculatePairsCountOnWeek(pairService.getByGroupsAndSubject(groups,
subject).size(), this.weeksList.size());
        for (Weeks week : this.weeksList) {

            locatePair(pairsCountOnWeek, groups, subject, week);

        }

    }

    this.pairsList = pairService.getByGroups(groups);

    return this.pairsList;
}

```

Представлена вище функція, аналізуючи список занять, визначає кількість пар на тиждень та розміщує їх у сітку розкладу понеділок з урахуванням різних типів навантаження. Для зручності використання

реалізовано навантаження даного методу: користувач може запустити авторозміщення як на семестр так і на певний тиждень, що додає додатку гнучкості, так як за стандартної поведінки система розподіляє пари рівномірно.

Кінцеве розміщення пар здійснюється генетичним алгоритмом при виклику функції `autolocate`, лістинг 3.3.

Лістинг 3.3 - Функція автоматичного розміщення пари генетичним алгоритмом

```
public void autolocation(List<Pair> pairs) {
    for (Pair pair : pairs) {
        Integer evolution = 1;

        List<Chromosome> population = getFirstPopulation(pair, 5, 4);

        for(int i = 0; i < evolution; i++) {
            population = fitness(population, pair);
            population = selection(population);
            population.addAll(fitness(crossing(population), pair));
        }

        Chromosome elite = population.get((new
Random()).nextInt(population.size()));

        pair.setWeekday(elite.getDay());
        pair.setTimetable(timetableService.one(elite.getTime().longValue()));
        pair.setAuditorium(auditoriumService.one(elite.getAuditorium().
longValue()));
        pair.setDate(createDateFromWeeksAndWeekday(pair.getWeeks(),
pair.getWeekday()));

        pairService.update(pair.getId(), pair);
    }
}
```

Для зручності представлений вище метод перевантажений і може бути викликаний користувачем безпосередньо з параметром `id` групи для повністю автоматизованої роботи програми: генерації занять, понеділкового розміщення та розподілу пар у сітку розкладу, а також за допомогою поетапного виклику описаних раніше функцій.

На вхід функція приймає перелік розподілених потижнево пар. Для кожної пари генерується первинна популяція, приклад функції генерації первинної популяції наведено у лістингу 3.4.

Лістинг 3.4 - Функція генерування первинної популяції

```
public List<Chromosome> getFirstPopulation(Pair pair, int weekdays, int
timetables) {
```



```

List<Auditorium> auditoriums = auditoriumService.all();

List<Chromosome> chromosomes = new ArrayList<>();

for (Auditorium auditorium : auditoriums) {
    for (int day = 1; day < (weekdays + 1); day++) {
        for (int time = 1; time < (timetables + 1); time++) {

            Chromosome chromosome = new Chromosome();
            chromosome.setDay(day);
            chromosome.setTime(time);
            chromosome.setAuditorium(auditorium.getId().intValue());
            chromosomes.add(chromosome);

        }
    }
}
return chromosomes;
}

```

Для зручності розробки реалізований клас Chromosome (додаток Д), що містить 4 параметри: день, час, аудиторія та змінна для зберігання результатів оцінки особи фітнес-функцією.

Популяція для пари генерується за принципом перебору всіх доступних варіантів розміщення з урахуванням дня тижня, аудиторії та часу проведення пари. На цьому етапі заповнюються 3 із 4 доступних параметрів.

Далі отримана популяція оцінюється фітнес-функцією (листинг 3.5). В основі функції лежить система оцінювання балами. За дотримання обов'язкових вимог до розкладу – відсутності накладок у зв'язці заняття, група, аудиторія – додається 15 балів до якості.

Для врахування додаткових вимог викладачів, груп та дисциплін та коректної роботи фітнес-функції розроблено таблицю обмежень (додаток Е).

Лістинг 3.5 - Функція оцінки якості особи популяції

```

public List<Chromosome> fitness(List<Chromosome> chromosomeList, Pair pair) {
    List<Limitations> limitationsList =
        limitationsService.getByLecturerOrSubjectOrGroups(pair.getLecturer(),
            pair.getSubject(), pair.getGroups());

    for (int i = 0; i < chromosomeList.size(); i++) {
        List<Pair> lecturerPairsOnTimetable =
            pairService.getByLecturerAndWeeksAndWeekdayAndTimetable
                (pair.getLecturer(), pair.getWeeks(), chromosomeList.get(i).getDay(),
                    timetableService.one(chromosomeList.get(i).getTime().longValue()));
        List<Pair> groupsPairsOnTimetable =
            pairService.getByGroupsAndWeeksAndWeekdayAndTimetable(pair.getGroups(),
                pair.getWeeks(), chromosomeList.get(i).getDay(),
                    timetableService.one(chromosomeList.get(i).getTime().longValue()));
    }
}

```

```

        List<Pair> auditoriumPairsOnTimetable =
pairService.getByAuditoriumAndWeeksAndWeekdayAndTimetable
(pair.getAuditorium(), pair.getWeeks(), chromosomeList.get(i).getDay(),
timetableService.one(chromosomeList.get(i).getTime().longValue()));
        chromosomeList.get(i).setQuality(0);

if (lecturerPairsOnTimetable.size() == 0)

chromosomeList.get(i).setQuality(chromosomeList.get(i).getQuality() + 15);
if (groupsPairsOnTimetable.size() == 0)

chromosomeList.get(i).setQuality(chromosomeList.get(i).getQuality() + 15);
if (auditoriumPairsOnTimetable.size() == 0)

chromosomeList.get(i).setQuality(chromosomeList.get(i).getQuality() + 15);

for (Limitations limitations : limitationsList) {
switch (limitations.getType()) {
case 1:
for (String str : limitations.getAcceptable().split(",")) {
if (chromosomeList.get(i).getDay().equals(Integer.parseInt(str))) {
chromosomeList.get(i).setQuality(chromosomeList.get(i).getQuality() + 10);
}
}
break;
case 2:
for (String str : limitations.getAcceptable().split(",")) {
if (chromosomeList.get(i).getTime().equals(Integer.parseInt(str))) {
chromosomeList.get(i).setQuality(chromosomeList.get(i).getQuality() + 10);
}
}
break;
case 3:
for (String str : limitations.getAcceptable().split(",")) {
if (chromosomeList.get(i).getAuditorium().equals(Integer.parseInt(str))) {
chromosomeList.get(i).setQuality(chromosomeList.get(i).getQuality() + 10);
}
}
break;
}
}
}

return chromosomeList;
}

```

При виконанні кожної додаткової вимоги розклад додається 10 балів.

Далі популяція піддається селекції, у результаті якої залишається 25% осіб. Вибір здійснюється за принципом «елітизму». Наступним етапом виконується операція кросовера: випадковим чином здійснюється вибір 2 осіб для схрещування. Схрещування може проводитись за кількома точками. За умовчанням застосовується одноточкове схрещування. Приклад функції представлений у лістингу 3.6.

Лістинг 3.6 – Функція схрещування

```

public List<Chromosome> crossing(List<Chromosome> population) {
    List<Chromosome> bestPopulation = new ArrayList<>();

```

```

        Integer bestPopulationsCount = Math.floorDiv((population.size() * 25),
100);
    for (int i = 0; i < bestPopulationsCount; i++) {

        Random random = new Random();
        Chromosome firstChromosome =
population.get(random.nextInt(population.size()));
        Chromosome secondChromosome =
population.get(random.nextInt(population.size()));

        bestPopulation.add(new Chromosome(firstChromosome.getDay(),
secondChromosome.getTime(),
secondChromosome.getAuditorium()));
        bestPopulation.add(new Chromosome(secondChromosome.getDay(),
firstChromosome.getTime(),
firstChromosome.getAuditorium()));
    }
    return bestPopulation;
}

```

Отримані в результаті схрещування хромосоми оцінюються фітнес-функцією і додаються у вихідну популяцію. Таким чином, здійснюється генерування одного покоління. У разі невдалої генерації популяції можна застосовувати мутацію для отримання нових хромосом і як наслідок отримувати нові варіанти розміщення пари в сітці розкладу. Далі з популяції вибирається одна особина, гени якої стають параметрами пари, що розміщується.

Таким чином здійснюється автоматичний розподіл пар у розклад.

3.3 Аналіз якості автоматично згенерованого розкладу

Застосування генетичного алгоритму завдання розробки модуля автоматичної генерації розкладу дозволило прискорити і спростити процес складання розкладу. Складений програмним модулем розклад задовольняє основним вимогам: накладки у розкладі у викладачів, студентів та аудиторій відсутні; аудиторії відповідають типу занять; кількість пар щодня вбирається у максимально допустиме значення, відповідно, вважатимуться складений розклад рівномірним.

Також при розподілі враховувалися і м'які обмеження: побажання викладачів щодо проведення занять у задані дні тижня та години,

виставлення пар до аудиторії відповідно до її режиму роботи (за потреби), вимоги дисципліни до аудиторного фонду.

У ході тестування програмного модуля було виявлено значне уповільнення роботи алгоритму на етапі оцінки особи популяції фітнес-функцією.

Проведений аналіз результатів роботи розробленого модуля автоматичної генерації розкладу дозволяє сформулювати перелік необхідних рекомендацій для підвищення ефективності та якості розкладу:

- для збільшення швидкості роботи генетичного алгоритму для механізму селекції необхідно реалізувати ще кілька методів відбору, крім методу «елітизму», що використовується на даний момент;
- збільшення кількості нащадків як наслідок підвищення якості популяції, що забезпечує перебування оптимального рішення, необхідно реалізувати багатоточкове схрещування;
- для досягнення універсальності застосування модуля автоматичної генерації розкладу необхідно розробити модуль конфігурації, який дозволить кінцевому Користувачеві налаштувати додаток під обмеження конкретного ЗВО або кафедри;
- для зручності та ефективності модуля складання розкладу розробити додатковий модуль для корекції навчального навантаження викладачів.

Висновки до розділу 3

У процесі розробки системи складання розкладу занять було вивчено та застосовано сучасні мультиплатформні технології програмування на основі мови Java. Спроектовано базу даних для зберігання інформації. Реалізовано модуль автоматичної генерації та розподілу пар на основі генетичного

алгоритму з урахуванням жорстких та м'яких вимог, що пред'являються до розкладу.

Програмний модуль автоматичної генерації може бути використаний для складання розкладу занять в окремому ЗВО. Серед переваг можна назвати можливість генерації прийнятних варіантів розкладу з першої ітерації.

В алгоритмі передбачена можливість значного покращення розкладу завдяки доданню додаткових критеріїв оцінки свободи та якості розташування занять у розкладі.

Наявність механізму обліку додаткових вимог дає змогу підвищувати якість популяції та результатів пошуку оптимального рішення загалом.

Застосування налаштованих коефіцієнтів дозволить зробити додаток гнучким та легко адаптованим під вимоги навчального закладу.

Використання генетичного алгоритму в системі складання розкладу може значно покращити якість розкладу, що складається.

ВИСНОВКИ

У кваліфікаційній роботі проведено дослідження стану проблеми складання розкладу занять у ЗВО, яке показало, що, незважаючи на численні дослідження на цю тему, досі не існує універсальних програмних рішень, що дозволяють враховувати численні зовнішні чинники довкілля та особливості освітніх закладів. Варто відзначити, що більшість програмного забезпечення, що пропонується на ринку, є платним і має досить високу вартість.

На даний момент розроблено напівавтоматичні та автоматичні засоби, які здатні генерувати якісний розклад, але за ідеальних умов.

Насамперед дані програмні модулі не допускають виникнення накладок у розкладі викладачів та груп, при цьому автоматичне розміщення пар в аудиторії у більшості рішень не передбачено і дана дія покладається на диспетчера.

У програмних продуктах, що передбачають можливість автоматичного розміщення пар в аудиторії, виставлені жорсткі обмеження, що не дозволяють у деяких випадках повноцінно використовувати цю функцію.

Також величезним недоліком можна вважати неможливість задати точний графік роботи викладачів, які працюють за сумісництвом: є лише вказати дні тижня.

Описані у першому розділі предметна область та вимоги до розкладу, а також результати аналізу існуючого програмного забезпечення дозволили розробити математичну модель розкладу. У контексті даної моделі були виявлені обов'язкові та додаткові обмеження для формування сітки розкладу занять, а саме: відсутність накладок, розміщення груп в аудиторії, виконання кількісних норм занять на день та, відповідно, облік побажань викладачів, дотримання логічної послідовності розміщення пар освітнього компонента тощо.

Проведене дослідження методів та алгоритмів, що застосовуються при автоматизації, виявило їх сильні та слабкі сторони.

Розгляд основних проблем автоматизації процесу складання розкладу дозволив здійснити вибір оптимального алгоритму на вирішення поставленої задачі. Для розробки власного автоматизованого модуля генерації розкладу було обрано генетичний алгоритм.

На основі проведених досліджень для кодування програмного модуля було вивчено та застосовано сучасні технології розробки, спроектовано сховище інформації, як алгоритм автоматизації генерації розкладу реалізовано генетичний алгоритм.

Аналіз результатів роботи програмного модуля підтвердив практичну значущість дослідження. Складання розкладу занять за допомогою автоматичного програмного модуля генерації здійснюється протягом декількох хвилин, що в порівнянні з ручним складанням, швидше та ефективніше.

Автоматичне розміщення занять з аудиторій дозволило звільнити від додаткового навантаження диспетчера, а облік графіка роботи викладачів, вимог дисципліни до аудиторій дозволило більш раціонально та якісно генерувати розклад. Проте, залишаються невирішеними проблеми мінімізації вікон у студентів та викладачів. Також існуюча версія ядра генетичного алгоритму не є ідеальною і вимагає модернізації та оптимізації відповідно до практичних рекомендацій.

Застосування додаткових конфігураційних надбудов, коефіцієнтів та методів відбору особин для алгоритму зробить створений програмний продукт гнучким та легко адаптованим до зовнішніх умов, бо може створити розклад відповідно до обраних користувачами критеріями.

Модульна реалізація розробленої системи автоматизованого складання розкладу у структурі загальної автоматизованої системи ведення документообігу забезпечує можливість централізованого використання баз даних та впровадження загальної політики захисту інформаційного забезпечення системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. PATAT Conferences. URL: <https://patatconference.org> .
2. Снитюк В.Є. Сіпко Є.Н. Про особливості формування цільової функції та обмежень в задачі складання розкладу занять//Математичні машини і системи, 2014. № 3. С. 67–76.
3. Томашевський В.М., Новіков Ю.Л., Камінська П.А. Складання розкладів занять у дистанційних системах навчання // Вісник НТУУ «КПІ» Інформатика, управління та обчислювальна техніка, 2010. №. 52. С. 118-130.
4. Bania Kumar Rubul, Duarah, Pinkey. Exam Time Table Scheduling using Graph Coloring Approach // International Journal of Computer Sciences and Engineering, 2018. №6. Pp. 84-93.
5. Jha S.K. Exam Timetabling Problem using Genetic algorithm // International Journal of Research in Engineering and Technology. 2014. Vol.3, №5, Pp. 649-655.
6. Астахова И.Ф., Фирас А.М. Составление расписания учебных занятий на основе генетического алгоритма // Вестник ВГУ, серия: Системный анализ и информационные технологии. 2013. № 2. С. 93-99.
7. Юрчак І.Ю., Москович Т.Р. Дослідження генетичних алгоритмів та їх застосування їхнього в автоматизованій системі розподілу навантаження для викладачів і студентів. URL: <http://eom.lp.edu.ua/sntk/doc/ksm2018/moskovytch.pdf>.
8. Leite Nuno, Melicio Fernando, Rosa Agostinho. A fast simulated annealing algorithm for the examination timetabling problem // Expert Systems with Applications, 2018. Vol. 122.
9. Гусейн А. Разработка механизма интеллектуального управления отношениями «студент-преподаватель» в пространстве виртуального образования с применением нейронных сетей // Open education. V. 22. № 5. 2018. Pp. 94-103.

10. Thepphakorn T, Pongcharoen P, Hicks C. An ant colony based timetabling tool // International Journal of Production Economics, 2014, № 149(3). С. 131-144.

11. Verma O.P., Garg. R., Bisht V.S., Optimal Time-Table Generation by Hybridized Bacterial Foraging and Genetic Algorithm // Proceedings of International Conference on Communication Systems and Network Technologies (CSNT'12), (2012), Pp. 919-923.

12. ITC 2019: International Timetabling Competition. URL: <https://www.itc2019.org/home>.

13. Система для складання розкладів у ЗВО. URL: <https://osvita.ua/vnz/53319/>

14. Мулява І.Я. Система формування розкладу навчального заняття з використанням суб'єктивних переваг // Міжнародний науковий журнал, 2016.№ 7. С. 22-27.

15. Лагоша Б.А., Петропавловская А.В. Комплекс моделей и методов оптимизации описания занятий в вузе // Экономика и математические методы, 1993. № 4. С. 48-56.

16. Безгинов, А.Н Трегубов С.Ю. Обзор существующих методов составления расписаний // Информационные технологии в программировании: межвуз. сб. ст. 2005. Вып. 2(14). С.5-19.

17. Бойко О.М. Еволюційна технологія розв'язування задачі складання розкладів навчальних занять // Штучний інтелект. 2006. № 3. С. 341-348.

18. Blum C. Ant colony optimization: Introduction and recent trends // Physics of Life Reviews 2, 2005. С. 353-373.

19. Жукова М.Ю., Аль-Габри В.М. Автоматизация построения расписания экзаменов ВУЗа с использованием генетического алгоритма // Инженерный вестник Дона, 2017. №3.

20. Skakalina E. Застосування мурашиних алгоритмів в рішенні

задачі маршрутизації // Системи управління, навігації та зв'язку. Збірник наукових праць, 2019. Т. 6 (58). С. 75-83.

21. Леснік С.В., Хижняк Т.А. Застосування методу лінійної згортки для вибору джерела альтернативної енергії. URL: <http://elc.kpi.ua/old/article/download/158162/187431> .

22. Устенко С.В., Бібко О.О. Використання методу мурашиної колонії для розв'язання оптимізаційних задач // Науковий вісник НЛТУ України. Вип. 25.3, 2015. С. 351-359.

23. Apache Maven // Wikipedia. 2017. URL: https://ru.wikipedia.org/wiki/Apache_Maven.

24. Spring. JPA Repositories. Reference Documentation // Spring. 2019. URL: <http://docs.spring.io/spring-data/jpa/docs/1.4.2.RELEASE/reference/html/jpa.repositories.html>.

25. Spring Data JPA. Query methods // Spring. 2019. URL: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.query-methods>.

26. LombokDemo // Lombok. 2019. URL: <https://projectlombok.org>.

27. Flyway. Get started // Flyway. 2019. URL: <https://flywaydb.org/getstarted/>.

28. Spring Framework // Wikipedia. 2014. URL: https://ru.wikipedia.org/wiki/Spring_Framework.

29. Hibernate // Wikipedia. 2017. URL: [https://ru.wikipedia.org/wiki/Hibernate_\(библиотека\)](https://ru.wikipedia.org/wiki/Hibernate_(библиотека)).

30. Angular. Architecture overview // Angular. 2019. URL: <https://angular.io/guide/architecture>.

31. Swagger. Create great API documentation // Swagger. 2019. URL: <https://swagger.io/solutions/api-documentation>.

Конфігураційний файл проєкту - pom.xml

```
<?xmlversion="1.0" encoding="UTF-8"?>
<projectxmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.1.4.RELEASE</version>
<relativePath/><!-- lookup parent from repository -->
</parent>
<groupId>edu.its</groupId>
<artifactId>schedule</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>schedule-api</name>
<description>REST API for Schedule</description>

<properties>
<java.version>1.8</java.version>
</properties>

<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>

<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>5.4.2.Final</version>
</dependency>

<dependency>
<groupId>org.postgresql</groupId>
<artifactId>postgresql</artifactId>
<version>9.4-1206-jdbc42</version>
<scope>runtime</scope>
</dependency>

<dependency>
<groupId>org.flywaydb</groupId>
<artifactId>flyway-core</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
<version>2.1.4.RELEASE</version>
</dependency>
```

```
<dependency>
<groupId>org.projectlombok</groupId>
<artifactId>lombok</artifactId>
<optional>>true</optional>
</dependency>

<dependency>
<groupId>io.springfox</groupId>
<artifactId>springfox-swagger2</artifactId>
<version>2.9.2</version>
</dependency>
</dependencies>

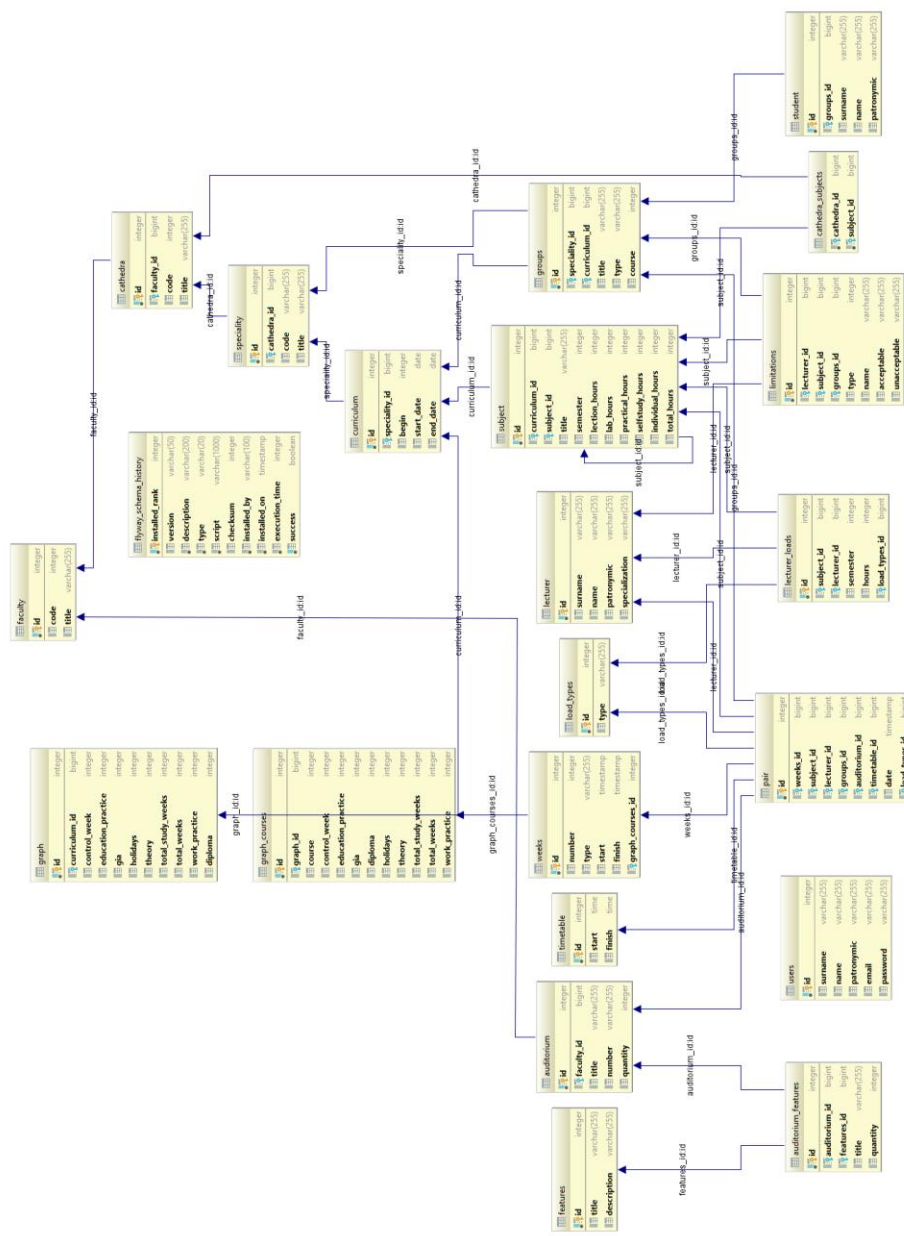
<build>
<plugins>
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
</plugin>

<plugin>
<groupId>org.flywaydb</groupId>
<artifactId>flyway-maven-plugin</artifactId>
<version>5.2.4</version>
</plugin>

<plugin>
<groupId>com.googlecode.flyway</groupId>
<artifactId>flyway-maven-plugin</artifactId>
<version>1.5</version>
</plugin>
</plugins>
</build>

</project>
```

Схема бази даних системи складання розкладу занять



SQL-скрипт генерації бази даних для Flyway

```
CREATE TABLE users
(
  id SERIAL PRIMARY KEY,
  surname VARCHAR(255),
  name VARCHAR(255),
  patronymic VARCHAR(255),
  email VARCHAR(255),
  password VARCHAR(255)
);
CREATE TABLE auditorium
(
  id SERIAL PRIMARY KEY,
  faculty_id BIGINT,
  title VARCHAR(255),
  number VARCHAR(255),
  quantity INTEGER
);
CREATE TABLE auditorium_features
(
  id SERIAL PRIMARY KEY,
  auditorium_id BIGINT,
  features_id BIGINT,
  title VARCHAR(255),
  quantity INTEGER
);
CREATE TABLE cathedra
(
  id SERIAL PRIMARY KEY,
  faculty_id BIGINT,
  code INTEGER,
  title VARCHAR(255)
);
CREATE TABLE cathedra_subjects
(
  cathedra_id BIGINT NOT NULL,
  subject_id BIGINT NOT NULL
);
CREATE TABLE curriculum
(
  id SERIAL PRIMARY KEY,
  speciality_id BIGINT,
  begin INTEGER
);
CREATE TABLE faculty
(
  id SERIAL PRIMARY KEY,
  code INTEGER,
  title VARCHAR(255)
);
CREATE TABLE features
(
  id SERIAL PRIMARY KEY,
  title VARCHAR(255),
  description VARCHAR(255)
);
CREATE TABLE graph
(
  id SERIAL PRIMARY KEY,
  curriculum_id BIGINT,
```

```

control_week INTEGER,
education_practice INTEGER,
gia INTEGER,
holidays INTEGER,
theory INTEGER,
total_study_weeks INTEGER,
total_weeks INTEGER,
work_practice INTEGER
);
CREATE TABLE groups
(
id SERIAL PRIMARY KEY,
speciality_id BIGINT,
curriculum_id BIGINT,
title VARCHAR(255),
type VARCHAR(255)
);
CREATE TABLE lecturer
(
id SERIAL PRIMARY KEY,
surname VARCHAR(255),
name VARCHAR(255),
patronymic VARCHAR(255),
specialization VARCHAR(255)
);
CREATE TABLE pair
(
id SERIAL PRIMARY KEY,
weeks_id BIGINT,
subject_id BIGINT,
lecturer_id BIGINT,
groups_id BIGINT,
auditorium_id BIGINT,
timetable_id BIGINT,
date TIMESTAMP,
type VARCHAR(255)
);
CREATE TABLE speciality
(
id SERIAL PRIMARY KEY,
cathedra_id BIGINT,
code VARCHAR(255),
title VARCHAR(255)
);
CREATE TABLE student
(
id SERIAL PRIMARY KEY,
groups_id BIGINT,
surname VARCHAR(255),
name VARCHAR(255),
patronymic VARCHAR(255)
);
CREATE TABLE subject
(
id SERIAL PRIMARY KEY,
curriculum_id BIGINT,
subject_id BIGINT,
title VARCHAR(255),
semester INTEGER,
lection_hours INTEGER,
lab_hours INTEGER,
practical_hours INTEGER,
selfstudy_hours INTEGER,
individual_hours INTEGER,

```

```

total_hours INTEGER
);
CREATE TABLE timetable
(
id SERIAL PRIMARY KEY,
start TIME,
finish TIME
);
CREATE TABLE weeks
(
id SERIAL PRIMARY KEY,
graph_id BIGINT,
number INTEGER,
type VARCHAR(255),
start TIMESTAMP,
finish TIMESTAMP
);
ALTER TABLE auditorium ADD FOREIGN KEY (faculty_id) REFERENCES faculty (id);
ALTER TABLE auditorium_features ADD FOREIGN KEY (auditorium_id) REFERENCES
auditorium (id);
ALTER TABLE auditorium_features ADD FOREIGN KEY (features_id) REFERENCES
features (id);
ALTER TABLE cathedra ADD FOREIGN KEY (faculty_id) REFERENCES faculty (id);
ALTER TABLE cathedra_subjects ADD FOREIGN KEY (cathedra_id) REFERENCES
cathedra (id);
ALTER TABLE cathedra_subjects ADD FOREIGN KEY (subject_id) REFERENCES subject
(id);
ALTER TABLE curriculum ADD FOREIGN KEY (speciality_id) REFERENCES speciality
(id);
ALTER TABLE graph ADD FOREIGN KEY (curriculum_id) REFERENCES curriculum (id);
ALTER TABLE groups ADD FOREIGN KEY (curriculum_id) REFERENCES curriculum
(id);
ALTER TABLE groups ADD FOREIGN KEY (speciality_id) REFERENCES speciality
(id);
ALTER TABLE pair ADD FOREIGN KEY (subject_id) REFERENCES subject (id);
ALTER TABLE pair ADD FOREIGN KEY (lecturer_id) REFERENCES lecturer (id);
ALTER TABLE pair ADD FOREIGN KEY (auditorium_id) REFERENCES auditorium (id);
ALTER TABLE pair ADD FOREIGN KEY (groups_id) REFERENCES groups (id);
ALTER TABLE pair ADD FOREIGN KEY (timetable_id) REFERENCES timetable (id);
ALTER TABLE pair ADD FOREIGN KEY (weeks_id) REFERENCES weeks (id);
ALTER TABLE speciality ADD FOREIGN KEY (cathedra_id) REFERENCES cathedra
(id);
ALTER TABLE student ADD FOREIGN KEY (groups_id) REFERENCES groups (id);
ALTER TABLE subject ADD FOREIGN KEY (curriculum_id) REFERENCES curriculum
(id);
ALTER TABLE subject ADD FOREIGN KEY (subject_id) REFERENCES subject (id);
ALTER TABLE weeks ADD FOREIGN KEY (graph_id) REFERENCES graph (id);

```


Код моделі сутності «Пара» – Pair.java

```
package edu.its.schedule.models;

import com.fasterxml.jackson.annotation.JsonBackReference;
import lombok.Data;

import javax.persistence.*;
import java.io.Serializable;
import java.util.Date;
import java.util.Objects;

@Data
@Entity(name = "pair")
@Table(name = "pair")
public class Pair implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "date")
    private Date date;

    @Column(name = "weekday")
    private Integer weekday;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "groups_id", foreignKey = @ForeignKey(name = "GROUPS_ID_FK"))
    @JsonBackReference
    private Groups groups;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "lecturer_id", foreignKey = @ForeignKey(name = "lecturer_ID_FK"))
    @JsonBackReference
    private Lecturer lecturer;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "subject_id", foreignKey = @ForeignKey(name = "SUBJECT_ID_FK"))
    @JsonBackReference
    private Subject subject;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "load_types_id", foreignKey = @ForeignKey(name = "load_types_ID_FK"))
    @JsonBackReference
    private LoadTypes loadTypes;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "auditorium_id", foreignKey = @ForeignKey(name = "AUDITORIUM_ID_FK"))
    @JsonBackReference
    private Auditorium auditorium;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "timetable_id", foreignKey = @ForeignKey(name =
```

```

"TIMETABLE_ID_FK"))
@JsonBackReference
private Timetable timetable;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "weeks_id", foreignKey = @ForeignKey(name =
"WEEKS_ID_FK"))
@JsonBackReference
private Weeks weeks;

public Long getId() {
return id;
}

public void setId(Long id) {
this.id = id;
}

public Date getDate() {
return date;
}

public void setDate(Date date) {
this.date = date;
}

public Integer getWeekday() {
return weekday;
}

public void setWeekday(Integer weekday) {
this.weekday = weekday;
}

public Groups getGroups() {
return groups;
}

public void setGroups(Groups groups) {
this.groups = groups;
}

public Lecturer getLecturer() {
return lecturer;
}

public void setLecturer(Lecturer lecturer) {
this.lecturer = lecturer;
}

public Subject getSubject() {
return subject;
}

public void setSubject(Subject subject) {
this.subject = subject;
}

public LoadTypes getLoadTypes() {
return loadTypes;
}

public void setLoadTypes(LoadTypes loadTypes) {
this.loadTypes = loadTypes;
}

```

```

    }

    public Auditorium getAuditorium() {
        return auditorium;
    }

    public void setAuditorium(Auditorium auditorium) {
        this.auditorium = auditorium;
    }

    public Timetable getTimetable() {
        return timetable;
    }

    public void setTimetable(Timetable timetable) {
        this.timetable = timetable;
    }

    public Weeks getWeeks() {
        return weeks;
    }

    public void setWeeks(Weeks weeks) {
        this.weeks = weeks;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Pair)) return false;
        Pair pair = (Pair) o;
        return Objects.equals(getDate(), pair.getDate()) &&
            Objects.equals(getGroups(), pair.getGroups()) &&
            Objects.equals(getLecturer(), pair.getLecturer()) &&
            Objects.equals(getSubject(), pair.getSubject()) &&
            Objects.equals(getLoadTypes(), pair.getLoadTypes()) &&
            Objects.equals(getWeekday(), pair.getWeekday());
    }

    @Override
    public String toString() {
        return "Pair{" +
            "id=" + id +
            ", date=" + date +
            ", groups=" + groups +
            ", lecturer=" + lecturer +
            ", subject=" + subject +
            ", loadTypes=" + loadTypes +
            ", auditorium=" + auditorium +
            ", timetable=" + timetable +
            ", weeks=" + weeks +
            ", weekday=" + weekday +
            '}';
    }

    @Override
    public int hashCode() {
        return Objects.hash(getDate(), getGroups(), getLecturer(), getSubject(),
            getLoadTypes(), getWeekday());
    }
}

```

Код класа Chromosome.java

```
package edu.its.schedule.models;
public class Chromosome {

    private Integer day;
    private Integer time;
    private Integer auditorium;
    private Integer quality;
    public Integer getDay() {
        return day;
    }

    public void setDay(Integer day) {
        this.day = day;
    }

    public Integer getTime() {
        return time;
    }

    public void setTime(Integer time) {
        this.time = time;
    }

    public Integer getAuditorium() {
        return auditorium;
    }

    public void setAuditorium(Integer auditorium) {
        this.auditorium = auditorium;
    }

    public Integer getQuality() {
        return quality;
    }

    public void setQuality(Integer quality) {
        this.quality = quality;
    }

    public Chromosome() {
    }

    public Chromosome(Integer day, Integer time, Integer auditorium) {
        this.day = day;
        this.time = time;
        this.auditorium = auditorium;
    }

    public Chromosome(Integer day, Integer time, Integer auditorium, Integer
quality) {
        this.day = day;
        this.time = time;
        this.auditorium = auditorium;
        this.quality = quality;
    }
}
```

Код моделі таблиці обмежень Limitations.java

```
package edu.its.schedule.models;

import com.fasterxml.jackson.annotation.JsonBackReference;
import lombok.Data;

import javax.persistence.*;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;

/**
 * Created by valerija on 26.02.19.
 */
@Data
@Entity(name = "limitations")
@Table(name = "limitations")
public class Limitations implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "type")
    private Integer type;

    @Column(name = "name")
    private String name;

    @Column(name = "acceptable")
    private String acceptable;

    @Column(name = "unacceptable")
    private String unacceptable;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "groups_id", foreignKey = @ForeignKey(name = "GROUPS_ID_FK"))
    @JsonBackReference
    private Groups groups;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "lecturer_id", foreignKey = @ForeignKey(name = "lecturer_ID_FK"))
    @JsonBackReference
    private Lecturer lecturer;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "subject_id", foreignKey = @ForeignKey(name = "SUBJECT_ID_FK"))
    @JsonBackReference
    private Subject subject;

    public Limitations() {
    }
}
```

```
public Limitations(Integer type, String name, String acceptable, String
unacceptable, Groups groups, Lecturer lecturer, Subject subject) {
    this.type = type;
    this.name = name;
    this.acceptable = acceptable;
    this.unacceptable = unacceptable;
    this.groups = groups;
    this.lecturer = lecturer;
    this.subject = subject;
}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public Integer getType() {
    return type;
}

public void setType(Integer type) {
    this.type = type;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getAcceptable() {
    return acceptable;
}

public void setAcceptable(String acceptable) {
    this.acceptable = acceptable;
}

public String getUnacceptable() {
    return unacceptable;
}

public void setUnacceptable(String unacceptable) {
    this.unacceptable = unacceptable;
}

public Groups getGroups() {
    return groups;
}

public void setGroups(Groups groups) {
    this.groups = groups;
}

public Lecturer getLecturer() {
    return lecturer;
}

public void setLecturer(Lecturer lecturer) {
```

```

    this.lecturer = lecturer;
}

public Subject getSubject() {
    return subject;
}

public void setSubject(Subject subject) {
    this.subject = subject;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Limitations)) return false;
    Limitations that = (Limitations) o;
    return getType().equals(that.getType()) &&
        getName().equals(that.getName()) &&
        Objects.equals(getAcceptable(), that.getAcceptable()) &&
        Objects.equals(getUnacceptable(), that.getUnacceptable()) &&
        Objects.equals(getGroups(), that.getGroups()) &&
        Objects.equals(getLecturer(), that.getLecturer()) &&
        Objects.equals(getSubject(), that.getSubject());
}

@Override
public int hashCode() {
    return Objects.hash(getType(), getName(), getAcceptable(), getUnacceptable(),
        getGroups(), getLecturer(), getSubject());
}

@Override
public String toString() {
    return "Limitations{" +
        "id=" + id +
        ", type=" + type +
        ", name='" + name + '\'' +
        ", acceptable='" + acceptable + '\'' +
        ", unacceptable='" + unacceptable + '\'' +
        '}';
}
}

```