Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра мережних технологій

Автоматизована система складання розкладу університету з адаптивним алгоритмом вирішення конфліктів

Текстова частина до кваліфікаційної роботи
за спеціальністю «Інженерія програмного забезпечення»

Керівник кваліфікаційної роботи
д.т.н., доц. Глибовець А. М.
_____ (*підпис*)
«____» _____ 2026р.
Виконав студент ІПЗ-4
Омельчук Руслан Ігорович
«____» _____ 2026р.

Київ 2026

ЗМІСТ

Анотація 2

Вступ 3

1 Теоретична частина 6

1.1 Аналіз задачі складання розкладу та огляд існуючих підходів 6

1.2 Дослідження специфіки предметної області 10

1.3 Обґрунтування вибору інструментарію та технологій 18

2 Практична частина 24

2.1 Збір та статистичний аналіз історичних даних навчального процесу . . . 24

2.2 Формалізація математичної моделі та обмежень 31

2.3 Проєктування архітектури системи 35

2.4 Реалізація алгоритмів та програмних модулів 38

2.5 Інтеграція з існуючою системою університету 43

3 Аналіз отриманих результатів 49

3.1 Оцінка якості згенерованих розкладів 49

3.2 Аналіз ефективності розробленого інструментарію 51

Висновки 53

Список літератури 55

АНОТАЦІЯ

Роботу присвячено розробці системи автоматизованого складання університетського розкладу для НаУКМА в умовах ліберальної освітньої моделі. Використано методи комбінаторної оптимізації та програмування в обмеженнях. Розроблено математичну модель і реалізовано алгоритм із трифазовою декомпозицією; результат інтегровано з університетською платформою Smart-UKMA у вигляді мікросервісу з редактором розкладу для адміністраторів та переглядачем для студентів і викладачів. Підтверджено практичну реалізовність підходу на напівсинтетичних даних.

Ключові слова: складання університетського розкладу, комбінаторна оптимізація, програмування в обмеженнях, ліберальна освіта, мікросервісна архітектура, автоматизація розкладу.

ВСТУП

У сучасному університеті складання навчального розкладу є рутинною, але надзвичайно трудомісткою задачею. Вручну це завдання потребує тижнів роботи відповідальних за розклад — і навіть після значних зусиль результат нерідко містить конфлікти або неоптимально розподіляє ресурси. Задача складання розкладу занять в університеті (University Course Timetabling Problem, UCTP) є окремим випадком комбінаторної оптимізації: необхідно розмістити фіксований набір занять у часових слотах із врахуванням обмежень на аудиторії, викладачів і студентів. З огляду на розмірність цього простору UCTP класифікується як NP-складна задача [1, 2], що унеможливлює знаходження оптимального рішення за розумний час при достатньо великому масштабі.

Особливу складність становить складання розкладу в закладах, що реалізують модель ліберальної освіти. Національний університет «Києво-Могилянська академія» (НаУКМА) є прикладом такого закладу: організація навчального процесу суттєво відрізняється від традиційної моделі з фіксованими академічними групами та єдиним навчальним планом. Це породжує специфічні вимоги до задачі складання розкладу, що принципово відрізняють її від стандартного UCTP і не дозволяють безпосередньо застосувати типові готові рішення.

НаУКМА використовує платформу Smart-UKMA на основі Java Spring Boot та Angular для автоматизації адміністративних процесів університету. Водночас автоматизований інструмент для складання розкладу в ній відсутній, і розклад традиційно формується вручну. Розробка такого інструменту та його інтеграція з наявною платформою є актуальним завданням із практичної точки зору.

Метою роботи є розробка автоматизованої системи складання університетського розкладу з алгоритмом на основі задоволення обмежень та її інтеграція з платформою Smart-UKMA. Для досягнення мети поставлено такі завдання:

1. Проаналізувати існуючі підходи до розв'язання задачі складання університетського розкладу.
2. Дослідити специфіку предметної області та сформулювати вимоги до системи.
3. Розробити або адаптувати існуючий алгоритм складання розкладу.
4. Реалізувати програмний засіб на основі обраного алгоритму та інтегрувати його з університетською платформою.
5. Проаналізувати отримані результати та сформулювати рекомендації щодо подальшого розвитку системи.

Об'єктом дослідження є процес складання університетського розкладу, а предметом — автоматизована система його складання з адаптивним алгоритмом вирішення конфліктів. Для її розв'язання використано: аналіз та систематизацію наукової літератури, математичне моделювання, статистичний аналіз реальних даних, методи програмної інженерії та обчислювальний експеримент.

У роботі запропоновано підхід до автоматизованого складання університетського розкладу в умовах ліберальної освіти на основі декомпозиції задачі та програмування в обмеженнях, що враховує структурну відмінність між обов'язковими та вибірковими дисциплінами. Розроблений прототип є основою для автоматизації складання розкладу в НаУКМА та скорочення тривалості цього процесу і зменшення людських помилок. Прототип інтегровано з платформою Smart-UKMA — адміністратори отримують інтерфейс для перегляду, редагуван-

ня та публікації розкладу, а студенти — доступ до персоналізованого перегляду опублікованого розкладу. Після доопрацювання бізнес-логіки та тестування в умовах реального навчального процесу систему може бути впроваджено в роботу університету.

Робота складається з трьох частин. Теоретична частина присвячена аналізу задачі складання університетського розкладу та огляду існуючих підходів до її розв’язання, дослідженню специфіки предметної області НаУКМА, а також обґрунтуванню вибору інструментів і технологій. Практична частина охоплює збір та статистичний аналіз реальних даних, формалізацію математичної моделі та обмежень, проектування архітектури системи, реалізацію алгоритму й програмних модулів та інтеграцію з університетською платформою. Аналітична частина містить аналіз отриманих результатів та рекомендації щодо подальшого розвитку системи.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Аналіз задачі складання розкладу та огляд існуючих підходів

UCTP належить до класу задач комбінаторної оптимізації, де необхідно розмістити фіксований набір занять у часових слотах з урахуванням різноманітних обмежень. Задача активно досліджується з 1960-х років і характеризується значним розмаїттям як постановок, так і методів розв'язання. Простір задачі формують чотири основні типи сутностей:

- Події - лекції, семінари, лабораторні роботи
- Ресурси - аудиторії, лабораторії, викладачі
- Часові інтервали - фіксовані блоки часу навчального тижня
- Навчальні плани - групи студентів, що мають відвідувати певні набори занять

З огляду на розмірність цього простору UCTP класифікується як NP-складна задача [1, 2]. Загальну класифікацію задач університетського розкладу наведено на Рис. 1.

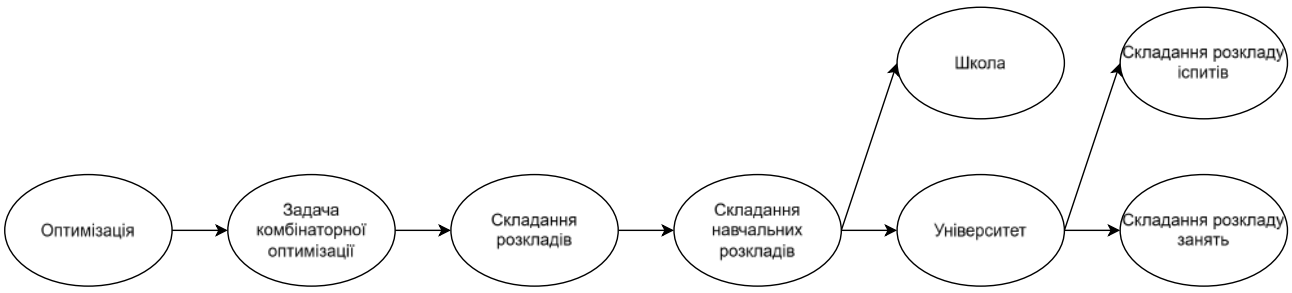

Рис. 1: Класифікація задач складання університетського розкладу

Обмеження в UCTP традиційно поділяються на дві категорії. Жорсткі обмеження визначають допустимість розкладу: їх порушення унеможливлює його використання. Прикладами є заборона проведень одночасно двох різних занять в одній аудиторії або у одного викладача; вимога відповідності місткості аудиторії

кількості записаних студентів. М’які обмеження визначають якість розкладу: їх порушення не робить розклад недопустимим, але погіршує результат. До типових м’яких обмежень належать: мінімізація, так званих, «вікон» у розкладі - тобто розміщення занять підряд; рівномірний розподіл занять протягом тижня; врахування побажань викладачів щодо зручного часу.

Дослідження цієї проблеми розпочалися ще у 1960-х роках [3]. Першу систематичну роботу зі складання розкладу опублікував Готліб у 1963 році. У 1965 році Чіма заклав теоретичне підґрунтя задачі на основі теорії графів, а у 1971 році де Верра запропонував евристичний алгоритм побудови шкільного розкладу. У 1990-х роках до галузі прийшли генетичні алгоритми та інші метаевристичні підходи. Починаючи з 2002 року регулярно проводяться Міжнародні змагання зі складання розкладу (International Timetabling Competition, ITC), що сформували стандартизовані тестові набори ITC-2002, ITC-2007 та ITC-2019 і сприяли порівняльному аналізу методів.

Задачі розкладу в університетах поділяються на два основних класи: складання розкладу занять (UCTP) та складання розкладу іспитів (University Examination Timetabling Problem). У свою чергу, UCTP поділяється на два підвиди залежно від джерела конфліктів: складання розкладу на основі навчального плану (Curriculum-Based Course Timetabling, CB-CTP), де конфлікти виникають із заздалегідь відомих навчальних планів, та складання розкладу після запису студентів (Post-Enrollment Course Timetabling, PE-CTP), де конфлікти визначаються реальними записами студентів на дисципліни [4].

Для розв’язання UCTP у літературі запропоновано широкий спектр методів, які можна класифікувати за п’ятьма основними підходами.

Точні методи — цілочисельне лінійне програмування (ILP) та програмування в обмеженнях (CP) — є єдиними підходами, що математично гарантують оптимальність результату. Задача відображається в алгебраїчній моделі, яка систематично досліджується вирішувачем. Практичним обмеженням є обчислювальна складність: через NP-складність UCTP зі збільшенням кількості курсів і студентів час виконання стає неприйнятно великим для реальних університетських екземплярів. Крім того, додавання нових специфічних обмежень вимагає реструктуризації базових рівнянь [4].

Конструктивні евристики — зокрема, розфарбування графів та послідовне розміщення — будують розклад крок за кроком із порожньої сітки на основі заздалегідь визначених правил. Їхньою перевагою є обчислювальна ефективність: базовий розклад генерується за секунди. Проте «жадібна» природа цих методів призводить до короткозорих рішень на ранніх кроках, що може унеможливити розміщення останніх занять або суттєво обмежити якість м’яких обмежень.

Методи локального пошуку — імітація відпалу (Simulated Annealing) та пошук із забороненими переходами (Tabu Search) — починають із повного, але недосконалого розкладу та ітеративно вносять невеликі зміни, оцінюючи їхній вплив на якість. Імітація відпалу зокрема здатна уникати локальних оптимумів, приймаючи погіршення на ранніх стадіях пошуку. Суттєвим обмеженням є залежність від якості початкового рішення та необхідність ретельного налаштування параметрів.

Популяційні метаевристики — насамперед генетичні алгоритми — підтримують множину розв’язків одночасно, обираючи та «схрещуючи» найкращі з них для отримання нових поколінь. Такий підхід забезпечує широкий охват

простору пошуку і більш надійне знаходження глобального оптимуму порівняно з методами локального пошуку. Водночас оцінювання великої кількості розв'язків є обчислювально затратним, а операції схрещування можуть порушувати жорсткі обмеження у нащадків [4].

Матевристики поєднують сильні сторони кількох підходів у єдиний гібридний конвеєр. Типова схема включає конструктивну евристику для швидкого формування початкового розкладу, метаевристику для глобальної оптимізації та точний метод для локального вирішення залишкових конфліктів. Переможці змагань ITC найчастіше використовували саме гібридні підходи, що поєднують кілька метаевристик або методи математичного програмування з евристичним пошуком [4]. Класифікацію описаних підходів наведено на Рис. 2.

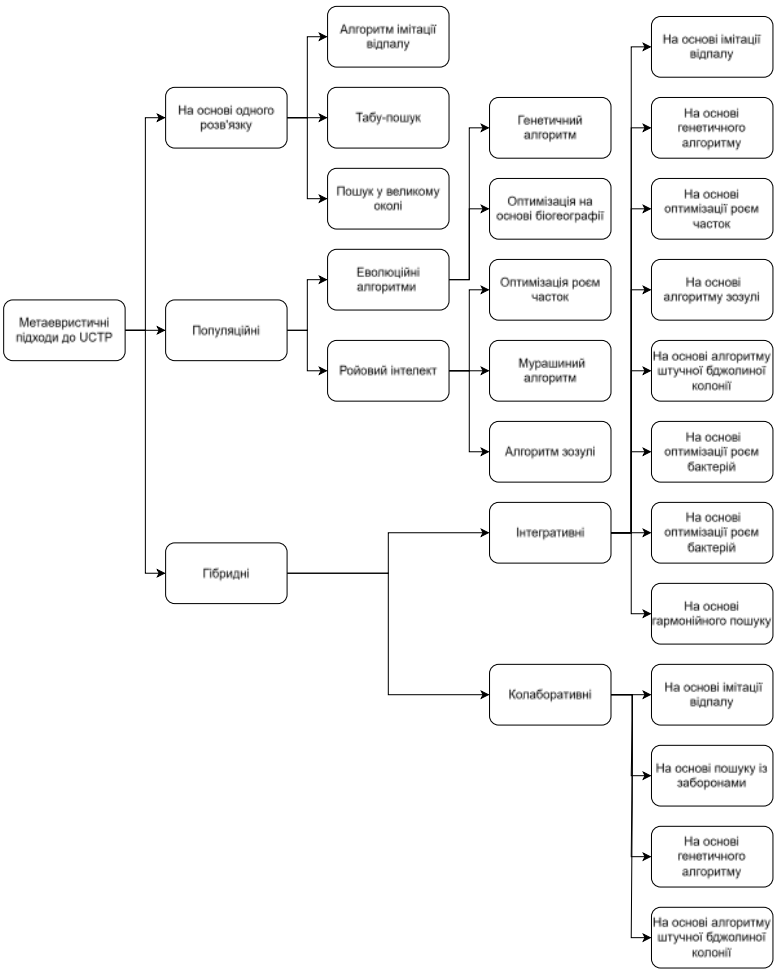

Рис. 2: Класифікація метаевристичних підходів до розв'язання UCTP

Система складання розкладу занять суттєво відрізняється між університетами: вимоги та правила встановлюються як на рівні конкретних закладів, так і на рівні освітньої системи країни, тому жодне єдине рішення, придатне для всіх університетів, не існує. З урахуванням цього, в роботі увага зосереджена на вирішенні задачі автоматизованого складання розкладу Національного університету «Києво-Могилянська академія» (НаУКМА).

Поряд з академічними дослідженнями існують готові програмні рішення для автоматизованого складання розкладу — зокрема UniTime, FET та Tablix. Ці системи підтримують широкий спектр обмежень і мають підтверджену ефективність у реальних закладах. Проте специфіка НаУКМА погано відповідає стандартній моделі даних таких систем, а нестандартна логіка обмежень значно простіше визначається у власній реалізації, ніж адаптується під готовий інструмент. З огляду на це, розробку нового модуля у вигляді мікросервісу в межах наявної системи університету визнано доцільнішим підходом. Детальна специфіка задачі для НаУКМА розглядається у наступному підрозділі.

1.2 Дослідження специфіки предметної області

У більшості університетів України організація навчального процесу підпорядкована жорсткій ієрархічній структурі. Університет поділяється на факультети, а факультети — на кафедри, що відповідають за окремі навчальні дисципліни. Студенти вступають до університету за певною спеціальністю, і вже з першого курсу формуються стійкі академічні групи — колективи чисельністю від п’ятнадцяти до тридцяти осіб. Усі студенти однієї групи навчаються разом за єдиним навчальним планом протягом усього терміну навчання. Склад груп є незмінним від вступу до випуску, якщо не брати до уваги відрахувань чи переведень.

Академічна група є основною одиницею розкладу: більшість практичних і семінарських занять проводяться саме на рівні групи. Лекції, як правило, об’єднують декілька академічних груп одного напряму підготовки та курсу — так звані потоки — і можуть охоплювати від кількох десятків до кількасот студентів, що вимагає великих лекційних аудиторій. Для практичних занять з малим коефіцієнтом студент-викладач, зокрема в комп’ютерних класах, на лабораторних роботах або на заняттях з іноземної мови, академічна група додатково ділиться на дві або три підгрупи чисельністю від восьми до п’ятнадцяти осіб.

Перелік дисциплін для кожного курсу визначається навчальним планом спеціальності, затвердженим кафедрою та факультетом. Вибіркові дисципліни, якщо й передбачені, становлять незначну частку навчального плану й нерідко обираються колективно: або вся група записується на один курс, або ж деканат пропонує обмежений список альтернатив. Таким чином, при складанні розкладу кількість змінних є відносно невеликою: необхідно розмістити фіксований набір дисциплін для кожної академічної групи в часових слотах, враховуючи доступність аудиторій та викладачів, студентське навантаження та відповідні санітарні норми. Кількість конфліктів між розкладами різних груп мінімальна, оскільки їхні навчальні плани практично не перетинаються.

Принципово інший підхід реалізований у Національному університеті «Києво-Могилянська академія». Після приєднання України до Болонського процесу у 2005 році та впровадження Європейської кредитно-трансферної системи (ECTS) вищі навчальні заклади отримали орієнтир на перехід до гнучких, орієнтованих на студента навчальних планів. НаУКМА реалізує цей принцип у формі ліберальної освіти: кожен студент щорічно формує індивідуальний навчальний

план (ІНП), самостійно обираючи навчальні дисципліни незалежно від факультету, кафедри або курсу їхнього викладання.

Фіксованих академічних груп у традиційному розумінні в університеті не існує. Натомість для кожного окремого курсу формується певна кількість окремих навчальних група, склад яких цілком визначається вибором студентів під час запису на дисципліну. Один і той самий курс можуть відвідувати студенти різних спеціальностей, різних курсів навчання. Кожна паралельна група курсу може мати різного викладача та відбуватись в інший часовий слот: студент обирає не лише дисципліну, а й конкретну групу всередині неї.

Усі дисципліни в НаУКМА поділяються на обов'язкові та вибіркові. Обов'язкові дисципліни студент зобов'язаний засвоїти у визначеному навчальним планом спеціальності семестрі, незалежно від особистих уподобань. Вибіркові дисципліни студент обирає самостійно в межах відповідного бюджету кредитів ECTS. Серед вибіркових виокремлюються «професійно-орієнтовані дисципліни» та вільні вибіркові курси. Перші безпосередньо пов'язані зі спеціальністю студента, другі можуть бути обрані з будь-якого доступного переліку без обмежень за напрямом або роком викладання.

Співвідношення обов'язкових і вибіркових дисциплін суттєво змінюється з курсом навчання. Студенти першого курсу здебільшого не мають вибору: їхній розклад майже повністю складається з обов'язкових дисциплін, визначених навчальним планом спеціальності. Починаючи з другого-третього курсу частка вибіркових дисциплін зростає та може перевищувати половину загального кредитного навантаження семестру. На старших курсах студент фактично самостійно формує значну частину свого навчального плану, добираючи курси з різних напрямів і навіть факультетів.

Така модель значно ускладнює процес складання розкладу занять порівняно з традиційним підходом. По-перше, склад навчальних груп наперед невідомий і визначається лише за результатами вибору студентів — тобто вже після того, як розклад має бути попередньо сформованим. По-друге, оскільки студенти різних спеціальностей і курсів можуть навчатися разом і мають різні набори обов'язкових дисциплін, будь-який збіг часових слотів між різними курсами потенційно створює конфлікт у чиємусь ІНП. По-третє, для обов'язкових дисциплін необхідно гарантувати повну відсутність конфліктів у розкладі для кожного студента, тоді як для вибіркових дисциплін слід прагнути до мінімізації конфліктів, але повне їх усунення здебільшого неможливе.

Формування ІНП у НаУКМА відбувається у два послідовних процеси. Перший — запис на дисципліни — відображено на Рис. 3. На підготовчому етапі деканат публікує перелік дисциплін, доступних для запису в поточному семестрі: їхні описи, викладачів, обмеження на наповнюваність груп. Студенти ознайомлюються з пропозицією та планують своє навчальне навантаження. Перший етап запису — основний: студенти подають заявки на дисципліни, які бажають відвідувати. Деякі категорії дисциплін можуть мати обмеження за кількістю вибраних курсів або встановлені квоти. Після першого етапу деканат проводить технічний контроль наповнюваності груп: якщо кількість охочих перевищує допустиму місткість, деканат може скоригувати розподіл; якщо ж група залишилася надто малою — вона може бути скасована. Потім відбуваються другий і третій етапи запису, під час яких студенти можуть доповнити або скоригувати свій ІНП, зокрема записатися на дисципліни, що звільнилися після першого еиапу. На завершення студенти відправляють підписані копії ІНП до деканату у встановленому порядку.

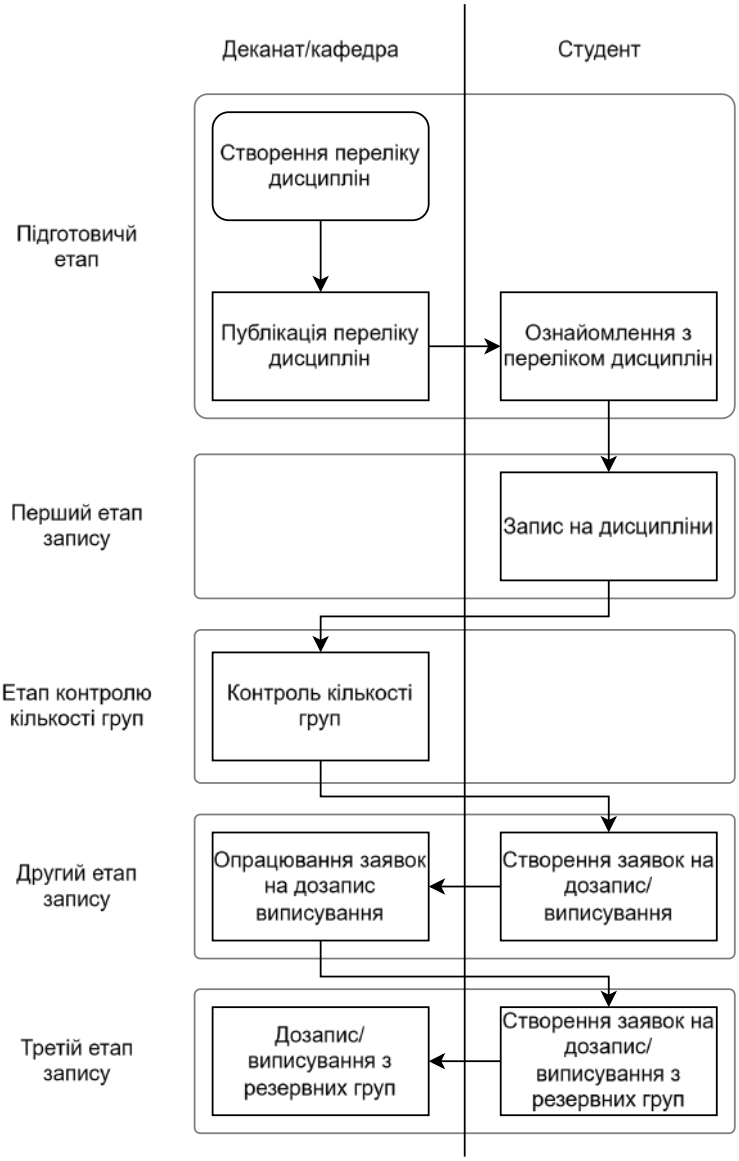

Рис. 3: Процес вибору дисциплін

Другий процес — вибір груп — розпочинається лише після того, як деканат склав і опублікував повний розклад занять із зазначенням переліку груп, викладачів і часових слотів для кожної дисципліни (Рис. 4).

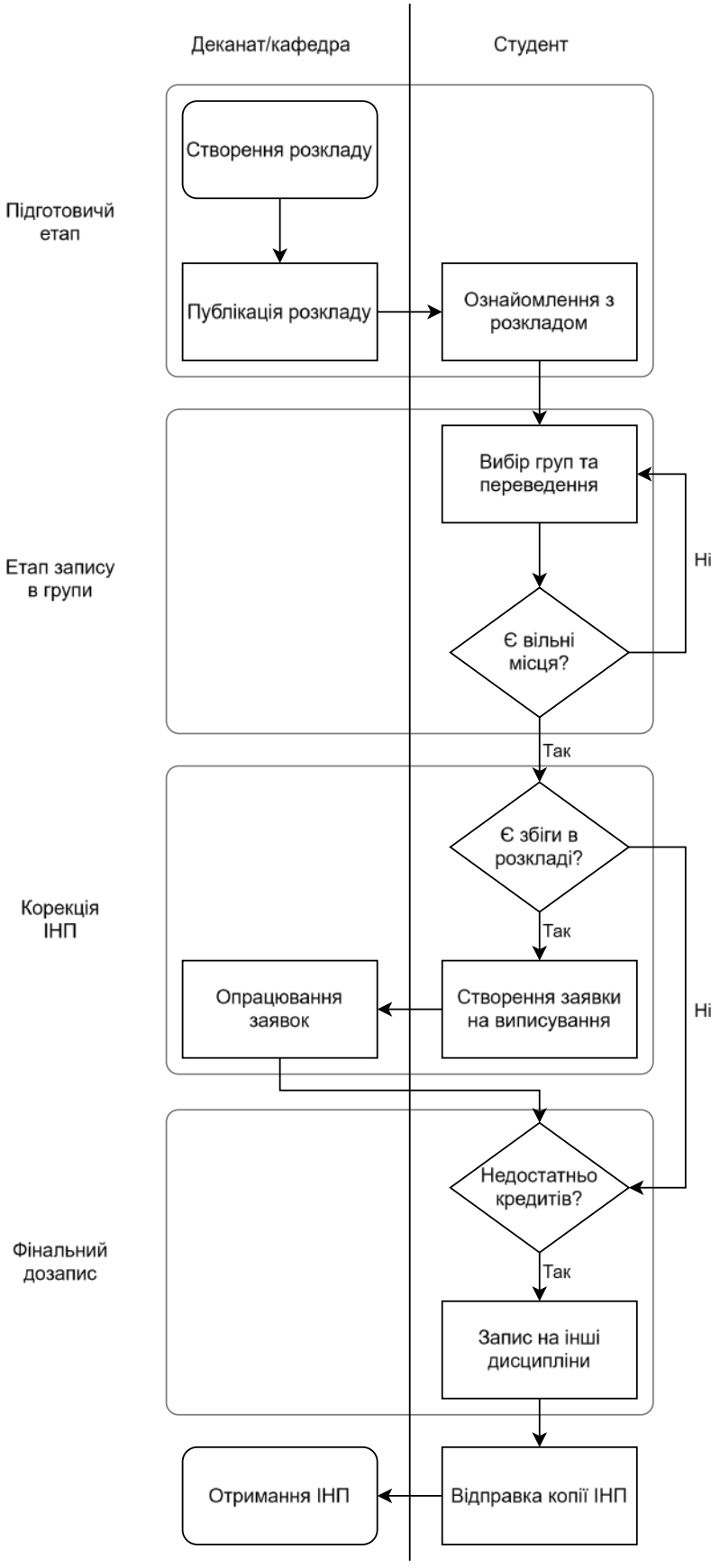

Рис. 4: Процес вибору груп

На підготовчому етапі вибору груп студенти ознайомлюються з опублікованим розкладом і визначають, до яких груп хочуть записатися, зважаючи на

зручність часу та, за наявності вибору, викладача. На першому етапі студенти подають заявки на конкретні групи; наповненість кожної з них є обмеженою, тому ранній запис може давати перевагу. Після завершення першого туру настає фаза коригування: якщо поєднання обраних дисциплін спричинює конфлікт у розкладі — тобто два заняття збігаються за часом, — студент подає заявку на виписування з відповідної дисципліни, а деканат розглядає й затверджує або відхиляє її. На завершальному етапі студенти, які через конфлікти втратили частину кредитів і не виконують мінімального кредитного порогу для переведення на наступний курс, можуть дозаписатися на дисципліни, що залишилися вільними. Після всіх коригувань студенти зберігають та відправляють фінальний варіант ІНП.

Розуміння цих двох процесів є ключовим для формулювання задачі, яку вирішує ця система. Система, представлена в цій роботі, автоматизує перший крок другого процесу — власне складання розкладу, без якого вибір груп неможливий. На вхід системи подається перелік так званих «архетипів занять»: яким викладачем викладається відповідний курс, яка максимальна наповненість групи, скільки разів на семестр має проводитися заняття тощо. На виході система формує повний розклад — призначення кожному заняттю аудиторії та часового слоту, — який деканат може передати студентам для подальшого вибору груп. При цьому система повинна гарантувати, що жоден студент не матиме конфліктів між обов’язковими дисциплінами, а кількість конфліктів для вибіркових дисциплін була мінімальною.

Нижче наведено фрагмент реального розкладу занять для однієї з навчальних груп університету (Табл. 1), що ілюструє практичну складність задачі складання розкладу.

День	Час	Дисципліна, викладач	Група	Тижні	Аудиторія
Понеділок	8:30-9:50	Веб-програмування, доц.О.В. Олецький	лекція	1-7	Дистанційно
	10:00-11:20	Веб-програмування, доц.О.В. Олецький	2	1-7,9-15	Дистанційно
		Патерни проектування, доц. В.В.Бублик	1	1-7,9-15	Дистанційно
	11:40-13:00	Патерни проектування, доц. В.В.Бублик	2	1-7,9-15	Дистанційно
		Веб-програмування, доц.О.В. Олецький	3	1-7,9-15	Дистанційно
	13:30-14:50	Патерни проектування, доц. В.В.Бублик	3	1-7,9-15	Дистанційно
		Веб-програмування, доц.О.В. Олецький	4	1-7,9-15	Дистанційно
	15:00-16:20	Веб-програмування, доц.О.В. Олецький	5	1-7,9-15	Дистанційно
		Патерни проектування, доц. В.В.Бублик	1	12	Дистанційно
		Патерни проектування, доц. В.В.Бублик	2	13	Дистанційно
		Патерни проектування, доц. В.В.Бублик	3	14	Дистанційно
	16:30-17:50				
	18:00-19:20				
Вівторок	8:30-9:50	Технології сучасних дата-центрів, ст.викл. Д. І. Черкасов	лекція	1-7,9-12	1-223 / Дистанційно
	10:00-11:20	Технології електронних видань, доц. А.О. Афонін	лекція	1-7,9-12	1-206 / Дистанційно
	11:40-13:00	Технології електронних видань, доц. А.О. Афонін	1	1-7,9-11	Дистанційно
	13:30-14:50	Теорія ймовірностей, проф.І.В. Федосова	лекція	1-7,9-13	1-223
	15:00-16:20	Теорія ймовірностей, проф.І.В. Федосова	1	1,3,5	1-223
		Теорія ймовірностей, проф.І.В. Федосова	2	2,4,6	1-223
	16:30-17:50	Глибинне навчання для задач комп'ютерного зору, ст.викл. Д. Кузьменко	лекція	1-7,9-12	1-225
	18:00-19:20	Глибинне навчання для задач комп'ютерного зору, ст.викл. Д. Кузьменко	1	1-7,9-11	1-206

Табл. 1: Приклад фрагменту розкладу

Аналіз наведеного прикладу дозволяє виділити кілька характерних особливостей. Розклад охоплює заняття різних типів: лекції для цілого потоку і практичні заняття, що проводяться окремо для підгруп. Для ряду дисциплін — наприклад, «Веб-програмування» та «Патерни проектування» — студентів поділено на до п'яти підгруп із різними часовими слотами, що збільшує загальну кількість подій у розкладі й вимагає уникнення перетинів між підгрупами одного курсу.

Заняття розподілені по тижнях нерівномірно: частина курсів проводиться неперервними діапазонами (наприклад, 1–7 або 9–15), інші — у вибрані тижні (наприклад, 1, 3, 5). Часовий слот, таким чином, не є простою одиницею «пара–день», а характеризується ще й тижневою прив'язкою. Додатковий вимір складності створює змішаний формат навчання: частина занять проводиться дистанційно, тому при призначенні ресурсів необхідно розрізняти фізичні аудиторії та онлайн-формат.

Між деякими заняттями в межах одного дня виникають «вікна» — незайняті слоти між двома парами. Їхня мінімізація є типовим м’яким обмеженням: воно не порушує допустимості розкладу, але безпосередньо впливає на зручність розкладу для студентів.

З огляду на розглянуту специфіку НаУКМА, задача складання розкладу є передусім PE-CTP: розклад публікується до того, як студенти обирають конкретні групи, тому точний склад груп на момент планування невідомий. Проте для обов’язкових дисциплін конфлікти між студентськими потоками можна визначити наперед із навчальних планів спеціальностей — що наближає цю підзадачу до CB-CTP. Така структурна асиметрія дозволяє декомпозувати загальну задачу на дві: спочатку розмістити обов’язкові заняття з гарантованою відсутністю конфліктів між потоками, а потім мінімізувати конфлікти для вибіркових дисциплін. Саме ця декомпозиція лежить в основі запропонованого підходу і суттєво спрощує розв’язання задачі.

1.3 Обґрунтування вибору інструментарію та технологій

Розробка системи автоматичного складання розкладу базується на використанні кількох груп технологій, кожна з яких виконує свою окрему роль у загальному життєвому циклі рішення. Важливо, що ці компоненти не завжди входять до фінальної продакшн-системи, але є необхідними для її створення, інтеграції та дослідження.

На основі аналізу літератури встановлено, що метаевристичні алгоритми є найпоширенішим підходом до розв’язання UCTP завдяки гнучкості та здатності знаходити якісні рішення у великих просторах пошуку. Водночас їхньою суттєвою вадою є необхідність ретельного налаштування параметрів та відсутність

гарантії глобального оптимуму [4]. Для практичного застосування у реальному університеті важливою перевагою є можливість декларативного опису задачі — формулювання обмежень незалежно від механізму пошуку рішення. Такий підхід реалізується фреймворками на основі виконання обмежень (constraint satisfaction), де жорсткі та м'які обмеження описуються явно, а оптимізаційний рушій застосовує метаевристику автономно.

При виборі інструментарію розглядалися три варіанти. Самостійна реалізація оптимізаційного рушія з нуля дає повний контроль, проте розробка ефективних евристик пошуку, механізмів відкату та управління доменами є окремою масштабною інженерною задачею, що суттєво виходить за межі цієї роботи. Тому порівняння зосереджено на двох готових фреймворках: TimeFold (раніше OptaPlanner) та Google OR-Tools.

TimeFold є спеціалізованим фреймворком для задач планування та розкладу. Його підхід до визначення обмежень орієнтований на предметну область: планувальні сутності — заняття, аудиторії, часові слоти — описуються як анотовані Java-класи, а обмеження декларуються через Constraint Streams — функціональний DSL, що нагадує SQL-з'єднання над об'єктами домену (Рис. 5). Жорсткі та м'які обмеження оголошуються в єдиному стилі: кожне — метод, що повертає об'єкт `Constraint` із ваговим штрафом. Вирішувач (метаевристика: відпал, Late Acceptance тощо) повністю прихований за API — фреймворк самостійно керує інкрементальним перерахунком оцінки, стратегією пошуку та критерієм зупинки. Як JVM-нативний інструмент, TimeFold природно інтегрується з існуючою Spring Boot інфраструктурою університету.

```
Constraint roomConflict(ConstraintFactory cf) {
    return cf
        .forEachUniquePair(Lesson.class,
            Joiners.equal(Lesson::getTimeslot),
            Joiners.equal(Lesson::getRoom))
        .filter((l1, l2) -> l1.getRoom() != null && l2.getRoom() != null)
        .penalize(HardSoftScore.ONE_HARD)
        .asConstraint("Room conflict");
}
```

Рис. 5: Визначення обмеження у TimeFold: Constraint Streams API

OR-Tools, натомість, є бібліотекою загальної комбінаторної оптимізації. Програміст будує явну математичну модель: цілочисельні та булеві змінні рішення з визначеними доменами, арифметичні обмеження (рівності, нерівності, глобальні обмеження `all_different` і `element`) та лінійну цільову функцію (Рис. 6). CP-SAT застосовує гібридний пошук — поширення обмежень, навчання на конфліктах за схемою CDCL та метод гілок і меж, — що дозволяє не лише знаходити якісні рішення, а й доводити їх оптимальність або обчислювати нижні межі. Програміст контролює кодування задачі, підказки для пошуку та часові ліміти вирішувача.

```
flat_vars = []
for i in range(n):
    v = model.new_int_var(0, num_slots * num_rooms - 1, f'sr_{i}')
    model.add(v == slot_vars[i] * num_rooms + room_vars[i])
    flat_vars.append(v)
model.add_all_different(flat_vars)
```

Рис. 6: Визначення обмеження в OR-Tools: явне арифметичне кодування

Порівняння ключових характеристик обох інструментів наведено в Табл. 2.

Характеристика	**TimeFold**	**OR-Tools CP-SAT**
Призначення	Спеціалізований фреймворк планування	Бібліотека загальної комбінаторної оптимізації
Визначення обмежень	Декларативний DSL над об’єктами предметної області	Явна математична модель (змінні, вирази)
Тип вирішувача	Метаевристика (SA, Late Acceptance)	Точний пошук (propagation + CDCL + B&B)
Гарантія оптимальності	Відсутня	Може довести оптимум або обмеження
Контроль над пошуком	Мінімальний	Повний (підказки, стратегії, ліміти)
Мовна підтримка	JVM (Java, Kotlin)	Python, Java, C++

Табл. 2: Порівняння TimeFold та OR-Tools CP-SAT

Попри переваги TimeFold у читабельності коду, на етапі доказу концепції виявилось, що фреймворк не може впоратися з простором пошуку задачі (близько 4 тисяч студентів та 26 тисяч занять на семестр): вирішувач не знаходив допустимого рішення за прийнятний час. OR-Tools з CP-SAT вирішувачем продемонстрував прийнятні результати на тих самих даних, що й зумовило остаточний вибір на користь цієї бібліотеки попри більшу складність коду. Серед доступних мовних прив’язок OR-Tools — Java, C++ та Python — обрано Python, оскільки ця мова вже використовується для збору та аналізу даних у проєкті. Це дозволяє запускати весь дослідницький конвеєр — від сирих даних до готового розкладу — в єдиному середовищі, а також тестувати зміни обмежень безпосередньо у Jupyter Notebook без перекомпіляції.

Вибір стеку для серверної та клієнтської частини системи визначається не технічними перевагами, а необхідністю інтеграції з існуючою інфраструктурою

НаУКМА. Серверна частина університетської платформи побудована з використанням мікросервісної архітектури на базі Spring Boot. Це передбачає наявність окремих сервісів, що взаємодіють між собою через стандартизовані API. З огляду на це, реалізація розроблюваного модуля у вигляді окремого мікросервісу з використанням тієї ж технології дозволяє забезпечити сумісність із наявною архітектурою, спрощену інтеграцію через REST API, повторне використання існуючих механізмів автентифікації, логування та конфігурації, а також зменшення витрат на підтримку та розгортання.

Клієнтська частина інформаційної системи університету реалізована з використанням фреймворку Angular. З метою збереження єдиного стилю користувацького інтерфейсу та забезпечення безшовної інтеграції з існуючими компонентами було прийнято рішення використовувати той самий фреймворк для розробки інтерфейсу взаємодії з модулем складання розкладу: це дозволяє інтегрувати новий функціонал без зміни існуючої фронтенд-архітектури, повторно використовувати наявні UI-компоненти та сервіси й забезпечити узгоджений користувацький досвід.

Таким чином, обраний стек технологій (Java Spring Boot для серверної частини та Angular для клієнтської) обумовлений не лише технічними характеристиками цих інструментів, але й необхідністю ефективної інтеграції з існуючою мікросервісною архітектурою університету. Це рішення дозволяє мінімізувати витрати на впровадження та забезпечити масштабованість і підтримуваність розроблюваної системи.

Оскільки компоненти системи реалізовані різними мовами — алгоритм на Python, серверний сервіс на Java, а клієнтська частина на TypeScript, — необхідний спільний формат передачі даних. Очевидною альтернативою є OpenAPI-

специфікація з генерацією клієнтського коду, проте якість генераторів суттєво відрізняється між мовами: підтримка для Python і TypeScript значно поступається Java-екосистемі. Ручне підтримання узгоджених схем у трьох мовах одночасно призвело б до дублювання коду та неминучих розбіжностей, що особливо критично на ранніх ітераціях, коли структура даних активно змінюється. Protocol Buffers (Protobuf) вирішує цю проблему інакше: схема визначається єдиний раз у `.proto`-файлі, а типізований код для Java, Python і TypeScript генерується автоматично з одного джерела. Підтримка Protobuf є першокласною для всіх трьох мов — генерований код є ідіоматичним і оновлюється за однією командою при зміні схеми. Це гарантує консистентність типів між компонентами й суттєво прискорює ітерації при зміні структури даних.

Окрему роль відіграють інструменти, які використовуються не в продакшн-системі, а на етапі дослідження та створення алгоритму.

Python використовується для збору даних зі системи САЗ, прототипування алгоритмів та аналізу результатів, тоді як Jupyter Notebook забезпечує зручне середовище для експериментів, візуалізації та перевірки коректності розкладів. Оскільки Python вже обрано для реалізації солвера, весь дослідницький конвеєр — від збору сирих даних до аналізу згенерованого розкладу — працює в єдиному середовищі без перемикання між мовами.

Таким чином, обраний технологічний стек формує чотири взаємодоповнювальні рівні: алгоритмічний (OR-Tools з Python), інтеграційний (Spring Boot і Angular), комунікаційний (Protobuf) та дослідницький (Python і Jupyter Notebook). Такий розподіл відповідальностей дозволяє вдосконалювати алгоритм незалежно від продакшн-системи, не жертвуючи узгодженістю інтеграції.

2 ПРАКТИЧНА ЧАСТИНА

2.1 Збір та статистичний аналіз історичних даних навчального процесу

Перш ніж проектувати алгоритм, необхідно дослідити реальні дані навчального процесу НаУКМА. Такий аналіз має на меті декілька цілей: визначити практичний масштаб задачі та параметри системи, виявити структурні характеристики, що впливають на вибір алгоритмічного підходу, а також підтвердити принципову розв'язність задачі.

Система автоматизованого запису (САЗ) НаУКМА є PHP-застосунком без публічного програмного інтерфейсу. Для отримання реальних даних розроблено модуль `smartsched-saz-scraper`, що автоматизує збір інформації безпосередньо з веб-інтерфейсу системи: метадані курсів (назва, факультет, кафедра, кількість груп, максимальна наповнюваність) та записи студентів (спеціальність, ознака обов'язковості, статус запису). Зібрані дані серіалізуються у формат Protobuf JSON і передаються далі по конвеєру до планувальника.

Аналізований набір охоплює осінній семестр 2025 року: після фільтрації до активних записів отримано 28 012 записів 4 366 студентів на 574 курси. З них 382 курси є обов'язковими хоча б для частини студентів (20 547 записів) та 270 — вибірковими (7 465 записів); частина курсів належить до обох категорій залежно від спеціальності студента. Середнє навчальне навантаження становить 6,42 курсу на студента. Для аналізу та формулювання ключових параметрів планувальника використовувалися Python і Jupyter Notebook.

Розподіл середніх розмірів груп є двомодальним з піками приблизно на 14 та 26 студентах (Рис. 7); лише незначна частка курсів має групи понад 80

студентів. Ці межі визначають реалістичний діапазон місткості аудиторій у задачі призначення приміщень.

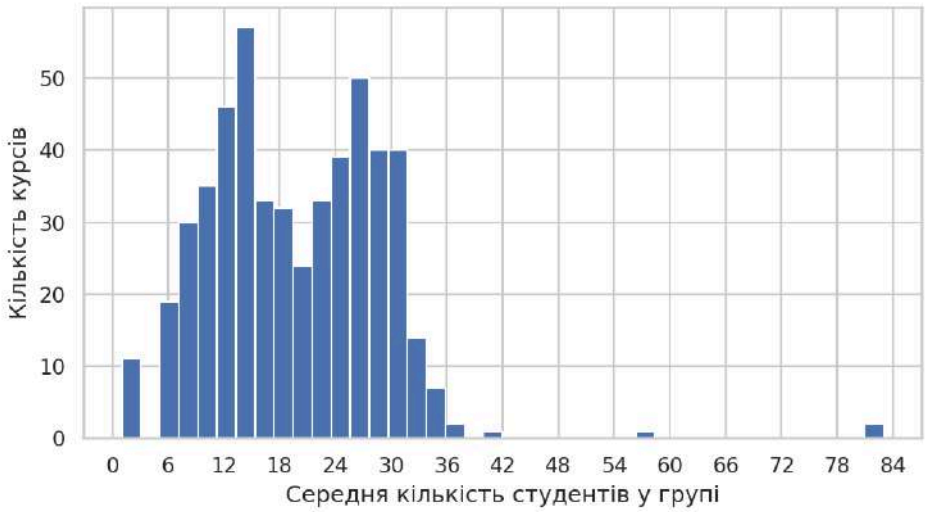

Рис. 7: Розподіл розміру груп

Розподіл кількості груп на курс суттєво правоскошений (Рис. 8): переважна більшість дисциплін має лише одну групу, значно менша частка — дві, а курси з трьома та більше групами є рідкісними. Це спостереження визначає гнучкість планувальника: курс з однією групою має фіксовану позицію в розкладі — всі студенти цього курсу отримують один і той самий часовий слот, і планувальник не може розподілити попит. Курси з кількома групами, навпаки, надають простір для маневру: різні групи можна розмістити в різних слотах, що знижує ймовірність конфліктів для студентів, які поєднують цей курс з іншими дисциплінами.

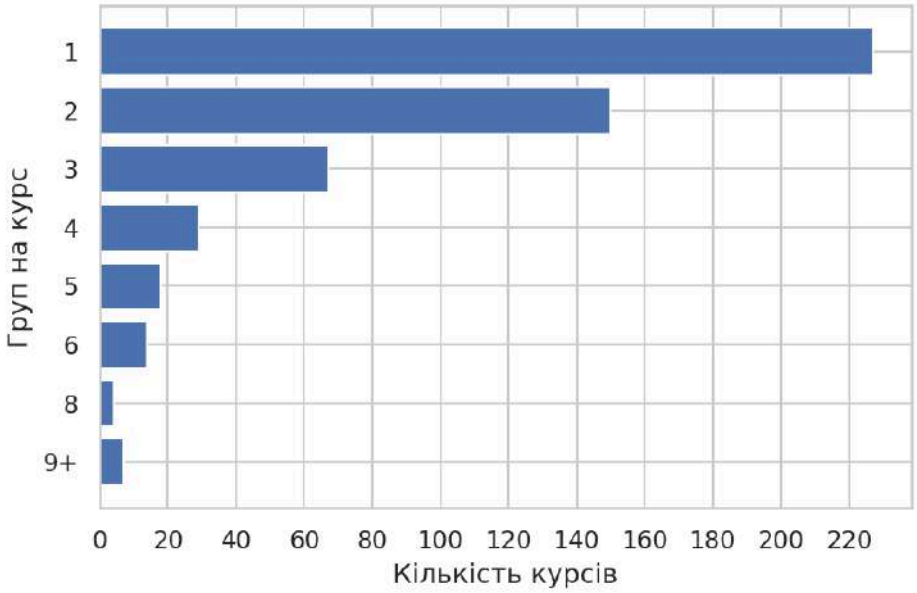

Рис. 8: Розподіл кількості груп на дисципліну

Поряд зі структурою груп, на складність задачі суттєво впливає міжфакультетська переплетеність вибіркових дисциплін. Матриця на Рис. 9 відображає частку студентів кожного факультету, що обирають вибіркові дисципліни певного факультету. Студенти факультету гуманітарних наук вибирають 77% вибіркових курсів у межах власного підрозділу, тоді як студенти факультету правничих наук — лише 41%. Дисципліни гуманітарних наук, соціальних наук та психології користуються стабільно високим попитом серед студентів усіх спеціальностей. Ця картина свідчить, що розклад вибіркових дисциплін є глибоко переплетеним між факультетами: декомпозиція задачі за факультетами не усуне конфліктів і є неприйнятною стратегією спрощення. Хоча поточні дані могли б підказати певні факультетські евристики, такі оптимізації були б крихкими: пропорції запису змінюються щороку зі зміною навчальних планів та студентських уподобань.

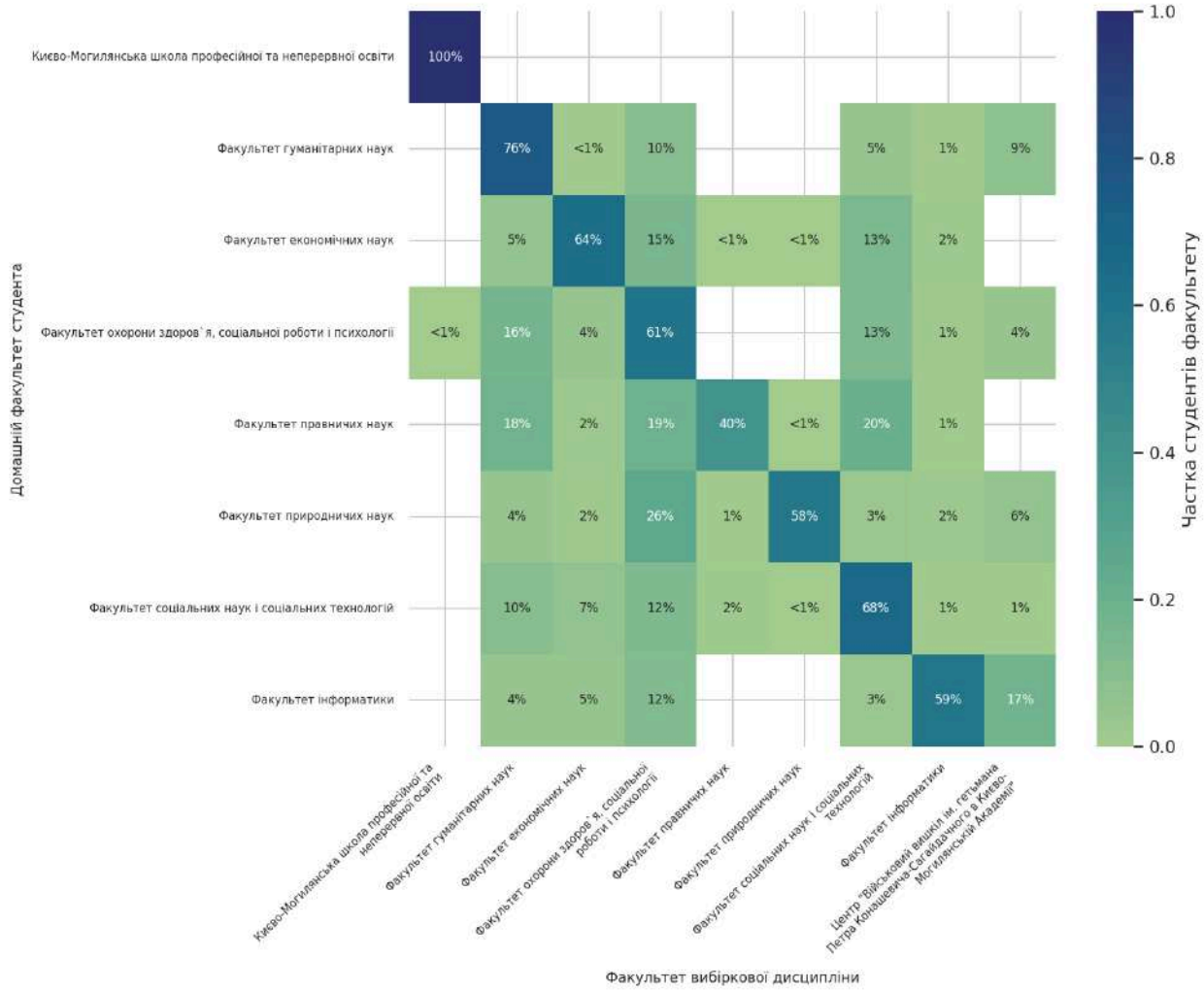

Рис. 9: Матриця факультетів вибіркових дисциплін студентів

На відміну від вибіркових, обов'язкові курси демонструють значно більш впорядковану картину. Попри те що 333 обов'язкові курси потенційно конкурують між собою, реальна структура конфліктів значно менш складна. Лише 19 з них мають у своїх записах студентів більш ніж однієї спеціальності — передусім загальноуніверситетські дисципліни: іноземна мова (англійська), українська мова, вища математика, статистика, мікроекономіка (Рис. 10). Решта є цілком специфічними для своєї спеціальності й не перетинаються з іншими програмами.

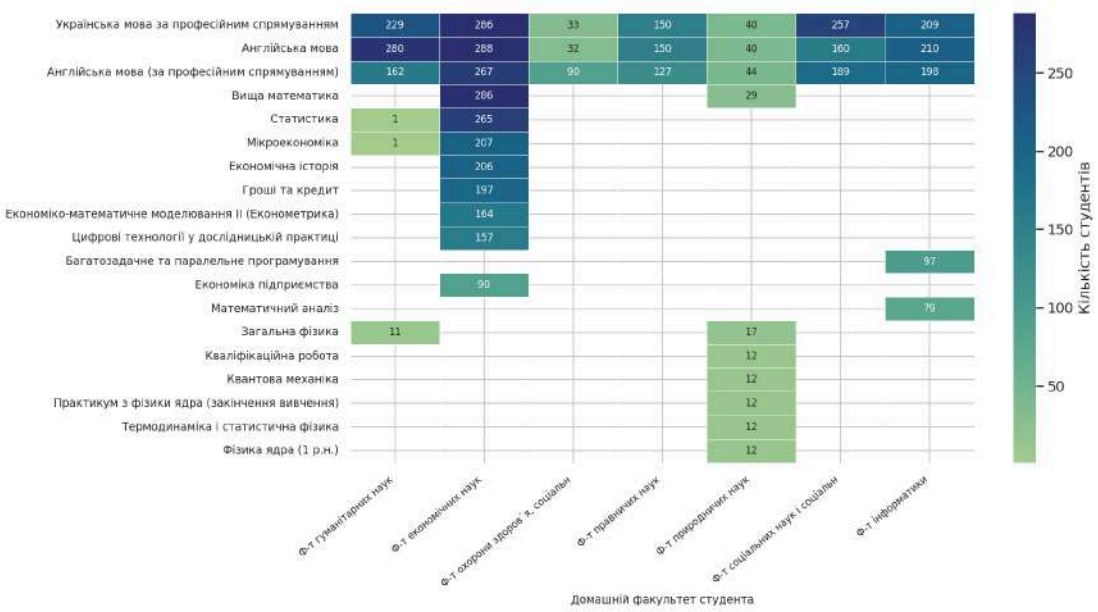

Рис. 10: Обов’язкові курси зі студентами різних спеціальностей

Для кількісної оцінки складності задачі та обґрунтування алгоритмічної стратегії використовується модель графа конфліктів. Граф конфліктів $G = (V, E)$ будується так: кожен курс є вузлом V, а ребро між двома курсами проводиться тоді і тільки тоді, коли хоча б один студент записаний на обидва курси одночасно. Задача складання розкладу зводиться до задачі розфарбування цього графа: призначення часового слоту кожному курсу еквівалентне присвоєнню кольору кожному вузлу так, щоб жодні два суміжні вузли — тобто курси, що конкурують між собою — не мали однакового кольору. Мінімальна кількість кольорів, за якої таке розфарбування можливе, є хроматичним числом $\chi(G)$ і відповідає мінімальній кількості слотів для безконфліктного розкладу. Оскільки обчислення $\chi(G)$ є NP-важкою задачею, для практичного аналізу використовуються три похідних метрики:

- Щільність графа $\rho = |E| \,/ \binom{|V|}{2}$ — частка реалізованих ребер від усіх можливих; відображає загальний рівень конкурентності між курсами.

- Ступінь вузла $\deg(v)$ — кількість курсів, що конкурують із курсом v; курси з великим ступенем є найбільш обмеженими у виборі слоту.
- Клікове-число $\omega(G)$ — розмір найбільшої кліки, тобто найбільшої підмножини курсів, кожна пара яких конфліктує. Оскільки всі курси кліки потребують попарно різних слотів, виконується $\omega(G) \leq \chi(G)$: клікове-число є нижньою межею на необхідну кількість слотів.

Наслідком цього є принципова різниця у щільності графів конфліктів (Рис. 11). У графі обов'язкових курсів (333 вузли, 1 022 ребра) щільність становить 1,85%, а медіанний ступінь вузла дорівнює 5. У графі вибіркових курсів (263 вузли, 2 758 ребер) щільність сягає 8,01%, медіанний ступінь — 20: граф є у 4,3 рази щільнішим.

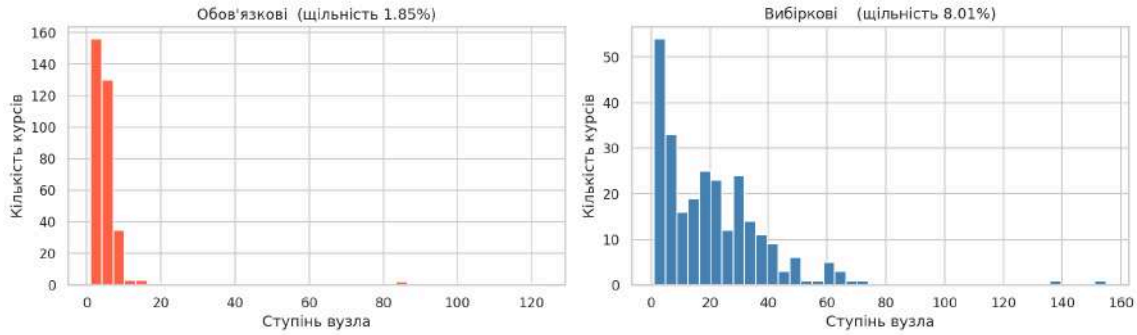

Рис. 11: Розподіл ступенів вузлів у графах конфліктів: обов'язкові та вибіркові

Отримані характеристики графів конфліктів дозволяють оцінити принципову реалізовність розкладу. Тижневий розклад НаУКМА налічує від 30 до 48 слотів (5–6 днів по 6–8 пар): субота використовується не завжди, а вечірні пари формально доступні, проте небажані і зазвичай уникаються при плануванні. Необхідна умова існування безконфліктного розкладу — це $\omega(G) \leq$ кількості доступних слотів: якщо найбільша кліка перевищує цей ліміт, жодне розфарбування не існує і задача принципово нерозв'язна. Ця умова є необхідною, але не доста-

тньою — $\chi(G)$ може бути більшим за $\omega(G)$, — проте її виконання чи порушення вже визначає принципову можливість розкладу.

Для обов'язкових курсів $\omega(G_{\text{обов}}) = 8 \leq 30$: необхідна умова виконана, а отже, жодного кліко-рівневого бар'єру для безконфліктного розкладу немає. Крім того, максимальне обов'язкове навантаження одного студента дорівнює 8 курсам, а 64,8% обов'язкових курсів мають ≥ 2 групи — тобто планувальник має простір для розподілу потоків по різних слотах. Сукупність цих факторів робить безконфліктний розклад обов'язкових дисциплін теоретично реалізовним.

Для вибіркових курсів $\omega\left(G_{\text{вибірк}}\right) = 14 \leq 30$: необхідна умова також виконана, проте щільність 8,01% свідчить про зовсім іншу природу задачі. У щільному графі $\chi(G)$ практично завжди суттєво перевищує $\omega(G)$, тому повне усунення конфліктів між вибірковими курсами є практично недосяжним незалежно від кількості доступних слотів у розумному діапазоні.

Серед обов'язкових курсів виокремлюється невелика підмножина «вузьких місць» — курсів, обов'язкових для ≥ 10 студентських потоків (Рис. 12). До них належать передусім загальноуніверситетські дисципліни: англійська мова (ступінь конфлікту 123) та українська мова (84) — найвищі значення у всьому наборі, що утворюють щільно зв'язане ядро і домінують у графі конфліктів.

Рис. 12: Розподіл кількості потоків на обов’язковий курс

Таким чином, зібрані дані розкривають асиметричну природу задачі планування. Обов’язкові курси мають розріджений і структурований граф конфліктів ($\omega = 8$, щільність 1,85%): безконфліктний розклад для них теоретично реалізовний і є природним жорстким обмеженням алгоритму. Вибіркові курси утворюють густо переплетений міжфакультетський граф (щільність 8,01%): повне усунення конфліктів недосяжне, і їх мінімізація виступає м’якою метою оптимізації. Нарешті, невелика підмножина обов’язкових «вузьких місць» з великою кількістю потоків домінує у графі конфліктів і вимагає пріоритетного розміщення. Ці три спостереження безпосередньо мотивують трифазову декомпозицію алгоритму, описану у підрозділі 2.4.

2.2 Формалізація математичної моделі та обмежень

Перш ніж описувати реалізацію, варто визначити задачу формально. Математична модель описує, що потрібно знайти і які умови накладаються на рішення; конкретна реалізація обмежень у CP-SAT вирішувачі описана у підрозділі 2.4.

Нехай E — множина архетипів занять (кожен архетип є одиницею планування — заняттям певного типу конкретного курсу з призначеним викладачем, яке проводиться у фіксованому слоті протягом визначеної кількості тижнів семе-

стру), R — множина аудиторій, T — впорядкована множина часових слотів, I — множина викладачів, Σ — множина студентських потоків (пара «спеціальність — рік навчання»). Кожен слот $t \in T$ однозначно ідентифікується днем тижня $\mathrm{day}(t)$ та порядковим номером у межах дня $\mathrm{tod}(t)$. Для кожного архетипу $e \in E$ задані: викладач $\mathrm{teacher}(e) \in I$, розмір групи $\mathrm{size}(e)$, курс $\mathrm{course}(e)$, ознака типу $\mathrm{isLec}(e) \in \{0, 1\}$ та множина потоків $\mathrm{streams}(e) \subseteq \Sigma$, для яких курс є обов'язковим. Місткість аудиторії $r \in R$ позначимо $\mathrm{cap}(r)$.

На основі введених множин для кожного архетипу $e \in E$ вводяться дві цілочисельні змінні:

$$s_e \in \{0, 1, ..., |T| - 1\}, \quad r_e \in \{0, 1, ..., |R| - 1\}$$

де s_e — індекс призначеного часового слоту, r_e — індекс призначеної аудиторії.

Допустимий розклад — такий набір значень (s_e, r_e), що задовольняє таким жорстким умовам.

(C1) Відсутність конфлікту аудиторій: жодні два архетипи не займають одну аудиторію в один слот:

$$(s_e, r_e) \neq (s_{e'}, r_{e'}) \quad \forall e \neq e' \in E$$

(C2) Місткість аудиторії: місткість призначеної аудиторії не менша за розмір групи:

$$\mathrm{cap}(r_e) \geq \mathrm{size}(e) \quad \forall e \in E$$

(C3) Відсутність конфлікту викладача: один викладач не може вести два заняття одночасно:

$$s_e \neq s_{e'} \quad \forall e \neq e' \in E : \mathrm{teacher}(e) = \mathrm{teacher}(e')$$

(C4) Лекція перед практикою: для кожного курсу всі лекції у розкладі передують практичним заняттям у порядку слотів:

$$s_l < s_p \quad \forall l, p \in E : \text{course}(l) = \text{course}(p), \quad \text{isLec}(l) = 1, \quad \text{isLec}(p) = 0$$

(C5) Відсутність конфліктів у потоці: для кожного потоку $\sigma \in \Sigma$ лекції обов'язкових курсів розміщуються у різних слотах і не перетинаються з практичними заняттями інших обов'язкових курсів того самого потоку. Позначимо $E_\sigma^{\text{lec}} = \{e \in E : \sigma \in \text{streams}(e), \text{isLec}(e) = 1\}$. Тоді:

$$s_e \neq s_{e'} \quad \forall e \neq e' \in E_\sigma^{\text{lec}}$$

$$s_l \neq s_p \quad \forall l \in E_\sigma^{\text{lec}}, \quad \forall p \in E : \sigma \in \text{streams}(p), \quad \text{isLec}(p) = 0, \quad \text{course}(p) \neq \text{course}(l)$$

З-поміж допустимих розкладів задача вимагає знайти оптимальний. Розв'язання відбувається у три фази; фази 1 і 2 використовують однакову математичну модель — мінімізацію зваженої суми шести якісних штрафних доданків:

$$\min \sum_{k=1}^{6} \omega_k \cdot P_k$$

де штрафні доданки наведено в Табл. 3. Різниця між фазами — не в моделі, а в домені: фаза 1 оперує над підмножиною $E^{(1)} \subset E$ найбільш обмежених архетипів, а фаза 2 — над повним E з фіксованими результатами фази 1.

Позначення	**Зміст**
P_1	Внутрішньоденний проміжок: $\sum_c (\max_e \text{tod}(s_e) - \min_e \text{tod}(s_e))$ по курсах c
P_2	Охоплення тижня: $\sum_c (\max_e \text{day}(s_e) - \min_e \text{day}(s_e))$ по курсах c
P_3	Відстань між лекцією та практикою: $\sum_{(l,p)} (s_p - s_l)$
P_4	Кількість пар курсів одного потоку, призначених в один день
P_5	Штраф за пізній часовий слот із різким зростанням у вечірній час
P_6	Дисперсія денного навантаження: $\sum_d n_d^2$, де n_d — кількість занять у день d

Табл. 3: Штрафні доданки цільової функції

У третій фазі модель розширюється для врахування вибору студентами конкретної секції дисципліни. Нехай $\mathcal{G}$ — множина груп студентів з однаковим набором записаних дисциплін. Для кожної групи $g \in \mathcal{G}$ та дисципліни c, на яку вона записана, нехай $A_c \subseteq E$ — множина архетипів (паралельних груп) цієї дисципліни. Вводиться булева змінна вибору секції:

$$p_{g,c,j} \in \{0,1\} \quad \forall g \in \mathcal{G}, c, j \in A_c$$

де $p_{g,c,j} = 1$ означає, що група g обрала секцію j дисципліни c, з обмеженням рівно одного вибору:

$$\sum_{j \in A_c} p_{g,c,j} = 1 \quad \forall g \in \mathcal{G}, c$$

Для кожної пари архетипів (a, b) з різних дисциплін, що можуть бути одночасно обрані групою g, визначається індикатор конфлікту $\text{conflict}_{g,a,b} \in \{0,1\}$, що дорівнює 1 тоді і тільки тоді, коли $p_{g,c(a),a} = 1$, $p_{g,c(b),b} = 1$ та $s_a = s_b$. Навантаження архетипу j визначається як сумарна кількість студентів груп, що обрали цю секцію:

$$\text{load}_j = \sum_{g \in \mathcal{G}} p_{g,c(j),j} \cdot |g|$$

де $|g|$ — кількість студентів у групі g.

До базової цільової функції ці змінні вибору секції додають три штрафні доданки рівня студента:

$$\min \sum_{k=1}^{6} \omega_k \cdot P_k + w_7 H + w_8 Q + w_9 O$$

де:

- $H = \sum_{g,a,b} \text{conflict}_{g,a,b}$ де обидві дисципліни $c(a)$ і $c(b)$ є обов'язковими для g
- $Q = \sum_{g,a,b} \text{conflict}_{g,a,b}$ де хоча б одна дисципліна є вибірковою

- $O = \sum_{j \in E} \max\bigl(0, \mathrm{load}_j - \mathrm{size}(j)\bigr)$ — сумарне перевищення наповнюваності груп

Ієрархія ваг $w_7 \gg w_8 \gg w_9 \gg \omega_k$ відображає пріоритети оптимізації: усунення конфліктів між обов'язковими дисциплінами є домінуючим критерієм, тоді як якісні показники розкладу покращуються лише після вирішення всіх критичних конфліктів.

Наведена модель охоплює основні обмеження задачі. Деякі аспекти — зокрема тижнева прив'язка занять та механізм передачі результатів між фазами — виходять за межі формального опису і розглядаються у підрозділі 2.4.

2.3 Проєктування архітектури системи

Розробка системи автоматизованого складання розкладу є комплексним ітеративним процесом, що значно ускладниться реалізацію єдиного монолітного застосунку з самого початку. На ранніх етапах необхідні синтетичні тестові дані, допоміжний код для налагодження та засоби аналізу проміжних результатів — усе це суперечить вимогам до програмного забезпечення розгорнутого в реальному середовищі. Алгоритм потребує численних ітерацій: зміна обмежень, перебудова евристик, коригування ваг — і кожна правка здатна зруйнувати попередньо досягнений результат. Тому системі необхідна архітектура, що забезпечує ізольованість компонентів, легкість заміни окремих частин та прозорість проміжних станів даних.

За основу архітектурного підходу взято принцип UNIX-ідеології, сформульований Дугласом МакІлроєм: «Пишіть програми, які роблять одну справу і роблять її добре. Пишіть програми, що працюють разом». Кожен модуль системи виконує єдину чітко визначену функцію та не знає нічого про внутрішній устрій

сусідніх модулів. Взаємодія між ними здійснюється через стандартні потоки введення-виведення: модуль зчитує вхідні дані зі stdin, обробляє їх і записує результат у stdout. Оператор конвеєра | командного рядка дозволяє довільно компонувати модулі без жодних змін у їхньому коді, а стандартні утиліти Linux — `cat`, `head`, `grep`, `jq` — природно вписуються у цей конвеєр для інспекції або фільтрації даних на будь-якому кроці без написання додаткового коду.

Для забезпечення узгодженості формату даних між модулями, реалізованими різними мовами програмування, як протокол обміну обрано Protocol Buffers (Protobuf). Альтернатива — звичайний JSON — потребувала б ручного підтримання синхронізації схеми у кожній мові, що зі збільшенням кількості полів неминуче стає джерелом помилок. Protobuf натомість визначає структуру даних у єдиному `.proto`-файлі, з якого автоматично генерується код для всіх цільових мов. Скрипт `gen_protos.sh` запускає компілятор `protoc` через Docker, що забезпечує відтворюваність генерації незалежно від локального середовища розробника, і розміщує згенеровані бібліотеки для Java, Python та TypeScript у каталозі `libs/generated/`. Центральне повідомлення `ScheduleTask` об'єднує як формулювання задачі (`ScheduleProblem`), так і її розв'язок (`ScheduleSolution`); модулі, що ще не сформували рішення, передають порожній об'єкт рішення — таким чином структура повідомлення залишається незмінною на всіх етапах конвеєра.

Розроблена система є складовою цифрової інфраструктури НаУКМА — платформи Smart-UKMA, побудованої за мікросервісною архітектурою. Кожен сервіс реалізовано засобами Java та Spring Boot і використовує власну реляційну базу даних PostgreSQL. Ключовими для цілей роботи є три сервіси:

- `user-info` - зберігає відомості про дисципліни, студентів та викладачів

- `account-server` - забезпечує централізовану автентифікацію та управління ролями (RBAC)
- `ui-server` - слугує Backend-For-Frontend — він обслуговує Angular-застосунок і слугує проксі для запитів до решти сервісів.

Сервіси взаємодіють синхронно через REST за допомогою Feign-клієнтів та асинхронно через шину подій ActiveMQ, яка використовується для сповіщень про зміну ролей. У рамках цього проєкту розроблено новий мікросервіс `schedule-service`, який відповідає за зберігання розкладу та координацію виклику алгоритму планування. Загальну структуру системи наведено на Рис. 13.

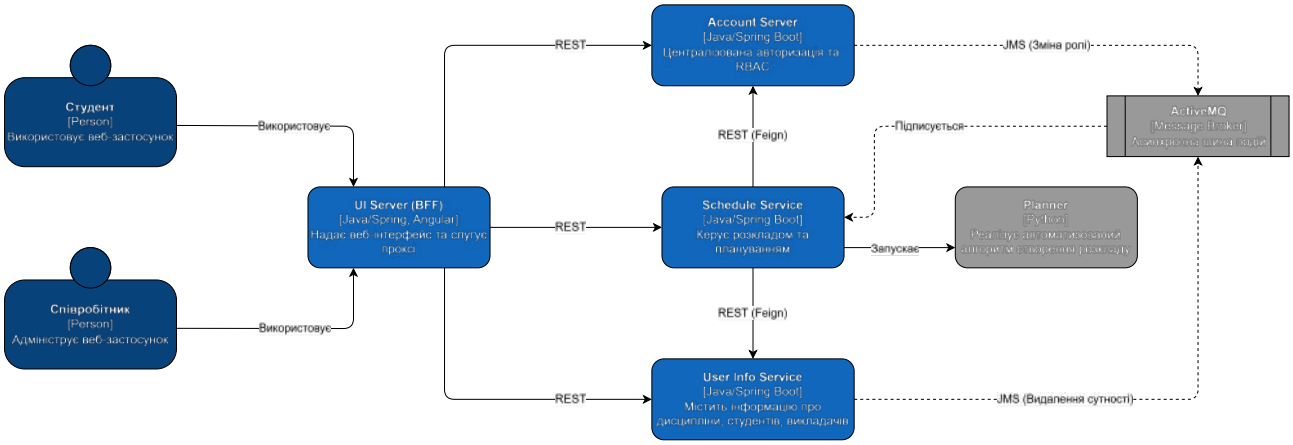

Рис. 13: C4 діаграма системи

Поєднання UNIX-конвеєра з єдиною Protobuf-схемою вирішує одночасно дві задачі. Під час розробки конвеєр запускається як самостійний інструмент командного рядка: `saz-scraper` витягує актуальні дані безпосередньо з університетської системи, `dev-augment` доповнює їх синтетичними записами для заповнення прогалин у реальних даних, `planner` виконує алгоритм складання розкладу, а `dev-viewer` дозволяє переглянути результат без залучення будь-якої веб-інфраструктури. Будь-який проміжний стан можна зупинити, зберегти у файл і відтворити повторно. У виробничому середовищі той самий виконуваний файл `planner` викликається сервісом `schedule-service` через REST API; результуючий

`ScheduleSolution` зберігається у базі даних PostgreSQL, звідки Angular-застосунок отримує його для відображення та редагування. Ключові сутності протоколу обміну наведено на Рис. 14.

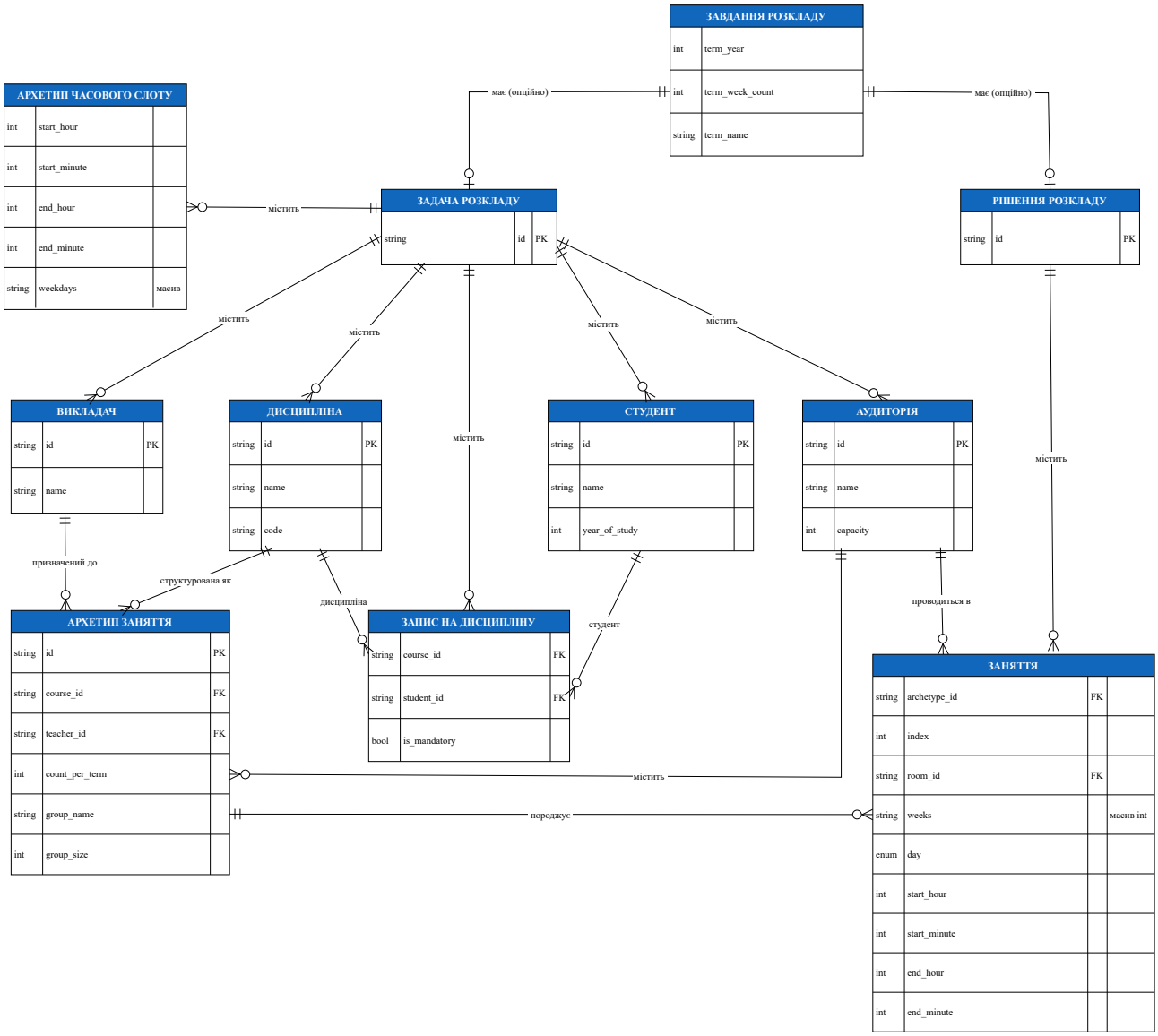

Рис. 14: ER-діаграма ключових сутностей для протоколу обміну даними

2.4 Реалізація алгоритмів та програмних модулів

Задача складання розкладу належить до класу NP-важких комбінаторних задач, тому для її розв'язання обрано CP-SAT (Constraint Programming with Satisfiability) — гібридний розв'язувач бібліотеки Google OR-Tools. На відміну від класичного цілочисельного лінійного програмування, CP-SAT поєднує три техніки: поширення обмежень (constraint propagation), навчання на конфліктах

у стилі CDCL (Conflict-Driven Clause Learning) та метод гілок і меж (branch-and-bound). Поширення обмежень на кожному кроці звужує домени змінних, видаляючи значення, несумісні з вже доданими обмеженнями; коли домен змінної стає порожнім, відбувається відкат. CDCL аналізує причину конфлікту і додає відповідну лемму до бази знань, що дозволяє уникнути аналогічних конфігурацій у майбутньому і суттєво прискорює пошук. Метод гілок і меж підтримує поточне найкраще значення цільової функції й відсікає гілки, що гарантовано не можуть дати кращого результату.

CP-SAT підтримує глобальні обмеження, яких немає в чистому ILP: зокрема, `AllDifferent` вимагає, щоб всі елементи набору мали попарно різні значення, і реалізується спеціалізованим алгоритмом поширення, а не розкладається на $O(n^2)$ попарних нерівностей. Обмеження `Element(index, table, target)` дозволяє вибирати значення з фіксованого масиву за змінним індексом — `table[index] == target` — що природно моделює пошук характеристики об’єкта (наприклад, місткості аудиторії) за її номером. Реіфіковані обмеження через `.only_enforce_if(b)` та `.only_enforce_if(b.Not())` активують або дезактивують довільне обмеження залежно від значення булевої змінної b, що дозволяє кодувати умовні залежності без введення великої кількості допоміжних конструкцій.

Розробка моделі в OR-Tools зводиться до послідовних кроків засобами Python API. Спочатку створюється об’єкт `CpModel`, до якого додаються цілочисельні змінні через `model.new_int_var(lb, ub, name)` та булеві — через `model.new_bool_var(name)`. Обмеження задаються декларативно: `model.add_all_different(vars)`, `model.add_element(idx,`

`table, target)`, `model.add(linear_expr)` для лінійних рівностей і нерівностей, а також `model.add_max_equality()`, `model.add_min_equality()`, `model.add_multiplication_equality()` та `model.add_abs_equality()` для нелінійних залежностей, що лінеаризуються через допоміжні змінні. Мінімізована цільова функція задається як `model.minimize(linear_expr)`. Часовий ліміт встановлюється параметром `max_time_in_seconds` об'єкта `SolverParameters`, а пошук запускається викликом `CpSolver().solve(model, params)`. Підказки для пришвидшення пошуку передаються через `model.add_hint(var, value)` ще до виклику `solve`.

Для кожного архетипу i вводяться дві цілочисельні змінні рішення: $s_i \in \{0, ..., N_s - 1\}$ — індекс часового слоту та $r_i \in \{0, ..., N_r - 1\}$ — індекс аудиторії, де N_s і N_r — загальна кількість слотів і аудиторій відповідно. Два допоміжних змінних d_i та tod_i представляють день тижня та час усередині дня, пов'язані співвідношенням $s_i = d_i \cdot K + \mathrm{tod}_i$, де K — кількість слотів на день. Додатково вводиться «пласка» змінна $\mathrm{sr}_i = s_i \cdot N_r + r_i$, що кодує пару (слот, аудиторія) єдиним цілим числом.

Жорсткі обмеження формують простір допустимих розкладів. Обмеження відсутності накладень в аудиторії (C1) задається як $\mathrm{AllDifferent}(\{\mathrm{sr}_i\})$ для всіх архетипів: оскільки sr_i однозначно кодує пару (слот, аудиторія), рівність двох таких значень означала б одночасне проведення двох занять в одній аудиторії. Обмеження місткості аудиторії (C2) реалізується через `add_element`: для кожного архетипу i значення $\mathrm{caps}[r_i]$ вибирається з масиву місткостей аудиторій і порівнюється з розміром групи, $\mathrm{caps}[r_i] \geq \mathrm{size}_i$. Обмеження відсутності накладень у викладача (C3) задається для кожного викладача t як

$\mathrm{AllDifferent}(\{s_i \mid \mathrm{teacher}(i) = t\})$. Обмеження порядку лекція-практична (C4) вимагає, щоб для кожного курсу слот лекції суворо передував слоту практичного заняття: $s_{\text{лек}} < s_{\text{пр}}$. Обмеження потоку (C5) забезпечує відсутність конфліктів у обов'язкових дисциплінах студентських потоків: для кожного потоку лекції не можуть збігатися в часі (AllDifferent на s_i усіх лекцій потоку), а лекція певної дисципліни не може збігатись із практичними заняттями інших дисциплін того самого потоку.

Якість розкладу визначається шістьма м'якими критеріями, що об'єднані у зважену суму яка мінімізується:

$$\min \sum_{k=1}^{6} w_k \cdot P_k.$$

Критерій внутрішньоденного інтервалу (P_1) мінімізує для архетипів одного курсу на одному дні різницю $\max(\mathrm{tod}) - \min(\mathrm{tod})$: розрив між першим і останнім заняттям дня обчислюється через `add_max_equality` та `add_min_equality`. Критерій денного розмаху (P_2) мінімізує для кожного курсу різницю $\max(d) - \min(d)$, заохочуючи концентрувати всі заняття курсу в якомога менше днів. Критерій лекція-практична (P_3) мінімізує абсолютну відстань за слотами між лекцією та практичним заняттям того самого курсу. Критерій розподілу потоку (P_4) штрафує розміщення двох курсів одного потоку в один і той самий день: реіфікована булева змінна b набуває одиниці при збігу цільових днів курсів через обмеження `.only_enforce_if(b)` та `.only_enforce_if(b.Not())`. Критерій пізніх слотів (P_5) обчислює штраф через `add_element` з попередньо порахованим масивом: базовий штраф зростає квадратично як $\mathrm{tod}^2 \cdot 10$, а слоти після 18:00 отримують фіксований великий штраф, що фактично унеможливлює дуже пізні заняття. Критерій рівномірності денного навантаження (P_6) мінімізує суму ква-

дратів кількості занять по кожному дню $\sum_d n_d^2$, де піднесення до квадрату реалізується через `add_multiplication_equality`, рівномірно розподіляючи навчальне навантаження протягом тижня. Конкретні значення ваг ω_k підібрані емпірично на реальних даних НаУКМА.

Через комбінаторну складність задачі реалізовано декомпозицію на три етапи. Перша фаза обробляє лише архетипи, обов'язкові для десяти і більше студентських потоків. До них належать загальноуніверситетські курси — «Українська мова», «Англійська мова» та суміжні дисципліни, обов'язкові для 1000–1200 студентів молодших курсів. Їх необхідно розмістити першими, оскільки конфлікти в масових курсах неможливо локально усунути на пізніших фазах. Друга фаза розміщує всі обов'язкові архетипи; результати першої фази фіксуються — для відповідних архетипів змінні s_i та r_i отримують домен з одного значення: `model.new_int_var(val, val, name)`, що унеможливлює їх зміну. Усі п'ять жорстких обмежень та шість м'яких критеріїв є активними. Третя фаза виконує переоптимізацію на рівні студентів: для кожної студентської групи g, курсу c та секції j вводяться булеві змінні-вибірники $p_{g,c,j}$, індикатори конфліктів та змінні завантаженості. Ціль третьої фази — мінімізувати послідовно кількість конфліктів між обов'язковими курсами H, між вибірковими курсами Q та переповнення аудиторій O з ієрархією ваг $w_7 \gg w_8 \gg w_9$.

Навіть за правильно сформульованої моделі CP-SAT потребує розумної початкової точки, інакше пошук витрачає значний час на дослідження нежиттєздатних областей. Реалізовано конструктивну евристику з впорядкуванням за стовбцями: кожному курсу призначається цільовий день тижня на основі зваженої оцінки, що враховує глобальне навантаження дня, локальне скупчення курсів

одного потоку та відстань від цільового зміщення потоку. Зміщення кожного потоку обчислюється за формулою $(N_{\text{зан}} \cdot c + \text{idx}) \bmod N_{\text{днів}}$, де $N_{\text{зан}}$ — загальна кількість занять, а c — емпірично підібрана константа множник. Завдяки різним зміщенням різні потоки тяжіють до різних днів тижня, що запобігає характерному для жадібних алгоритмів скупченню занять на початку тижня. Лекції кожного курсу розміщуються на цільовому дні раніше за практичні заняття, а обчислені позиції передаються солверу через `model.add_hint()`: CP-SAT може відхилятися від підказок при порушенні жорстких обмежень, але використовує їх як початкову точку пошуку, суттєво скорочуючи час до першого допустимого розв'язку.

Розроблена реалізація поєднує декларативний опис задачі через систему жорстких обмежень та зваженої цільової функції з практичними евристиками — трифазовою декомпозицією, конструктивним початковим розміщенням та пріоритизацією найбільш обмежених змінних, — що разом забезпечують знаходження якісного допустимого розкладу для реального масштабу задачі НаУКМА в межах практично прийнятного часу.

2.5 Інтеграція з існуючою системою університету

Алгоритм складання розкладу, реалізований як автономний конвеєр командного рядка, є інструментом для розробника, а не для кінцевого користувача. Щоб розклад, сформований алгоритмом, став доступним студентам і адміністрації університету, його необхідно вбудувати в існуючу цифрову інфраструктуру. Замість розробки окремої ізольованої системи прийнято рішення розширити платформу Smart-UKMA: такий підхід дозволяє повторно використати вже наявні механізми автентифікації, керування ролями та Angular-інтерфейс, знайомий університетській спільноті.

Щоб відповідати стандартам платформи, новий мікросервіс `schedule-service` потребував повноцінного REST API і власної бази даних. Сервіс розроблено засобами Java та Spring Boot відповідно до архітектурних підходів, прийнятих у платформі. REST API організовано навколо чотирьох ресурсів: навчальні семестри, часові слоти, розклади та окремі заняття. Доступ регулюється двома дозволами — `VIEW_SCHEDULE` для читання та `MANAGE_SCHEDULE` для повного управління, — які перевіряються через спільний механізм JWT-токенів платформи.

Схема бази даних відображає три окремі задачі: зберігання самого розкладу, відстеження запусків планувальника та збереження занять із прив'язкою до реальних тижнів семестру. Таблиця `schedule` зберігає записи розкладів із мітками часу створення, публікації й оновлення. Таблиця `scheduled_lesson` зберігає ідентифікатори дисципліни, викладача та аудиторії як зовнішні посилання на дані з `user-info` — сервіс не дублює ці дані, а посилається на них. Оскільки одне заняття може проводитись не щотижня, дочірня таблиця `scheduled_lesson_week` зберігає список тижнів семестру, в яких воно відбувається, дозволяючи задавати довільні нерегулярні патерни. Таблиця `planner_job` фіксує кожен запуск планувальника зі статусом і часовими мітками — саме ці дані транслюються в інтерфейс для відображення прогресу в реальному часі.

Оскільки `schedule-service` не є власником навчальних даних — дисципліни, викладачі та аудиторії зберігаються у `user-info`, — сервіс запитує ці дані через Feign REST-клієнти в момент запуску планувальника. Клас `ScheduleTaskBuilder` отримує всі пари «дисципліна — викладач» разом із погодинним розподілом навантаження, а `LessonArchetypeDeriver` перетворює їх на

архетипи занять: один архетип на пару (дисципліна, лектор) для лекцій та один архетип на трійку (дисципліна, викладач, номер групи) для практичних. Аудиторії та факультети отримуються з того самого сервісу і додаються до опису задачі.

Окремою задачею є підтримка ітеративного робочого процесу: адміністратор може вручну скоригувати частину занять, а потім перезапустити планувальник для решти — без втрати зроблених правок. Це вирішує параметр `ignoreFixed` в ендпоінті запуску планувальника: якщо він не встановлений, вже розміщені заняття передаються солверу як фіксовані обмеження. Оскільки виконання може тривати кілька хвилин, запуск здійснюється асинхронно: клас `PlannerJobRunner` запускає Python-файл солвера як підпроцес, передає Protobuf-повідомлення `ScheduleTask` через stdin і зчитує розв'язок зі stdout, фіксуючи хід виконання зі статусами QUEUED → RUNNING → COMPLETED / FAILED. Після завершення `PlannerJobService` зіставляє ідентифікатори архетипів із даними дисциплін, викладачів і груп та зберігає записи занять у базі даних.

Ще одна проблема — узгодженість даних: якщо викладача або дисципліну видалено з `user-info`, відповідні заняття у `schedule-service` стають посиланнями на неіснуючі сутності. Пряма залежність між сервісами порушила б принцип ізоляції платформи, тому рішення реалізовано через спільну шину подій ActiveMQ: клас `UserInfoEntityDeletedEventListener` підписується на події видалення і каскадно прибирає пов'язані заняття або знімає прив'язку видаленої аудиторії.

Розклад існує у двох станах — чернетка та опублікований, — що розділяє ітеративний процес генерації від студентського доступу. Поки розклад є чернеткою, його можна вільно редагувати, перезапускати планувальник і видаляти без жодних наслідків для студентів. Публікація робить розклад видимим у перегля-

дачі; при цьому заняття залишаються редагованими і після публікації — поле `updatedAt` фіксує факт кожної зміни, тоді як сам розклад залишається опублікованим.

Перед початком роботи зі складання розкладу адміністратор налаштовує базові параметри: семестр (навчальний рік, тип — осінній, весняний або літній — та кількість тижнів) та доступні часові слоти (час початку, закінчення та дні тижня). Ці параметри є обов'язковою передумовою для будь-якої подальшої роботи з розкладом.

З боку інтерфейсу Angular-застосунок розширено повноцінним модулем управління розкладом. Адміністративний процес починається на сторінці-списку, де відображаються всі розклади обраного семестру зі статусом останньої задачі планувальника поруч із кожним. Звідси адміністратор може запустити планувальник — із збереженням або без збереження фіксованих занять, — спостерігати за ходом задачі в реальному часі та опублікувати готовий розклад. Основний інструмент редагування — сіткова таблиця (Рис. 15), де рядки відповідають дням тижня, а стовпці — часовим слотам, — відтворює звичний для фахівців спосіб мислення про розклад і мінімізує когнітивне навантаження. Заняття можна переміщувати перетягуванням, фільтрувати за тижнями для перегляду нерегулярних занять та редагувати через модальне вікно. Сторінка підходить як для повністю ручного складання розкладу, так і для внесення коректив у автоматично згенерований.

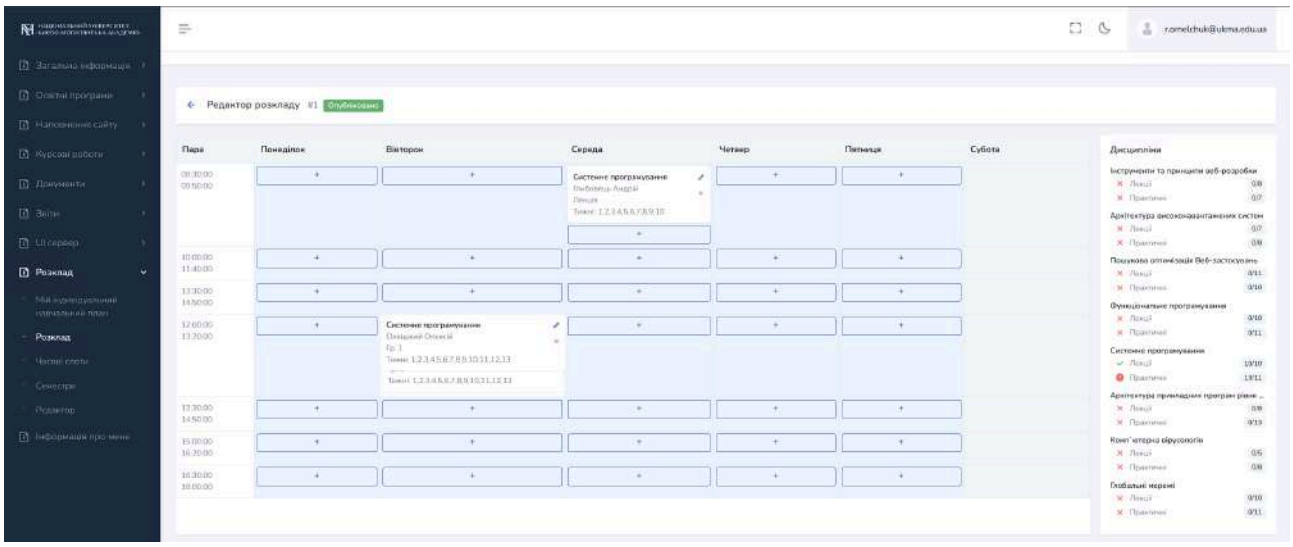

Рис. 15: Веб-сторінка редактора розкладу

Для студентів та інших користувачів із доступом лише на читання реалізовано переглядач розкладу (Рис. 16). Два режими задовольняють різні потреби: персональний фільтрує заняття відповідно до навчального плану конкретного студента, потоковий — надає повний розклад обраної групи або спеціальності. Підтримка тижневого та місячного форматів з коректним урахуванням тижневої структури семестру дозволяє планувати навчальний час з різним горизонтом. Кожне заняття відображається з повним набором деталей — дисципліною, викладачем, аудиторією та типом заняття, — що знімає потребу звертатись до додаткових джерел інформації.

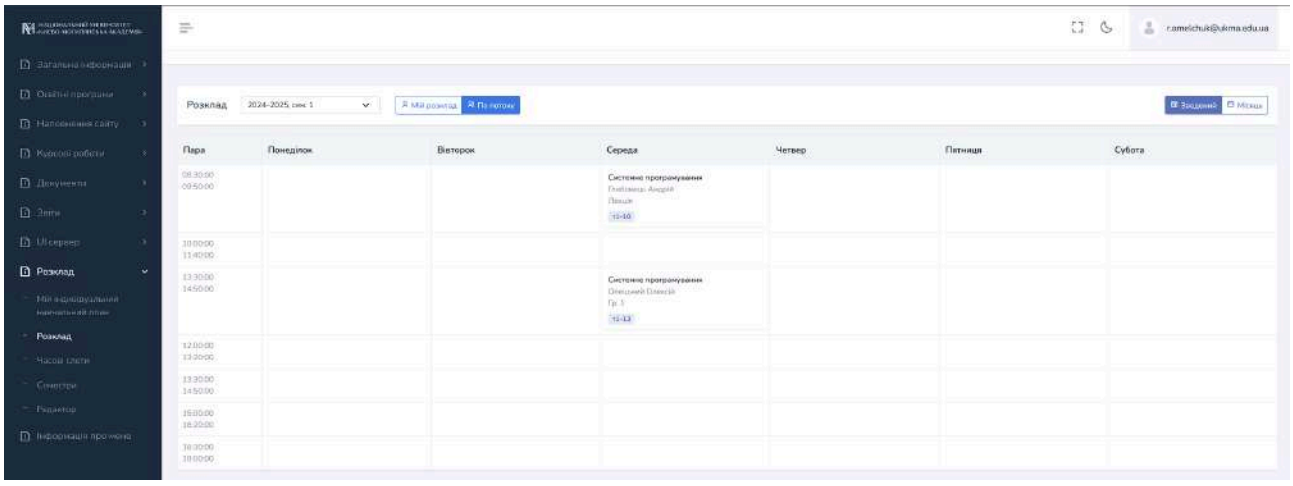

Рис. 16: Веб-сторінка перегляду розкладу

З точки зору обсягу змін у платформі, `schedule-service` є цілком новим мікросервісом, написаним з нуля, тоді як два наявні сервіси потребували доповнень. У `user-info` необхідно було додати механізм публікації подій видалення через ActiveMQ — щоб `schedule-service` міг підтримувати узгодженість даних без прямої залежності, — а також розширити REST API для надання даних про навчальні плани та прив’язки «дисципліна — викладач», яких раніше у відкритому доступі не було, та додати сутності аудиторій і корпусів, яких у системі взагалі не існувало. У `account-server` потрібно було зареєструвати `schedule-service` як повноцінного учасника системи RBAC: визначити його ролі — від студентського читання до адміністративного управління — і відобразити відповідні права в інтерфейсі керування користувачами.

3 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Оцінка якості згенерованих розкладів

Оцінка алгоритму проводилась на напівсинтетичних даних: реальні записи студентів НаУКМА на осінній семестр 2025 року (28 012 записів, 4 366 студентів, 574 курси) були доповнені синтетичними ресурсами — викладачами та аудиторіями — за допомогою модуля `dev-augment`. Така необхідність зумовлена обмеженням системи САЗ: вона надає доступ лише до даних про записи студентів, але не містить інформації про реальний розподіл аудиторій та розклади викладачів. Тестовий сценарій використовував 201 аудиторію та викладачів у кількості, заданій відсотком від числа архетипів занять, — ресурсний пул, достатній для розміщення всіх занять без суттєвих ресурсних обмежень. Відповідно, оцінка відображає поведінку алгоритму за сприятливих ресурсних умов, а не максимально обмеженого реального розгортання.

Першочерговим критерієм коректності є задоволення жорстких обмежень. Планувальник розмістив усі 2 008 архетипів занять (100% повнота розміщення). Перевірка студентських шляхів виявила лише 2 студенти з конфліктами в обов’язкових дисциплінах із 4 578 перевірених — показник 0,04%, що практично відповідає нульовому рівню конфліктів. Серед вибіркових дисциплін 8 161 із 8 556 записів (95,38%) забезпечені безконфліктним розміщенням. Два залишкові конфлікти обов’язкових дисциплін, ймовірно, зумовлені особливостями конкретних записів у тестовому наборі й не спростовують загальної коректності підходу.

Розподіл занять за днями тижня та часовими слотами наведено на Рис. 17. Навантаження є відносно рівномірним між днями, проте спостерігається незначне зміщення на кінець тижня — відомий артефакт стовпцевої евристики при

поточній конфігурації ваг. Погодинний розподіл виявляє характерну ранкову концентрацію: слоти 08:00 та 10:00 домінують, тоді як вечірні слоти після 18:00 задіяні лише у 3,7% випадків. Така структура є наслідком поточної конфігурації ваги штрафу за пізні слоти (P_5) та може бути скоригована підвищенням ваги критерію рівномірності денного навантаження (P_6).

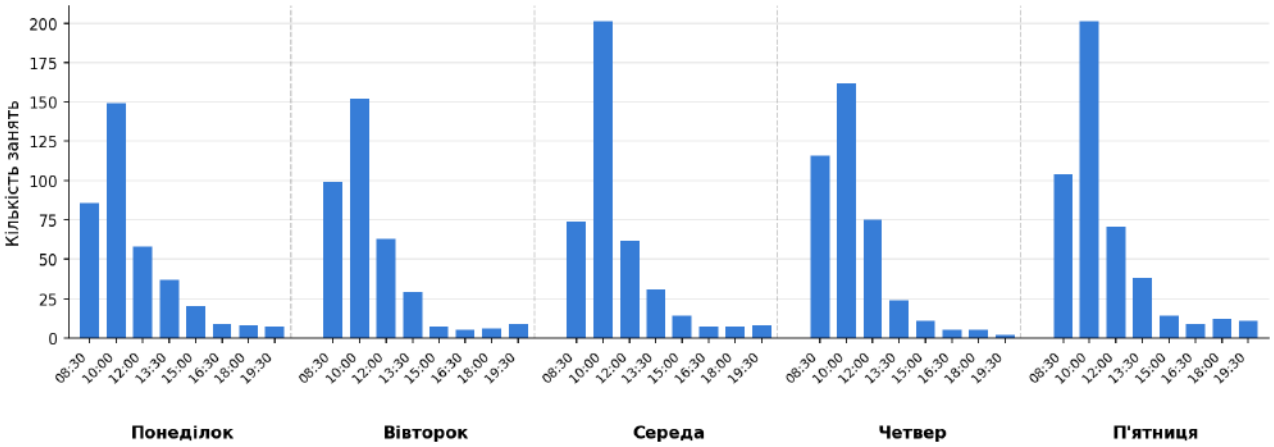

Рис. 17: Розподіл занять за днями тижня та часовими слотами

Отримані результати слід інтерпретувати в контексті рівня зрілості системи. Використання синтетичних ресурсів означає, що алгоритм не стикався з реальними обмеженнями — переповненими аудиторіями, недоступністю конкретних викладачів, прив'язкою курсів до спеціалізованих приміщень. У реальному розгортанні ці чинники суттєво підвищують складність задачі. Крім того, якість м'яких показників — зокрема ранкова концентрація та нерівномірне навантаження в кінці тижня — поступається вручну складеним розкладам, де досвідчений адміністратор враховує неформальні пріоритети. Система є доведенням концепції: вона демонструє коректність трифазової декомпозиції та практичну реалізовність автоматичного складання розкладу для НаУКМА, залишаючи подальше калібрування ваг та інтеграцію реальних ресурсних даних для наступних етапів розробки.

3.2 Аналіз ефективності розробленого інструментарію

Розроблений інструментарій обслуговує дві категорії користувачів з різними потребами: студентів, яким необхідний зручний доступ до опублікованого розкладу, та адміністраторів факультетів, яким потрібен повний цикл від автоматичної генерації до публікації. Відповідно до цих ролей система надає три функціональні шари — переглядач, редактор та автоматизований планувальник, — кожен з яких вирішує конкретну задачу, але разом вони утворюють єдиний інтегрований процес.

Найсуттєвішою з точки зору практичної ефективності є можливість автоматичної генерації розкладу. Процес складання розкладу для великого університету вручну — це задача, що традиційно займає кілька тижнів і потребує значної координації між факультетами. Розроблений планувальник скорочує цей процес до хвилин. Виконання здійснюється асинхронно — планувальник запускається у фоновому режимі, а адміністратор спостерігає за ходом задачі в реальному часі через інтерфейс, що періодично оновлюється. Параметр `ignoreFixed` дозволяє використовувати планувальник ітеративно: зберігаючи вже відредаговані вручну заняття фіксованими, адміністратор може перезапустити автоматичну генерацію для незаповнених частин розкладу, не втрачаючи попередніх коректив.

Автоматично згенерований розклад, однак, часто потребує ручного доопрацювання, і тут ключову роль відіграє редактор розкладу. Основний робочий інструмент — сіткова таблиця, де рядки відповідають дням тижня, а стовпці — доступним часовим слотам, — відтворює звичний для фахівців спосіб мислення про розклад і мінімізує когнітивне навантаження під час редагування. Переміщення занять через перетягування (drag-and-drop) дозволяє швидко усувати

конфлікти без навігації між формами. Фільтр за тижнями враховує нерегулярну структуру семестру: заняття може проводитись не щотижня, і редактор коректно відображає цю інформацію. Поділ на стани «чернетка — опублікований» надає процесу чіткий хід: адміністратор може вільно редагувати й перезапускати планувальник, доки розклад є чернеткою, і публікувати його лише після завершення перевірки.

Переглядач розкладу забезпечує доступ до опублікованої версії для студентів та всіх інших зацікавлених сторін. Два режими перегляду задовольняють різні потреби: персональний фільтрує заняття відповідно до навчального плану конкретного студента, тоді як потоковий надає повний розклад обраної групи або спеціальності. Підтримка тижневого та місячного форматів з коректним урахуванням тижневої структури семестру дозволяє планувати навчальний час з різним горизонтом. Кожне заняття відображається з повним набором деталей — дисципліною, викладачем, аудиторією та типом заняття, — що знімає потребу звертатись до додаткових джерел інформації.

Важливим виміром ефективності є також безшовність інтеграції у наявну платформу: розроблений модуль не вимагає від університетської спільноти жодних нових облікових записів або знайомства з окремою системою, а узгодженість даних підтримується автоматично без втручання адміністратора.

Таким чином, розроблений інструментарій реалізує повний цикл роботи зі складання розкладу: від автоматичної генерації через редагування до публікації та перегляду. Взаємодоповнювальність трьох шарів — планувальника, редактора та переглядача — разом із безшовною інтеграцією у платформу Smart-UKMA перетворює систему із набору ізольованих технологічних компонентів на практичний інструмент для університетської адміністрації.

ВИСНОВКИ

У роботі досліджено задачу автоматизованого складання розкладу для НаУКМА — університету з ліберальною освітньою моделлю, де відсутність фіксованих академічних груп і процес формування індивідуального навчального плану породжують гібридну задачу CB/PE-CTP зі структурно асиметричними підзадачами. Статистичний аналіз реальних даних підтвердив цю асиметрію кількісно: граф конфліктів обов'язкових дисциплін є розрідженим (щільність 1,85%, $\omega(G) = 8$), що робить безконфліктний розклад для них теоретично реалізовним, тоді як граф вибіркових дисциплін є щільним і міжфакультетським (щільність 8,01%, $\omega(G) = 14$) — повне усунення конфліктів у ньому практично недосяжне. Ця асиметрія безпосередньо визначила архітектуру рішення.

На основі аналізу розроблено математичну модель у вигляді задачі задоволення обмежень з оптимізацією: п'ять жорстких обмежень гарантують допустимість розкладу, шість м'яких критеріїв якості об'єднано у зважену цільову функцію, а для третьої фази введено розширену модель вибору студентами навчальних секцій. Реалізовано трифазову декомпозицію на основі CP-SAT (Google OR-Tools): перша фаза пріоритетно розміщує загальноуніверситетські курси-«вузькі місця», друга — решту обов'язкових архетипів, третя — оптимізує розподіл студентів за секціями, мінімізуючи конфлікти в індивідуальних навчальних планах. Для скорочення часу пошуку реалізовано конструктивну евристику ініціалізації з цільовими зміщеннями занять різних потоків.

Розроблений алгоритм інтегровано у платформу Smart-UKMA у вигляді мікросервісу `schedule-service`. Взаємодія між Python-компонентом планувальника та Java/Spring Boot мікросервісами реалізована через UNIX-конвеєр та

Protobuf-протокол; адміністраторам надано Angular-інтерфейс із повним циклом від автоматичної генерації до редагування та публікації розкладу.

Перевірка алгоритму на напівсинтетичних даних осіннього семестру 2025 року показала, що планувальник розміщує 100% архетипів занять; лише 2 із 4 578 перевірених студентів (0,04%) мають конфлікти в обов'язкових дисциплінах, а 95,38% вибіркових записів є безконфліктними. Оскільки тестовий сценарій використовував синтетичні ресурси, результати відображають поведінку алгоритму за сприятливих умов, а не в умовах реального ресурсного обмеження. Якість м'яких показників поступається вручну складеним розкладам, проте виявлені відхилення є виправними через калібрування ваг. Отримані показники свідчать про коректність трифазового підходу та його придатність як основи для подальшого розвитку.

Подальший розвиток системи потребує вирішення як алгоритмічних, так і системних завдань. З боку алгоритму — інтеграції реальних даних про аудиторії та розклади викладачів, врахування побажань викладачів щодо зручного часу та вимог до приміщень (наявність проєктора, комп'ютерного обладнання тощо), а також систематичного калібрування ваг цільової функції на основі порівняння з еталонним розкладом. З боку системи — доопрацювання бізнес-логіки, покращення зручності інтерфейсу для адміністраторів та всебічного тестування в умовах реального навчального процесу.

СПИСОК ЛІТЕРАТУРИ

1. Even S. On the Complexity of Timetable and Multicommodity Flow Problems / S. Even, A. Itai, A. Shamir // SIAM Journal on Computing. — 1976. — Vol. 5, No. 4. — P. 691–703. — DOI: 10.1137/0205048.
2. Garey M. R. Computers and Intractability: A Guide to the Theory of NP-Completeness / M. R. Garey, D. S. Johnson. — New York : W. H. Freeman, 1979. — 340 p. — ISBN 0-7167-1045-5.
3. Aguirre Lam M. A. University Timetabling Problem: An Analysis of the State of the Art / M. A. Aguirre Lam, F. Balderas, N. Rangel-Valdez, J. A. Martinez-Flores, A. G. Aguirre L., J. A. Pérez-Vázquez // International Journal of Combinatorial Optimization Problems and Informatics. — 2025. — Vol. 16, No. 3. — P. 489–498. — DOI: 10.61467/2007.1558.2025.v16i3.852.
4. Abdipoor S. Meta-heuristic approaches for the University Course Timetabling Problem / S. Abdipoor, R. Yaakob, S. L. Goh, S. Abdullah // Intelligent Systems with Applications. — 2023. — Vol. 19. — URL : https://www.sciencedirect.com/science/article/pii/S2667305323000789 (дата звернення: 15.05.2026).
5. Alghamdi H. A Review of Optimization Algorithms for University Timetable Scheduling / H. Alghamdi, T. Alsubait, H. Alhakami, A. Baz // Engineering, Technology & Applied Science Research. — 2020. — Vol. 10, No. 6. — P. 6410–6417. — DOI: 10.48084/etasr.3832.
6. Islam T. University Timetable Generator Using Tabu Search / T. Islam, Z. Shahriar, M. A. Perves, M. Hasan // Journal of Computer and Communications. — 2016. — Vol. 4, No. 16. — P. 28–37. — DOI: 10.4236/jcc.2016.416003.

7. Salem D. University Examination Timetable Scheduling Using Constructive Heuristic Compared to Genetic Algorithm / D. Salem // Fayoum University Journal of Engineering. — 2023. — DOI: 10.21608/fuje.2022.157195.1018.
8. Biswas S. Graph Coloring in University Timetable Scheduling / S. Biswas, S. A. Nusrat, N. Sharmin, M. Rahman // International Journal of Intelligent Systems and Applications. — 2023. — DOI: 10.5815/ijisa.2023.03.02.
9. Akı O. University Course Timetabling Using Genetic Algorithms [Електронний ресурс] / O. Akı // Proceedings of the International Scientific Conference UNITECH'2020. — Gabrovo, Bulgaria, 2020. — URL : https://www.researchgate.net/publication/346969094 (дата звернення: 15.05.2026).
10. Krause J. Combinatorial Optimization / J. Krause, T. Kuen, C. Scholl // Unlocking Artificial Intelligence. — Springer, 2024. — P. 137–152. — DOI: 10.1007/978-3-031-64832-8_7.
11. Diallo F. P. Optimizing the Scheduling of Teaching Activities in a Faculty / F. P. Diallo, C. Tudose // Applied Sciences. — 2024. — DOI: 10.3390/app14209554.
12. Ajanovski V. V. Integration of a Course Enrolment and Class Timetable Scheduling in a Student Information System / V. V. Ajanovski // International Journal of Database Management Systems. — 2013. — DOI: 10.5121/ijdms.2013.5107.
13. International Timetabling Competition 2019 (ITC2019) [Електронний ресурс]. — URL : https://www.itc2019.org/home#overview (дата звернення: 15.05.2026).
14. Timefold Solver [Електронний ресурс] / Timefold. — URL : https://timefold.ai/ (дата звернення: 15.05.2026).
15. Google OR-Tools [Електронний ресурс] / Google LLC. — URL : https://developers.google.com/optimization (дата звернення: 15.05.2026).

16. Усачов К. Розробка автоматизованої системи складання розкладу : маг. робота / К. Усачов ; НаУКМА. — Київ, 2020. — URL : https://ekmair.ukma.edu.ua/items/6b790011-3421-45a0-b64e-2960791312dd (дата звернення: 15.05.2026).
17. Леськів О. Розробка системи управління аудиторіями «KMAuditoriums» : бакал. робота / О. Леськів ; НаУКМА. — Київ, 2020. — URL : https://ekmair.ukma.edu.ua/items/c3659f43-706d-4ce7-90a5-89f50ad0258c (дата звернення: 15.05.2026).
18. Синицин В. Розробка системи вступу в НаУКМА : бакал. робота / В. Синицин ; НаУКМА. — Київ, 2022. — URL : https://ekmair.ukma.edu.ua/items/8d2a6055-f770-4005-9c96-b0fd8d7113db (дата звернення: 15.05.2026).
19. Müller T. Real-world university course timetabling at the International Timetabling Competition 2019 / T. Müller, H. Rudová, Z. Müllerová // Journal of Scheduling. — 2024. — Vol. 28, No. 2. — P. 247–267. — DOI: 10.1007/s10951-023-00801-w.
20. Rudová H. Real-life Curriculum-based Timetabling / H. Rudová, T. Müller // 9th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012). — 2012. — URL : https://www.unitime.org/present/patat12.pdf (дата звернення: 15.05.2026).
21. UniTime APIs [Електронний ресурс] / UniTime project. — 2019. — URL : https://help.unitime.org/manuals/api (дата звернення: 15.05.2026).