МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ЛЕЩЕНКО ВЛАДИСЛАВ ОЛЕКСАНДРОВИЧ

Допускається до захисту:
в. о. завідувача кафедри
інформаційних технологій,
канд. техн. наук, доцент
_____Оксана ЗЕЛІНСЬКА
«____»_____ 2024 р.

ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ ПІДТРИМКИ ДІЯЛЬНОСТІ СТРУКТУРНОГО ПІДРОЗДІЛУ ЗАКЛАДУ ВИЩОЇ ОСВІТИ

Спеціальність 122 «Комп'ютерні науки»

Кваліфікаційна (магістерська) робота

Науковий керівник:
О.В. Зелінська, доцент кафедри
інформаційних технологій,
канд. техн. наук, доцент
_____
(підпис)

Оцінка: ___/___/_____
(бали за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця 2024

АНОТАЦІЯ

Лещенко В. О. Інформаційна система для підтримки діяльності структурного підрозділу закладу вищої освіти. Спеціальність 122 «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2024.

Кваліфікаційна (магістерська) робота присвячена розробці інформаційної системи для автоматизованого складання розкладу занять у закладі вищої освіти. Проведено аналіз існуючих підходів до планування навчального процесу, їхні переваги та недоліки. Описано предметну область і сформульовано методику автоматизованого складання розкладу, враховуючи специфіку структурного підрозділу університету. Основна увага приділяється проектуванню програмної архітектури, розробці алгоритму складання розкладу, включаючи жадібний алгоритм та механізми перевірки обмежень, а також розробці функціоналу системи та її тестуванню.

Ключові слова: інформаційна система, автоматизоване складання розкладу, заклад вищої освіти, Python, планування навчального процесу, жадібний алгоритм.

60 с., 2 табл., 15 рис., 2 дод., 39 джерел.

ABSTRACT

Leshchenko V.O. Information System for Supporting the Activities of a Structural Unit of a Higher Education Institution. Specialty 122 "Computer Science". Vasyl’ Stus Donetsk National University, Vinnytsia, 2024.

The qualification (master's) thesis is devoted to the development of an information system for automated scheduling of classes in a higher education institution. The analysis of existing approaches to the planning of the educational process, their advantages, and disadvantages has been conducted. The subject area is described, and a methodology for automated scheduling is formulated, considering the specifics of the university's structural unit. The main focus is on designing the software architecture, developing the scheduling algorithm, including a greedy algorithm and

constraint checking mechanisms, as well as developing the system's functionality and testing it.

Keywords: information system, automated scheduling, higher education institution, Python, educational process planning, greedy algorithm.

60 pages, 2 tables, 15 figures, 2 appendices, 39 references.

ЗМІСТ

ВСТУП

Сучасні заклади вищої освіти стикаються з необхідністю ефективного управління навчальним процесом, особливо в контексті планування розкладу занять. Ручне складання розкладу є трудомістким та схильним до помилок процесом, який потребує врахування безлічі факторів: дисциплін, груп студентів, викладачів, аудиторій тощо. Це підкреслює актуальність розробки автоматизованих систем, які можуть оптимізувати цей процес і забезпечити більш ефективне використання ресурсів закладу.

Об’єкт дослідження: інформаційна система для підтримки діяльності структурного підрозділу закладу вищої освіти.

Предмет дослідження: проектування та розробка інформаційної системи для автоматизованого складання розкладу занять, спрямованого на оптимізацію планування та управління навчальним процесом.

Мета досліджень: набуття практичних навичок у розробці інформаційних систем, зокрема створення застосунку для автоматичного формування розкладу занять.

Актуальність задачі обумовлена зростаючими вимогами до оперативності та точності планування навчального процесу. Впровадження автоматизованої системи розкладу не лише спрощує адміністративну роботу, але й підвищує задоволеність студентів та викладачів завдяки більш гнучкому та зручному розкладу [1].

Основні завдання дослідження включають:

- Аналіз існуючих систем автоматизації розкладу та визначення їх переваг і недоліків.
- Розробка методики автоматизованого формування розкладу, яка враховує специфіку структурного підрозділу університету.
- Створення програмного забезпечення, яке реалізує розроблену методику.
- Тестування та впровадження системи у реальні умови роботи кафедри.

Очікувані результати передбачають підвищення ефективності планування навчального процесу та можливість подальшого впровадження системи в інших структурних підрозділах університету.

Методика наукового дослідження базується на використанні системного підходу та сучасних інформаційних технологій. Для вирішення поставлених завдань застосовуються такі методи та моделі:

- Алгоритми оптимізації та штучного інтелекту для розв'язання задачі складання розкладу з множиною обмежень [2].
- Об'єктно-орієнтоване програмування для розробки гнучкої та масштабованої системи.
- Бази даних для зберігання та управління інформацією про дисципліни, групи, викладачів та аудиторії.

Практична цінність отриманих результатів полягає у можливості їх безпосереднього використання в процесах планування та управління навчальним процесом кафедри. Розроблена система сприятиме прийняттю більш обґрунтованих рішень, зменшить навантаження на адміністративний персонал та покращить якість освітніх послуг.

РОЗДІЛ 1
ТЕОРЕТИКО-МЕТОДИЧНІ ОСНОВИ АВТОМАТИЗАЦІЇ СКЛАДАННЯ РОЗКЛАДУ ЗАНЯТЬ

1.1 Опис предметної області

Розклад занять є одним із ключових інструментів управління навчальним процесом у закладах вищої освіти. Він забезпечує координацію між студентами, викладачами та ресурсами університету, такими як аудиторії та обладнання. Ефективний розклад сприяє оптимальному використанню ресурсів, підвищує продуктивність та задоволеність усіх учасників освітнього процесу [3].

У сучасних університетах, де пропонуються різноманітні навчальні програми та спеціальності, складання розкладу стає дедалі складнішим завданням. Зростає потреба в інтеграції міждисциплінарних курсів, гнучких графіків та індивідуальних навчальних траєкторій студентів [4]. Це вимагає більш ефективних методів планування, які можуть адаптуватися до змін та враховувати різноманітні обмеження та пріоритети.

Розклад також впливає на якість освіти, оскільки від нього залежить рівномірність навантаження студентів та викладачів, можливість проведення занять у оптимальний час та забезпечення необхідних умов для навчання. Правильно складений розклад може зменшити стрес, пов'язаний з перенавантаженням або конфліктами у графіку, та покращити загальний досвід навчання.

Ручне складання розкладу занять є трудомістким та складним процесом, що часто призводить до ряду проблем:

- **Велика кількість змінних та обмежень**: Необхідно враховувати розклад викладачів, доступність аудиторій, особливості навчальних дисциплін, побажання студентів та викладачів.

- **Схильність до помилок**: Людський фактор може призводити до помилок, таких як подвійне бронювання аудиторій або накладання занять для однієї групи студентів.
- **Тривалість процесу**: Складання розкладу може займати від кількох днів до тижнів, що є неефективним використанням ресурсів адміністрації.
- **Нездатність швидко реагувати на зміни**: Будь-які зміни, як-от відсутність викладача або додавання нового курсу, вимагають перегляду значної частини розкладу.
- **Обмежена оптимальність**: Ручне складання рідко призводить до оптимального розподілу ресурсів, оскільки складно врахувати всі можливі комбінації та варіанти.

Ці виклики підкреслюють необхідність у впровадженні автоматизованих систем, які можуть ефективніше вирішувати задачу складання розкладу, мінімізуючи помилки та час на планування.

Задача складання розкладу відноситься до класу складних комбінаторних задач, відомих як NP-складні [5]. Вона полягає у призначенні набору занять до певних часових слотів та ресурсів (аудиторій, викладачів) з урахуванням набору обмежень.

Основні компоненти задачі**:**

- Заняття (Events)**:** Множина занять, які необхідно розподілити.
- Часові слоти (Timeslots)**:** Доступні періоди, протягом яких можуть проводитися заняття.
- Ресурси (Resources)**:** Викладачі, аудиторії та обладнання.
- Обмеження (Constraints)**:** Правила, яких необхідно дотримуватися при складанні розкладу.

Типи обмежень**:**

- Жорсткі обмеження (Hard Constraints)**:** Обов'язкові до виконання; порушення цих обмежень робить розклад неприйнятним.
  - Приклад: Викладач не може вести два заняття одночасно.

- М'які обмеження (Soft Constraints)**:** Бажані до виконання; порушення цих обмежень не робить розклад неприйнятним, але знижує його якість.
  - Приклад: Уникнення ранніх або пізніх занять для певних груп студентів.

Математична модель**:**

Задачу можна формалізувати за допомогою цілочисельного лінійного програмування або як задачу задоволення обмежень (Constraint Satisfaction Problem) [6].

Нехай:

- $E = \{e_1, e_2, \dots, e_n\}$ – множина занять;
- $T = \{t_1, t_2, \dots, t_m\}$ – множина часових слотів;
- $R = \{r_1, r_2, \dots, r_k\}$ – множина ресурсів.

Потрібно знайти відображення $f: E \to T \times R$, таке, що всі жорсткі обмеження виконуються, а функція вартості, що відображає порушення м'яких обмежень, мінімізується.

Цільова функція полягає у мінімізації сумарного штрафу за порушення м'яких обмежень:

$$\text{Minimize} \sum_{e \in E} \sum_{c \in C_S} \omega_c * v_c(e), \tag{1.1}$$

де:

- C_S — множина м'яких обмежень,
- ω_c — ваговий коефіцієнт для обмеження c,
- $v_c(e)$ — функція, що визначає порушення обмеження c для заняття e.

Задача є NP-складною, що означає неможливість знайти оптимальне рішення за поліноміальний час для великих розмірів задачі [7]. Тому використовуються наближені та евристичні методи для знаходження прийнятних рішень за розумний час.

1.2 Теоретико-методичні підходи до вирішення задачі

Математичні моделі розкладу

Складання розкладу занять є складною комбінаторною задачею, яку можна формалізувати за допомогою різних математичних моделей. Основні підходи до математичного моделювання задачі включають:

1) Цілочисельне лінійне програмування (ЦЛП)

Цей підхід формулює задачу як набір лінійних рівнянь та нерівностей з цілочисельними змінними. ЦЛП дозволяє враховувати жорсткі обмеження та знаходити оптимальні рішення для малих або середніх за розміром задач [8]. Формулювання включає:

- Змінні: Бінарні змінні, що вказують, чи призначено заняття в певний часовий слот та аудиторію.
- Цільова функція: Мінімізація порушень м'яких обмежень або оптимізація певного критерію, наприклад, рівномірності навантаження.
- Обмеження: Викладачі, аудиторії та студенти не можуть бути зайняті в більш ніж одному місці одночасно.

2) Задача задоволення обмежень (Constraint Satisfaction Problem, CSP)

CSP моделює задачу як множину змінних з відповідними доменами значень та набором обмежень [9]. Мета полягає у знаходженні такого присвоєння значень змінним, яке задовольняє всі обмеження. Переваги цього підходу включають гнучкість у вираженні складних обмежень та можливість використання ефективних алгоритмів пошуку.

3) Графові моделі

У графових моделях вершини представляють заняття, а ребра — конфлікти між ними. Задача зводиться до розфарбування графа, де кольори відповідають часовим слотам, і жодні суміжні вершини не повинні мати однаковий колір [10]. Це класична задача розфарбування графа, яка є NP-складною, але для якої існують ефективні апроксимаційні алгоритми.

4) Гіперграфові моделі

Гіперграфи розширюють поняття графів, дозволяючи ребрам (гіперребрам) з'єднувати більше двох вершин. Це корисно для моделювання складніших взаємозв'язків, наприклад, коли кілька занять мають спільні ресурси [11].

5) Моделі з м'якими та жорсткими обмеженнями

Розрізнення між жорсткими (обов'язковими) та м'якими (бажаними) обмеженнями дозволяє більш точно моделювати реальні вимоги до розкладу. М'які обмеження можуть бути включені в цільову функцію з ваговими коефіцієнтами, що відображають їх відносну важливість.

Алгоритми та методи оптимізації

Після формалізації задачі необхідно вибрати відповідний алгоритм для її розв'язання. Основні категорії методів включають:

1) Комбінаторні методи:

- Методи гілок і меж (Branch and Bound)**:** Ці методи систематично перебирають можливі рішення, відсікаючи гілки, які не можуть привести до оптимального рішення [12]. Вони гарантують знаходження оптимального рішення, але обчислювальна складність швидко зростає з розміром задачі.
- Методи цілочисельного лінійного програмування**:** Використовуються спеціалізовані алгоритми, такі як розгалуження і зв'язування, для розв'язання ЦЛП.

2) Евристичні методи

- Жадібні алгоритми: Швидко будують розклад, приймаючи локально оптимальні рішення на кожному кроці [13]. Вони не гарантують глобальної оптимальності, але можуть давати прийнятні результати за короткий час.

- Правила пріоритетів: Встановлюють порядок розміщення занять на основі певних критеріїв, наприклад, занять з найбільшою кількістю конфліктів.

3) Метаевристичні методи

- Генетичні алгоритми (Genetic Algorithms): Імітують процес природної еволюції, використовуючи операції схрещування та мутації для генерації нових рішень [14]. Вони добре працюють для великих та складних задач.
- Алгоритм мурашиної колонії (Ant Colony Optimization): Імітує поведінку мурах (рис. 1.1) у пошуку найкоротших шляхів, використовуючи механізм феромонів для керування пошуком [15].

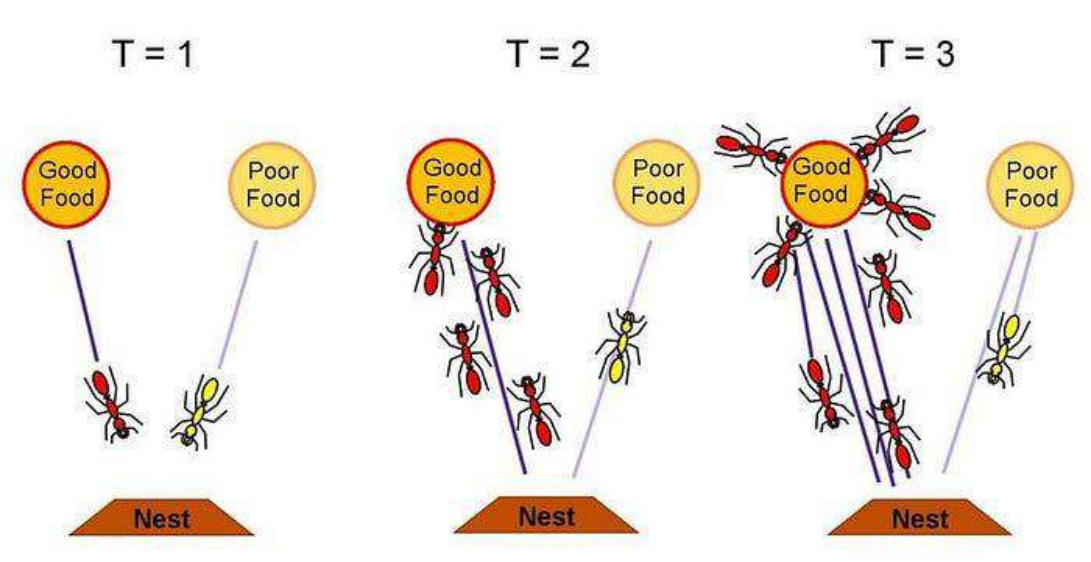


Рисунок 1.1 - Алгоритм мурашиної колонії

- Табу-пошук (Tabu Search): Використовує пам'ять для зберігання заборонених рухів, щоб уникнути циклічних шляхів у просторі пошуку [16].

4) Комбіновані та гібридні методи

- Поєднують різні алгоритми для підвищення ефективності, наприклад, генетичні алгоритми з локальним пошуком або симульованим відпалом.

1.3 Аналіз існуючих механізмів та підходів

Існує ряд програмних засобів та систем, розроблених для автоматизації процесу складання розкладу занять у закладах вищої освіти. Розглянемо найбільш відомі з них.

1) UniTime

UniTime є комплексною, веб-орієнтованою системою з відкритим кодом для складання розкладу (Рис. 1.2), планування аудиторій та управління навчальним процесом [17]. Вона підтримує різні аспекти планування:

- Розклад занять: Враховує потреби студентів, викладачів та ресурсів.
- Екзаменаційні розклади: Допомагає оптимізувати проведення іспитів.
- Запис студентів на курси: Підтримує процес реєстрації на курси та формування груп.

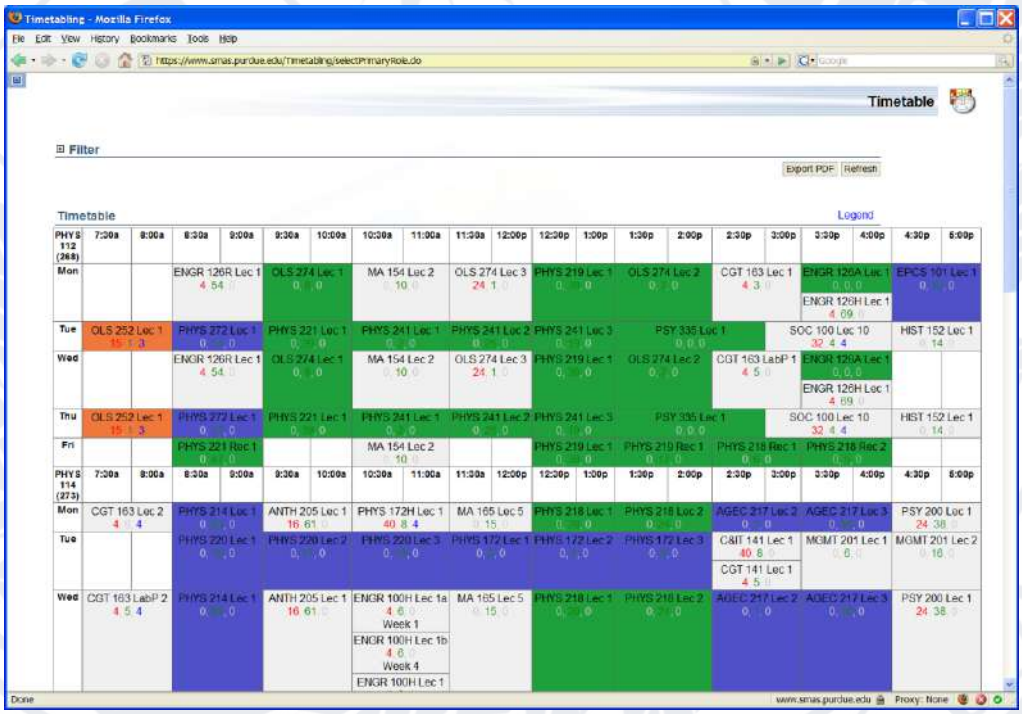


Рисунок 1.2 – інтерфейс UniTime

Особливості UniTime**:**

- Гнучкість: Підтримує різні типи навчальних закладів та структур навчальних планів.

- Алгоритми оптимізації: Використовує комбінацію евристичних та метаевристичних методів.
- Інтеграція: Може інтегруватися з іншими інформаційними системами університету.

2) FET (Free Timetabling Software)

FET — це безкоштовне програмне забезпечення з відкритим кодом для автоматизованого складання розкладу [18]. Воно використовує швидкий алгоритм генерації розкладів, що базується на обмеженнях (Рис. 1.2).

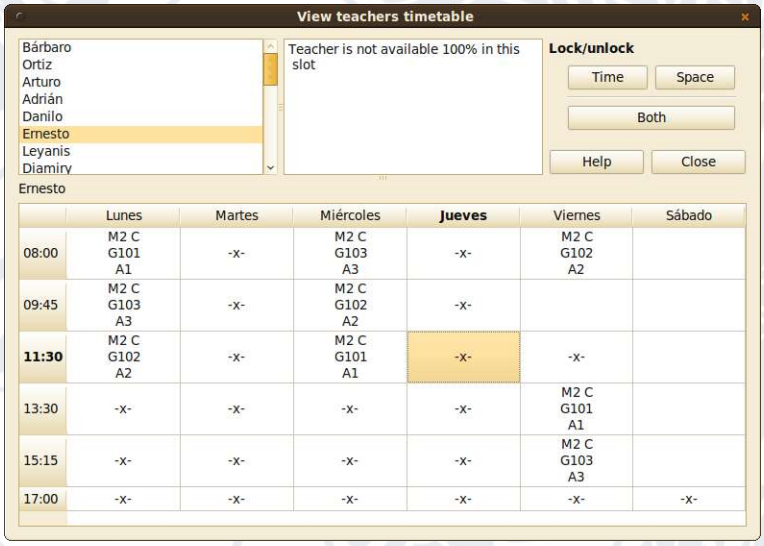


Рисунок 1.3 – інтерфейс FET

Особливості FET**:**

- Швидкість**:** Може генерувати розклад за короткий час навіть для великих задач.
- Налаштування обмежень: Підтримує широкий спектр жорстких та м'яких обмежень.
- Простота використання: Інтуїтивний інтерфейс користувача.

3) ASC Timetables

ASC Timetables — комерційне програмне забезпечення для складання розкладу занять у школах та університетах [19]. Воно поєднує автоматизовані алгоритми з можливістю ручного коригування (Рис. 1.4).

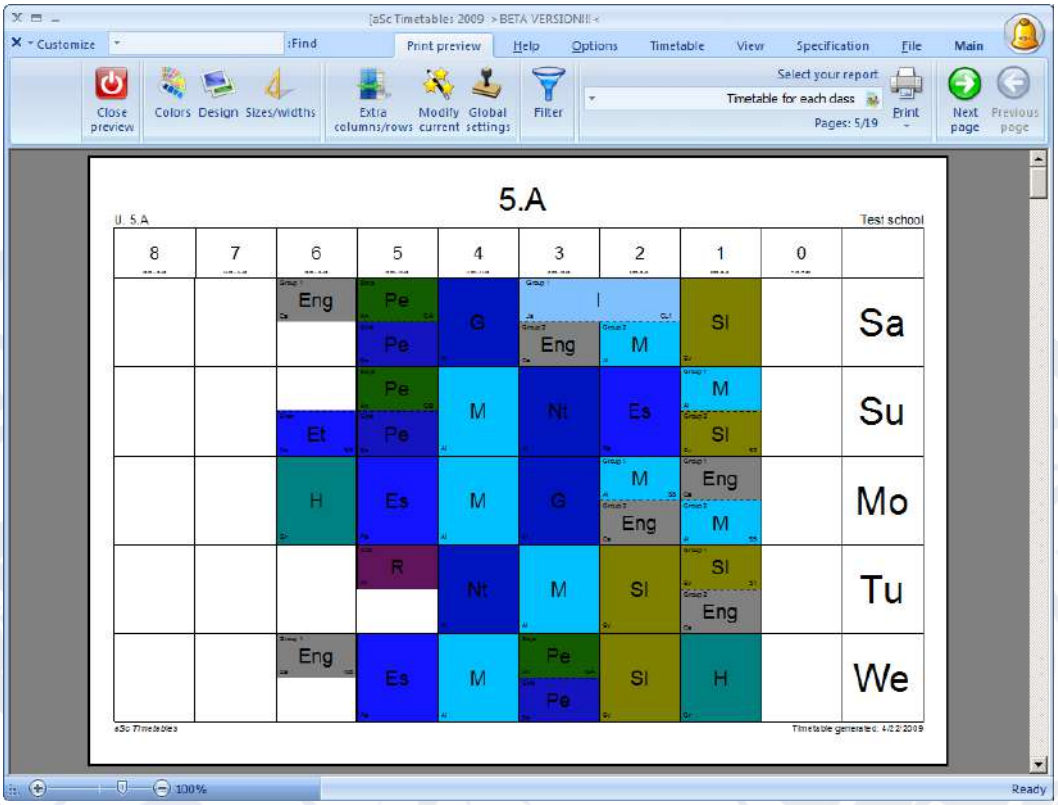


Рисунок 1.4 – Інтерфейс ASC Timetables

Особливості ASC Timetables:

- Інтуїтивний інтерфейс: Дозволяє легко вносити зміни та налаштовувати розклад.
- Автоматизація: Швидко генерує розклад з урахуванням обмежень.
- Мобільні додатки: Підтримує доступ до розкладу через мобільні пристрої.

Порівняльний аналіз методів та алгоритмів

Для оцінки існуючих систем та підходів проведемо порівняльний аналіз за такими критеріями:

- Алгоритмічна основа
- Гнучкість у налаштуванні обмежень
- Простота використання
- Швидкість генерації розкладу
- Можливості інтеграції

1) Алгоритмічна основа

- UniTime: Використовує комбінацію цілочисельного програмування та евристичних методів.
- FET: Застосовує власний алгоритм на основі CSP (Constraint Satisfaction Problem), оптимізований для швидкості.
- ASC Timetables: Використовує власні запатентовані алгоритми, деталі яких не розкриваються.

2) Гнучкість у налаштуванні обмежень

- UniTime: Підтримує широкий спектр обмежень, як жорстких, так і м'яких, з можливістю налаштування вагових коефіцієнтів.
- FET: Надає користувачу можливість встановлювати різні типи обмежень, але може бути складним у налаштуванні специфічних вимог.
- ASC Timetables: Пропонує інтуїтивні засоби для налаштування обмежень, підходить для користувачів без глибоких технічних знань.

3) Простота використання

- UniTime: Вимагає певних технічних знань для встановлення та налаштування.
- FET: Має простий інтерфейс, але налаштування складних обмежень може бути не інтуїтивним.
- ASC Timetables: Орієнтований на кінцевого користувача з дружнім інтерфейсом.

4) Швидкість генерації розкладу

- UniTime: Генерація може займати значний час залежно від розміру задачі та обмежень.
- FET: Відзначається високою швидкістю генерації навіть для великих розкладів.
- ASC Timetables: Швидкість генерації оптимізована для середніх за розміром задач.

5) Можливості інтеграції

- UniTime: Має API та можливості для інтеграції з іншими системами університету.
- FET: Обмежені можливості інтеграції, вимагає додаткової розробки.
- ASC Timetables: Підтримує експорт та імпорт даних у різних форматах.

Таблиця 1.1. Порівняльний аналіз програмних засобів

Критерій	**UniTime**	**FET**	**ASC Timetables**
Алгоритмічна основа	ЦЛП, евристики	CSP	Запатентовані алгоритми
Гнучкість обмежень	Висока	Середня	Висока
Простота використання	Середня	Середня	Висока
Швидкість генерації	Низька	Висока	Середня
Можливості інтеграції	Високі	Низькі	Середні

Проведений аналіз показав, що існуючі системи для автоматизації складання розкладу мають як переваги, так і недоліки:

- UniTime є потужним інструментом з широкими можливостями, але вимагає значних ресурсів для налаштування та експлуатації.
- FET відзначається швидкістю та простотою, але може бути обмеженим у налаштуванні специфічних обмежень та інтеграції.
- ASC Timetables пропонує зручність використання, але як комерційне рішення може бути недоступним для деяких закладів через вартість ліцензії.

З урахуванням специфіки діяльності структурного підрозділу та особливостей існуючих систем, виникає потреба у розробці власного рішення, яке б відповідало таким вимогам:

- Адаптивність: Можливість налаштування під специфічні обмеження та вимоги кафедри.

- Інтеграція: Легке поєднання з існуючими інформаційними системами університету.
- Ефективність: Швидка генерація розкладу з врахуванням великої кількості обмежень.
- Простота використання: Інтуїтивний інтерфейс для користувачів без глибоких технічних знань.
- Відкритість: Використання відкритих технологій та стандартів для забезпечення гнучкості та можливості подальшого розвитку.

Очікувані переваги власного рішення:

- Відповідність специфічним потребам: Система буде розроблена з урахуванням особливостей кафедри та університету.
- Гнучкість та масштабованість: Можливість адаптувати систему до змін та розширювати її функціонал.
- Економічна ефективність: Використання відкритих технологій знижує витрати на розробку та підтримку.
- Підтримка користувачів: Можливість надання документації та навчальних матеріалів для користувачів системи.

1.4 Наукові погляди та категорії дослідження

Для успішного проведення дослідження та розробки ефективної системи автоматизації складання розкладу необхідно визначити та зрозуміти ключові наукові концепції та терміни, які будуть використовуватися.

Автоматизоване складання розкладу (Automated Timetabling) — це процес використання комп'ютерних алгоритмів та програмного забезпечення для створення розкладу занять, який відповідає заданим обмеженням та вимогам [20]. Автоматизація дозволяє зменшити людський фактор, підвищити точність та швидкість складання розкладу.

Комбінаторна оптимізація — це галузь математики та інформатики, яка займається пошуком оптимальних або близьких до оптимальних рішень у

дискретних структурах [21]. Задача складання розкладу є класичною проблемою комбінаторної оптимізації.

NP-складні задачі — це клас задач, для яких не відомі алгоритми, що можуть розв'язати їх за поліноміальний час [22]. Задача складання розкладу належить до цього класу, що ускладнює пошук оптимального рішення.

Метаевристичні алгоритми — це алгоритми, що використовуються для пошуку прийнятних рішень у складних оптимізаційних задачах, де комбінаторні методи є неприйнятними через високу обчислювальну складність [23]. Приклади: генетичні алгоритми, алгоритм імітації відпалу, табу-пошук.

Системи управління базами даних (СУБД) — це програмне забезпечення, що дозволяє створювати, підтримувати та керувати базами даних [24]. В контексті розробки системи для складання розкладу, СУБД використовується для зберігання інформації про курси, викладачів, студентів та ресурси.

Об'єктно-орієнтоване програмування (ООП) — це парадигма програмування, що базується на концепції "об'єктів", які містять дані та методи для роботи з цими даними [25]. Використання ООП сприяє модульності, повторному використанню коду та легкості супроводу програмного забезпечення.

Інтеграція систем — це процес об'єднання різних інформаційних систем та програмних компонентів в єдину узгоджену систему [26]. Інтеграція дозволяє забезпечити обмін даними між системами та підвищити ефективність їх використання.

Системний підхід передбачає розгляд об'єкта дослідження як системи, що складається з взаємопов'язаних елементів [27]. Це дозволяє аналізувати не тільки окремі компоненти, але й їх взаємодію та вплив на загальну функціональність системи.

Емпіричні методи передбачають збір та аналіз даних шляхом спостереження або експерименту [28]. У контексті дослідження це включає тестування розробленої системи на реальних даних кафедри та оцінку її ефективності.

Алгоритмічний підхід зосереджується на розробці алгоритмів для розв'язання конкретних задач [29]. Це ключовий аспект дослідження, оскільки ефективність системи багато в чому залежить від обраних алгоритмів оптимізації.

Категорії та параметри дослідження

Основні категорії:

1) Заняття (Courses/Lectures): Окремі навчальні заходи, які необхідно розмістити у розкладі.
2) Викладачі (Teachers/Instructors): Персонал, який проводить заняття, з їх розкладом та обмеженнями.
3) Студентські групи (Student Groups): Групи студентів, які відвідують певні заняття.
4) Аудиторії (Rooms): Фізичні місця проведення занять з їх характеристиками (місткість, обладнання).
5) Часові слоти (Timeslots): Доступні періоди часу, протягом яких можуть проводитися заняття.

Параметри дослідження:

1) Кількість занять: Загальна кількість занять, які необхідно розмістити.
2) Кількість викладачів: Число викладачів, що беруть участь у навчальному процесі.
3) Обмеження: Жорсткі та м'які обмеження, що застосовуються у процесі складання розкладу.
4) Критерії якості розкладу: Показники, за якими оцінюється якість отриманого розкладу (наприклад, кількість порушень м'яких обмежень).
5) Час генерації розкладу: Час, необхідний системі для створення повного розкладу.

Змінні дослідження:

- Залежні змінні:

1) Якість розкладу (кількість порушень обмежень).
2) Задоволеність користувачів (оцінюється через опитування).
3) Продуктивність системи (час генерації розкладу).

- Незалежні змінні:

1) Обраний алгоритм оптимізації.
2) Налаштування параметрів алгоритму (наприклад, розмір популяції в генетичному алгоритмі).
3) Кількість та складність обмежень.

Метрики оцінки:

1) Кількість порушень жорстких обмежень: Повинна бути нульовою для прийнятного розкладу.
2) Сума штрафів за порушення м'яких обмежень: Використовується для порівняння якості різних розкладів.
3) Час обчислення: Важливий для оцінки практичної застосовності системи.
4) Кількість ітерацій алгоритму: Може використовуватися для налаштування ефективності алгоритму.

Інструменти та технології:

1) Мови програмування: Наприклад, Java, Python або C#, що підтримують ООП.
2) СУБД: MySQL, PostgreSQL або інші системи для зберігання даних.
3) Фреймворки та бібліотеки: Для реалізації веб-інтерфейсу та алгоритмів оптимізації.
4) Середовища розробки: Інтегровані середовища для зручності розробки та тестування.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ МЕТОДИКИ ТА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ДЛЯ АВТОМАТИЗОВАНОГО СКЛАДАННЯ РОЗКЛАДУ

2.1 Формування наукових гіпотез щодо вирішення задачі

У сучасних закладах вищої освіти процес складання розкладу занять є складним та трудомістким завданням, яке вимагає врахування великої кількості змінних та обмежень. Зокрема, необхідно координувати дисципліни, викладачів, аудиторії, групи студентів, а також враховувати специфічні вимоги до обладнання аудиторій, розклад викладачів та особливості навчальних планів.

Особливістю поставленої задачі є необхідність розробки застосунку для автоматичного формування розкладу занять для всіх спеціальностей кафедри на семестр з врахуванням наступних умов:

Вхідні дані**:**

1) Дисципліни: назва, викладачі, поділ на лекції та практичні/лабораторні, кількість занять, вимоги до обладнання.
2) Викладачі: ПІБ, посада, графік роботи, дисципліни, які викладають.
3) Аудиторії: номер, місткість, прив'язка до спеціальностей, наявність обладнання.
4) Групи студентів: спеціальність, курс, назва групи, кількість студентів, дисципліни, поділ на підгрупи, винятки днів.
5) Графік семестру: початок і закінчення навчального семестру.

Обмеження та умови складання розкладу**:**

1) Постійність аудиторій для занять протягом семестру.
2) Обмеження на кількість "вікон" у студентів.
3) Рівномірний розподіл занять протягом семестру.
4) Лекції передують практичним/лабораторним заняттям.
5) Уникнення повторення однакових занять в один день.
6) Обробка виключних ситуацій (неможливість скласти розклад).

Вимоги до функціоналу системи:

1) Введення та зберігання даних: забезпечити можливість введення всіх необхідних вхідних даних та їх зберігання в базі даних.
2) Автоматичне складання розкладу: реалізувати алгоритм, який на основі вхідних даних та обмежень генерує оптимальний розклад.
3) Інтерфейс користувача: надати інтуїтивний графічний інтерфейс для взаємодії з системою.
4) Експорт розкладу: забезпечити можливість експорту розкладу у різних форматах (таблиця, PDF, текстовий файл).
5) Обробка помилок: реалізувати механізми повідомлення про неможливість складання розкладу з вказанням причин.

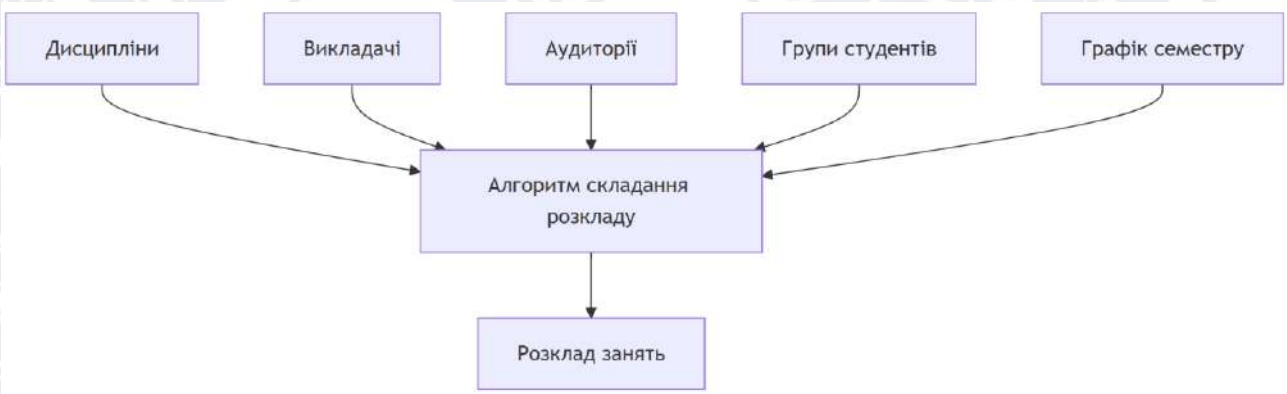


Рисунок 2.1 — Схематичне представлення взаємодії основних компонентів системи

Аналіз можливих підходів до вирішення задачі

Жадібний алгоритм (Greedy Algorithm) приймає локально оптимальні рішення на кожному кроці з надією, що сукупність цих рішень призведе до глобально оптимального результату [30].

Переваги:

- Простота реалізації.
- Швидкість роботи.

Недоліки:

- Не гарантує глобальної оптимальності.
- Може не враховувати складні обмеження.

Метаевристичні алгоритми

- Генетичні алгоритми**:** Імітують процес природної еволюції для пошуку оптимальних рішень [31].
- Алгоритм імітації відпалу: Базується на імітації процесу охолодження металів [32].

Переваги:

- Здатність знаходити близькі до оптимальних рішень**.**
- Гнучкість у врахуванні складних обмежень**.**

Недоліки:

- Висока обчислювальна складність**.**
- Складність налаштування параметрів алгоритму**.**

Екзактні методи

- Цілочисельне лінійне програмування: забезпечує точні рішення [33]. Недолік - непридатність для великих задач через обчислювальну складність**.**

Комбіновані підходи

Поєднання жадібних та метаевристичних алгоритмів для досягнення балансу між швидкістю та якістю рішень.

Таблиця 2.1 - Порівняльний аналіз підходів

Критерій	**Жадібний алгоритм**	**Метаевристичні алгоритми**	**Екзактні методи**
Простота реалізації	Висока	Середня	Низька
Швидкість роботи	Висока	Низька	Низька
Якість рішень	Низька/Середня	Висока	Висока
Обчислювальна складність	Низька	Висока	Висока

Продовження таблиці 2.1

Врахування складних обмежень	Обмежене	Широке	Широке

Вибір оптимального підходу та обґрунтування

З огляду на специфіку задачі та вимоги до системи, було вирішено обрати жадібний алгоритм як основний підхід для розробки алгоритму складання розкладу.

Обґрунтування вибору:

1) Простота реалізації та підтримки

Жадібний алгоритм є відносно простим для реалізації та не вимагає складних налаштувань, що важливо в умовах обмежених ресурсів та часу.

2) Висока швидкість роботи

Для практичного застосування системи важливо, щоб генерація розкладу відбувалася швидко. Жадібний алгоритм забезпечує високу швидкість обчислень, що дозволяє оперативно реагувати на зміни та коригувати розклад.

3) Достатня якість рішень

Хоча жадібний алгоритм може не завжди знаходити глобально оптимальні рішення, він здатний генерувати прийнятні розклади, які відповідають більшості вимог .

4) Можливість врахування обмежень

Обраний алгоритм буде доповнено механізмами перевірки обмежень, що дозволить врахувати специфічні вимоги кафедри та забезпечити коректність розкладу.

Рішення щодо використання інших підходів

Метаевристичні алгоритми та екзактні методи відхилені через високу обчислювальну складність та складність реалізації. Вони можуть бути розглянуті для майбутнього вдосконалення системи.

Формулювання наукових гіпотез

На основі проведеного аналізу та обґрунтування вибору методу сформульовано наступні наукові гіпотези:

Гіпотеза 1

Використання жадібного алгоритму для автоматизованого складання розкладу занять дозволить швидко та ефективно генерувати прийнятні розклади, які відповідають більшості встановлених обмежень та вимог.

Гіпотеза 2

Додавання механізмів перевірки та обробки обмежень до жадібного алгоритму підвищить якість розкладу та забезпечить відповідність специфічним потребам кафедри.

Гіпотеза 3

Реалізація системи з використанням Python, Tkinter та MySQL забезпечить достатню продуктивність та зручність у використанні, що сприятиме ефективному впровадженню системи в навчальний процес.

Гіпотеза 4

Система зможе виявляти ситуації, коли складання розкладу неможливе через конфлікти в обмеженнях, та надавати користувачу зрозумілі повідомлення про причини, що підвищить її практичну цінність.

Очікування від перевірки гіпотез

Для гіпотези 1 та 2: Планується провести тестування алгоритму на реальних даних кафедри та оцінити якість згенерованих розкладів за встановленими метриками.

Для гіпотези 3: Оцінити продуктивність та зручність використання системи через відгуки користувачів та вимірювання часу генерації розкладу.

Для гіпотези 4: Перевірити ефективність механізмів виявлення неможливості складання розкладу та якість повідомлень для користувача.

2.2 Формальний опис моделі та системний аналіз процесів

Для успішного вирішення задачі автоматизованого складання розкладу необхідно формалізувати вхідні дані та визначити їх структуру. Основними компонентами системи є:

- Дисципліни
- Викладачі
- Аудиторії
- Групи студентів
- Графік семестру

1. Дисципліни

Кожна дисципліна характеризується наступними параметрами:

- $D = \{d_1, d_2, \ldots, d_n\}$ — множина дисциплін.
- Назва дисципліни: $name_d$.
- Викладачі дисципліни: $Teachers_d \subseteq T$, де T — множина всіх викладачів.
- Типи занять: лекції (L), практичні (P), лабораторні (Lab).
- Кількість занять за семестр: $count_d^L, count_d^P, count_d^{Lab}$.
- Вимоги до аудиторії: наявність проектора $projector_d$, комп'ютерів $computers_d$.

2. Викладачі

Кожен викладач характеризується:

- $T = \{t_1, t_2, \ldots, t_m\}$ — множина викладачів.
- ПІБ: $name_t$.
- Посада та науковий ступінь: $position_t, degree_t$.
- Графік роботи: множина доступних часових слотів $availability_t \subseteq S$, де S — множина всіх часових слотів.
- Дисципліни, які викладає: $Disciplines_t \subseteq D$.
- Типи занять, які проводить: лекції, практичні/лабораторні.

3. Аудиторії

Характеристики аудиторій:

- $R = \{r_1, r_2, \ldots, r_k\}$ — множина аудиторій.
- Номер аудиторії: $number_r$.
- Місткість: $capacity_r$.
- Прив'язка до спеціальності: $specialties_r \subseteq S_p$, де S_p — множина спеціальностей.
- Обладнання:
  - Наявність комп'ютерів $computers_r$ та їх кількість $num\_computers_r$.
  - Наявність проектора $projector_r$.

4. Групи студентів

Характеристики груп студентів:

- $G = \{g_1, g_2, \ldots, g_l\}$ — множина груп.
- Спеціальність: $specialty_g \in S_p$.
- Курс навчання: $course_g$.
- Назва групи: $name_g$.
- Кількість студентів: $num\_students_g$.
- Дисципліни: $Disciplines_g \subseteq D$.
- Поділ на підгрупи: $subgroups_g$.
- Винятки днів: множина днів $exceptions_g \subseteq D_w$, де D_w — дні тижня.

5. Графік семестру

- Початок семестру: $start\_date$.
- Закінчення семестру: end_date.
- Часові слоти: Множина пар на тиждень $S = \{s_1, s_2, \ldots, s_p\}$, де кожен часовий слот визначається днем тижня та номером пари.

Математична модель задачі

Мета моделювання — формалізувати задачу складання розкладу у вигляді математичної моделі, яка дозволить визначити оптимальний розподіл занять з урахуванням заданих обмежень.

Рішення: Задача моделюється як задача призначення, де необхідно знайти відображення:

$$f: Z \to S \times R \tag{2.1}$$

де Z — множина занять, S — множина часових слотів, R — множина аудиторій.

1. Змінні

$$x_{z,s,r} = \begin{cases} 1, \text{якщо заняття } z \text{ проводиться в часовому слоті } s \text{ в аудиторії } r \\ 0, \text{інакше} \end{cases}$$

2. Обмеження

1) Жорсткі обмеження (Hard Constraints):

- Обмеження на викладача:

Для кожного викладача t:

$$\sum_{z \in Z_t} x_{z,s,r} \leq 1, \qquad \forall s \in availability, \tag{2.2}$$

де Z_t — множина занять, які проводить викладач t.

- Обмеження на групу студентів:

Для кожної групи g:

$$\sum_{z \in Z_g} x_{z,s,r} \leq 1, \qquad \forall s \notin exceptioins_g, \tag{2.3}$$

де Z_g — множина занять для групи g.

- Обмеження на аудиторію:

Для кожної аудиторії r:

$$\sum_{z \in Z} x_{z,s,r} \leq 1, \qquad \forall s \in S, \tag{2.4}$$

- Вимоги до обладнання аудиторії:

Якщо заняття z потребує проектора, то $projector_r = True$. Аналогічно для комп'ютерів.

- Місткість аудиторії:

Кількість студентів $num\_students_g$ не повинна перевищувати $capacity_r$.

2) М'які обмеження (Soft Constraints):

- Уникнення "вікон" у розкладі групи:

Кількість "вікон" не більше одного на день для кожної групи.

- Постійність аудиторій:

Для занять одного типу з однієї дисципліни бажано використовувати ту саму аудиторію.

- Розподіл занять протягом семестру:

Заняття рівномірно розподіляються по тижнях семестру.

- Лекції передують практичним/лабораторним:

Лекції повинні проводитися до практичних/лабораторних занять.

3. Цільова функція

Мінімізувати сумарний штраф за порушення м'яких обмежень:

$$\text{Minimize} \sum_{z \in Z} (\omega_1 * violation\_windows(z) + \omega_2 * violation\_room\_consistency(z) + \omega_3 * violation\_lecture\_before\_practice(z)),$$

де ω_i — вагові коефіцієнти, які визначають важливість кожного м'якого обмеження.

Системний аналіз процесу складання розкладу

Процес складання розкладу можна представити як послідовність етапів, де кожен етап взаємодіє з відповідними компонентами системи.

Етапи процесу:

1. Збір та перевірка вхідних даних

– Користувач вводить дані про дисципліни, викладачів, аудиторії, групи студентів та графік семестру.
– Система перевіряє коректність та повноту введених даних.

2. Формування списку занять

– Генерується повний список занять Z на основі кількості занять з кожної дисципліни для кожної групи.

3. Визначення пріоритетів занять

– Заняття сортуються за пріоритетами, наприклад, лекції з великою кількістю студентів можуть мати вищий пріоритет.

4. Призначення занять до часових слотів та аудиторій

– За допомогою жадібного алгоритму здійснюється призначення занять з урахуванням обмежень.

5. Перевірка обмежень

– Після призначення кожного заняття система перевіряє, чи не порушуються жорсткі та м'які обмеження.

6. Обробка виключних ситуацій

– Якщо призначення неможливе, система генерує повідомлення з вказанням причин.

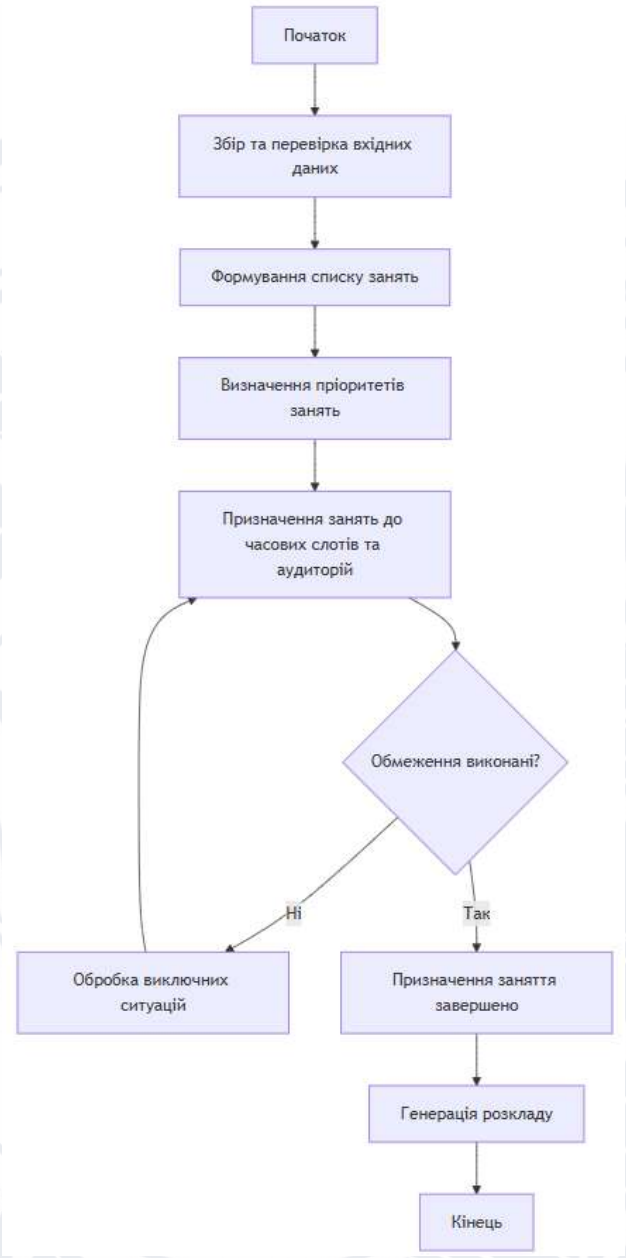


Рисунок 2.2 - Блок-схема процесу складання розкладу

Взаємодія компонентів:

- Дані з бази даних отримуються та використовуються алгоритмом для прийняття рішень.
- Інтерфейс користувача дозволяє вводити та редагувати дані, а також отримувати результати.
- Модуль перевірки обмежень інтегрований в алгоритм та забезпечує коректність розкладу.

Опис алгоритму складання розкладу

Жадібний алгоритм реалізується наступним чином:

1. Ініціалізація

- Отримати всі вхідні дані з бази даних.
- Створити список всіх занять Z.

2. Сортування занять

- Сортувати Z за пріоритетом (наприклад, за кількістю студентів або фіксованими вимогами до обладнання).

3. Призначення занять

Для кожного заняття $z \in Z$:

3.1. Знайти можливі часові слоти S_z та аудиторії R_z, які відповідають обмеженням.

3.2. Вибрати перший доступний часовий слот $s \in S_z$ та аудиторію $r \in R_z$.

3.3. Перевірити жорсткі обмеження:

- Викладач доступний у s.
- Група студентів не має інших занять у s.
- Аудиторія вільна у s.

3.4. Якщо обмеження виконуються, призначити заняття до s та r.

3.5. Якщо ні, перейти до наступного можливого s та r.

3.6. Якщо немає доступних s та r, зафіксувати невдачу для z.

4. Обробка м'яких обмежень

- Після призначення всіх занять, оцінити розклад на предмет порушення м'яких обмежень.
- За можливості, провести локальні зміни для покращення розкладу (наприклад, пересунути заняття для уникнення "вікон").

5. Генерація результатів

- Зберегти отриманий розклад у базі даних.

- Надати користувачу можливість переглянути та експортувати розклад.

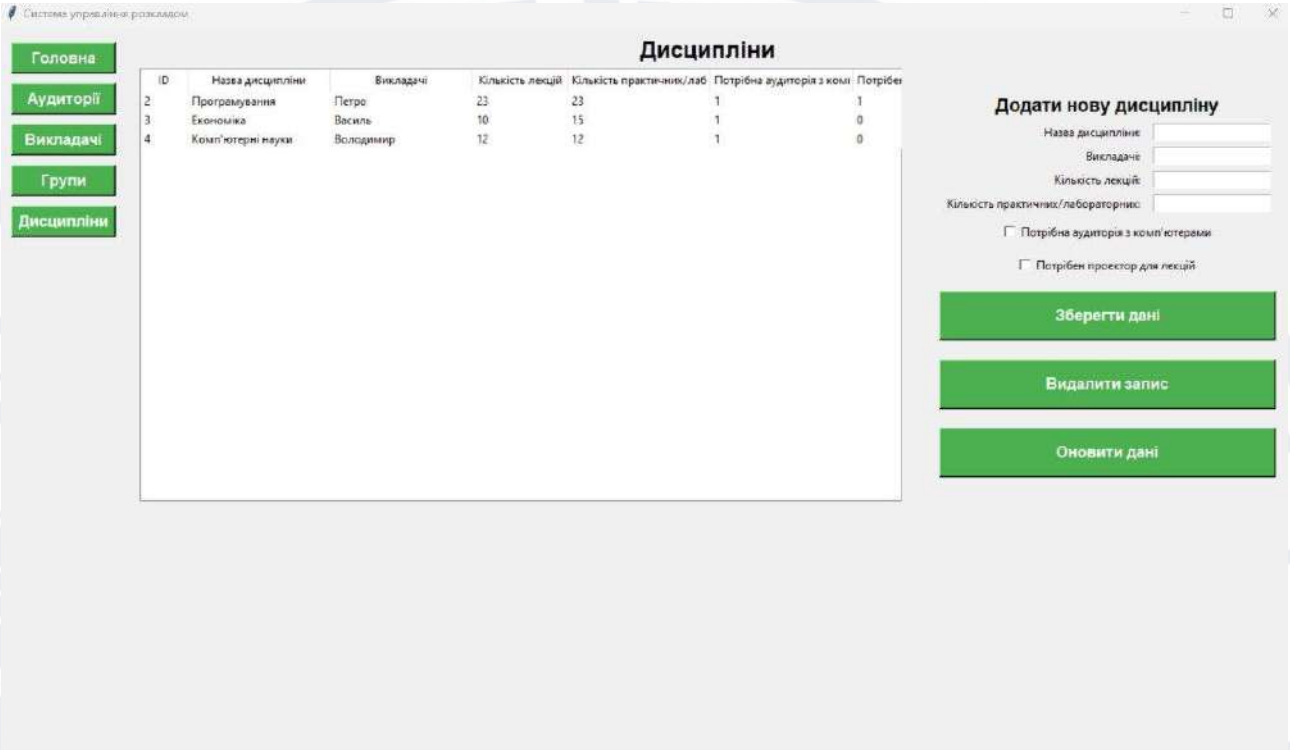


Рисунок 2.3 - Ілюстрація процесу призначення заняття

Обробка виключних ситуацій

Під час призначення занять можуть виникати ситуації, коли для певного заняття неможливо знайти доступний часовий слот та аудиторію, що відповідають усім жорстким обмеженням.

Причини неможливості:

- Конфлікти в розкладі викладачів: Викладач зайнятий в усі доступні для нього часові слоти.
- Конфлікти в розкладі групи: Група має інші заняття в усі можливі часові слоти.
- Відсутність відповідних аудиторій: Немає вільних аудиторій, які відповідають вимогам заняття.
- Недостатня кількість занять для покриття навчального плану: Заняття не встигають провести за час семестру.

Механізм обробки:

- Реєстрація невдачі: Якщо заняття не вдалося призначити, система зберігає інформацію про це з вказанням причин.
- Повідомлення користувача: Після завершення складання розкладу система надає користувачу список занять, які не вдалося призначити, та причини.
- Рекомендації: Система може пропонувати можливі рішення, наприклад:
    - Змінити графік роботи викладача.
    - Переглянути винятки днів для групи.
    - Додати додаткові аудиторії або обладнання.

Процес обробки виключних ситуацій зображено у додатку А.

2.3 Розробка програмної архітектури системи

Для розробки інформаційної системи автоматизованого складання розкладу було обрано такі технології та інструменти:

Мова програмування Python: Відомий своєю простотою та читабельністю, Python має велику кількість бібліотек та фреймворків, що спрощують розробку складних застосунків [33].

Графічний інтерфейс Tkinter: Стандартна бібліотека для створення графічних інтерфейсів у Python, яка не вимагає додаткових встановлень та забезпечує кросплатформенність [34].

Система управління базами даних MySQL: Реляційна СУБД з відкритим кодом, що забезпечує високу продуктивність, надійність та масштабованість [35].

Переваги вибраних технологій:

1. Python

- Простота та швидкість розробки: Синтаксис Python дозволяє швидко розробляти та прототипувати застосунки.
- Багатство бібліотек: Наявність бібліотек для роботи з базами даних (наприклад, `mysql-connector-python`), алгоритмами та іншими необхідними функціями [36].

- Кросплатформенність: Python працює на різних операційних системах, що забезпечує універсальність розробленого застосунку.

2. Tkinter

- Інтеграція з Python: Tkinter є частиною стандартної бібліотеки Python, що спрощує його використання.
- Простота створення інтерфейсу: Забезпечує базовий набір віджетів для створення інтуїтивного графічного інтерфейсу [37].
- Кросплатформенність: Інтерфейси, створені з використанням Tkinter, працюють на різних операційних системах без змін у коді.

3. MySQL

- Продуктивність та масштабованість: Підтримує великі обсяги даних та високі навантаження [38].
- Поширеність та підтримка: Широко використовується в індустрії, має велику спільноту та хорошу документацію.
- Безкоштовність та відкритий код: MySQL є відкритою СУБД, що знижує вартість розробки.

Альтернативні технології та обґрунтування відмови від них

- Мови програмування Java або C#: Хоча вони потужні, їх синтаксис є складнішим, а процес розробки довшим [39].
- Фреймворки для веб-розробки (Django, Flask): Не були обрані, оскільки проект не вимагає веб-інтерфейсу на початковому етапі.
- СУБД PostgreSQL: Хоча є потужною, MySQL було обрано через простоту налаштування та достатність функціональності для поточного проекту.

Архітектура системи

Система має модульну архітектуру, що складається з наступних основних компонентів:

- Інтерфейс користувача (GUI): Забезпечує взаємодію користувача із системою.
- Логіка застосунку: Містить реалізацію алгоритму складання розкладу та обробку даних.
- Доступ до даних: Відповідає за взаємодію з базою даних MySQL.
- База даних: Зберігає всю необхідну інформацію для роботи системи.

Модульність та взаємодія компонентів:

- GUI взаємодіє з логікою застосунку, отримуючи та відправляючи дані.
- Логіка застосунку використовує модуль доступу до даних для зчитування та запису інформації у базу даних.
- Модуль доступу до даних виконує SQL-запити до бази даних та повертає результати логіці застосунку.

Переваги обраної архітектури:

- Модульність: Полегшує підтримку та розширення системи.
- Відокремлення логіки від інтерфейсу: Дозволяє змінювати інтерфейс без впливу на основну логіку застосунку.
- Можливість масштабування: Додаючи нові модулі або функціональність, можна легко адаптувати систему до нових вимог.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Проектування бази даних

База даних складається з наступних основних таблиць (Рис. 3.1):

- discipline: Зберігає інформацію про дисципліни.
  - *id (PK)*
  - *name*
  - *lecture_count*
  - *practice_count*
  - *requires_projector*
  - *requires_computers*
- teacher: Інформація про викладачів.
  - *id (PK)*
  - *full_name*
  - *position*
  - *degree*
  - *availability (часові слоти)*
- auditorium: Дані про аудиторії.
  - *id (PK)*
  - *number*
  - *capacity*
  - *specialties*
  - *has_projector*
  - *computer_count*
- student_group: Інформація про студентські групи.
  - *id (PK)*
  - *specialty*
  - *course*
  - *group_name*

- *student_count*
- *exceptions (винятки днів)*

- schedule: Збережені розклади.
  - *id (PK)*
  - *discipline_id (FK)*
  - *teacher_id (FK)*
  - *auditorium_id (FK)*
  - *group_id (FK)*
  - *timeslot*

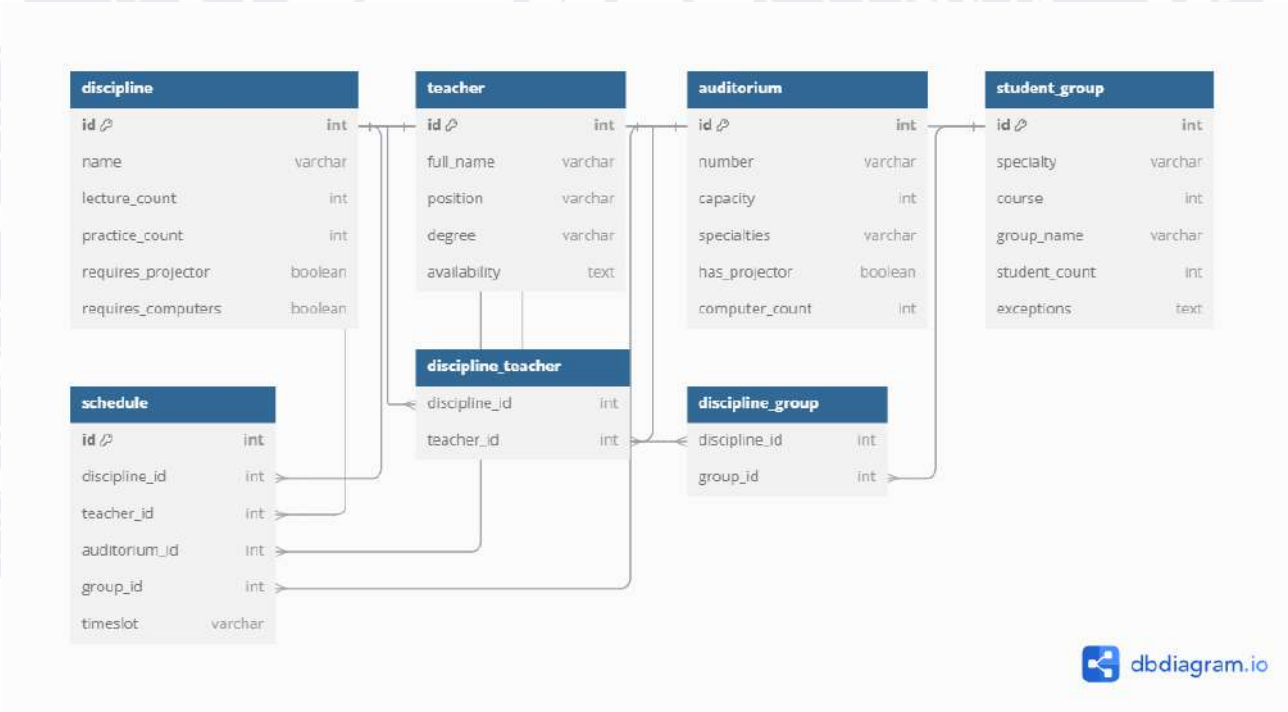


Рисунок 3.1 - ER-діаграма бази даних

Зв'язки між таблицями:

- *discipline* має зв'язок багато-до-багатьох з *teacher* через проміжну таблицю *discipline_teacher*.
- *discipline* пов'язана з *student_group* через таблицю *discipline_group*.
- *schedule* містить зовнішні ключі до основних таблиць, що дозволяє зберігати інформацію про призначені заняття.

База даних спроектована відповідно до третьої нормальної форми (3НФ), що зменшує дублювання даних та забезпечує цілісність інформації.

3.2 Інтерфейс користувача

Опис графічного інтерфейсу

Інтерфейс користувача розроблений з використанням Tkinter та складається з наступних основних вікон:

- Головне меню
  - *Опції*: Дисципліни, Викладачі, Аудиторії, Групи студентів, Скласти розклад.

Рисунок 3.2 - Вікно головного меню інтерфейсу

- Вікно керування дисциплінами
  - *Функціональність*: Додавання, редагування, видалення дисциплін.
  - *Поля вводу*: Назва, викладачі, кількість лекцій та практичних, вимоги до обладнання.

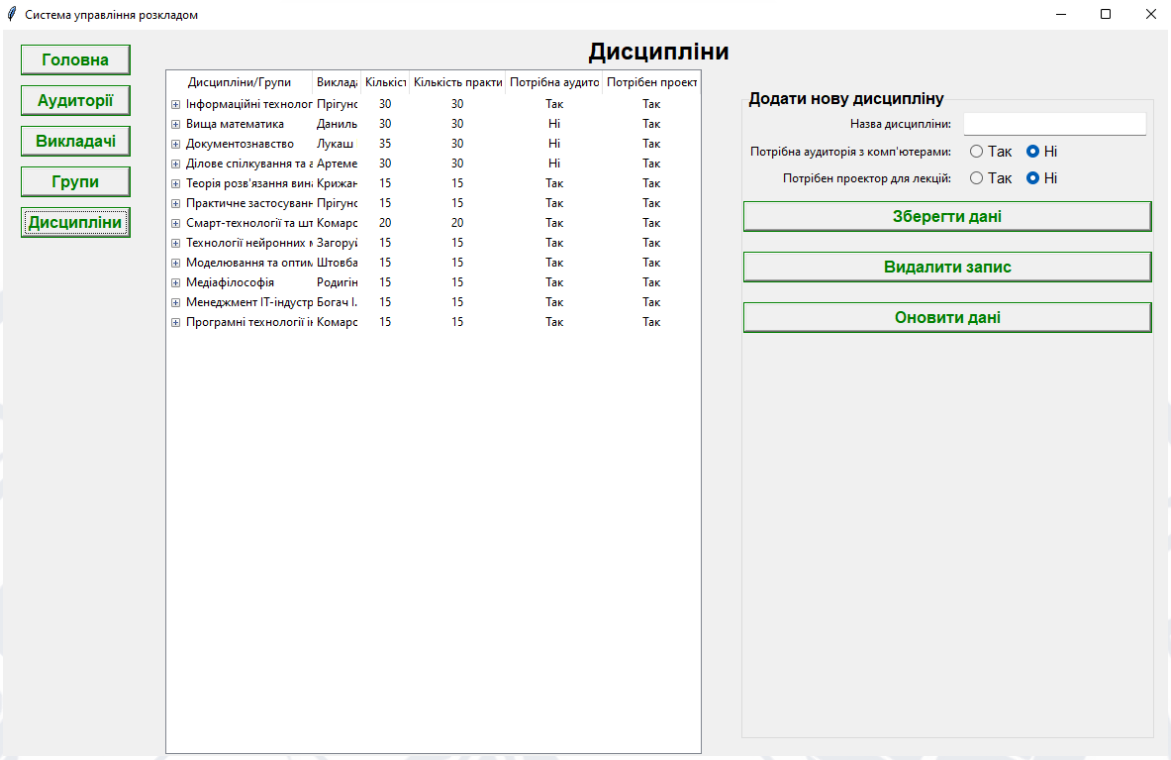


Рисунок 3.3 - Вікно керування дисциплінами

- Вікно керування викладачами
  - *Функціональність*: Додавання, редагування, видалення викладачів.
  - *Поля вводу*: ПІБ, посада, ступінь, графік роботи, дисципліни.

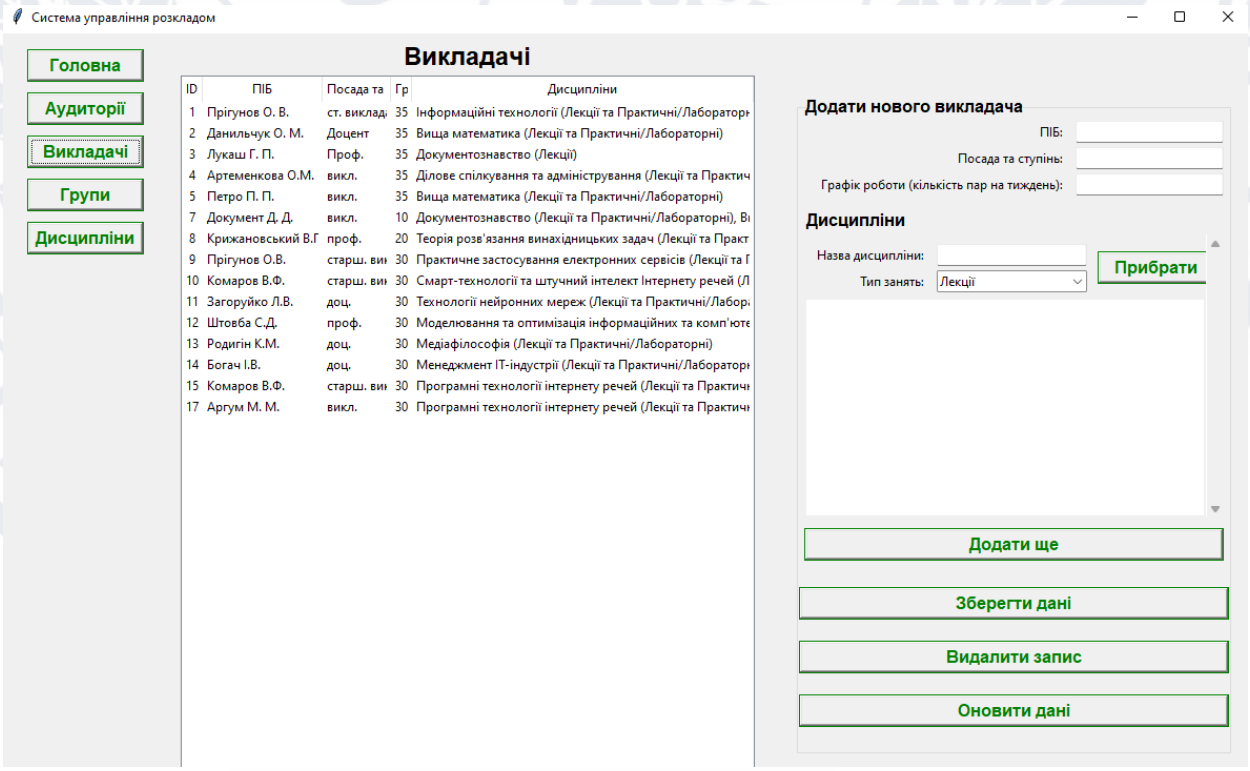


Рисунок 3.4 - Вікно керування викладачами

- Вікно керування аудиторіями
  - *Функціональність*: Додавання, редагування, видалення аудиторій.
  - *Поля вводу*: Номер, місткість, спеціальності, обладнання.

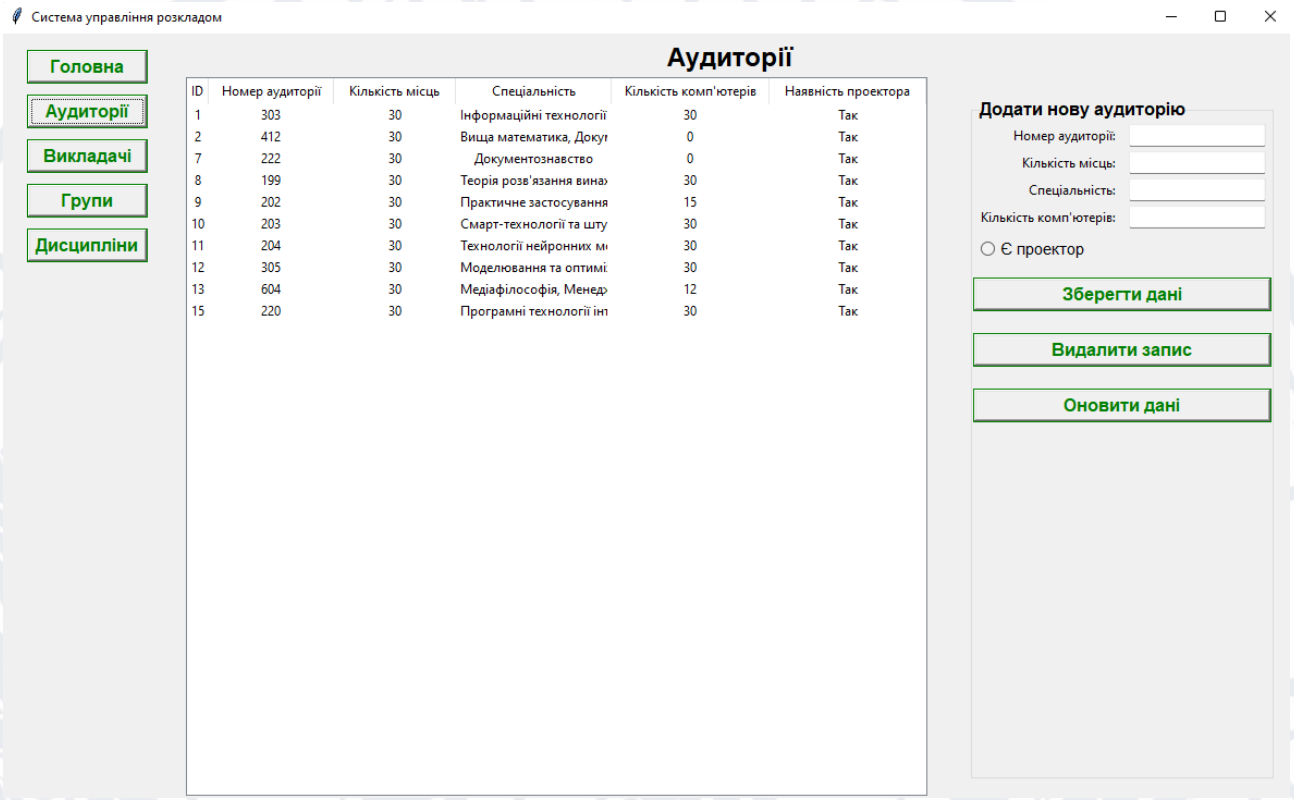


Рисунок 3.5 - Вікно керування аудиторіями

- Вікно керування групами студентів
  - *Функціональність*: Додавання, редагування, видалення груп.
  - *Поля вводу*: Спеціальність, курс, назва групи, кількість студентів, дисципліни, винятки днів.

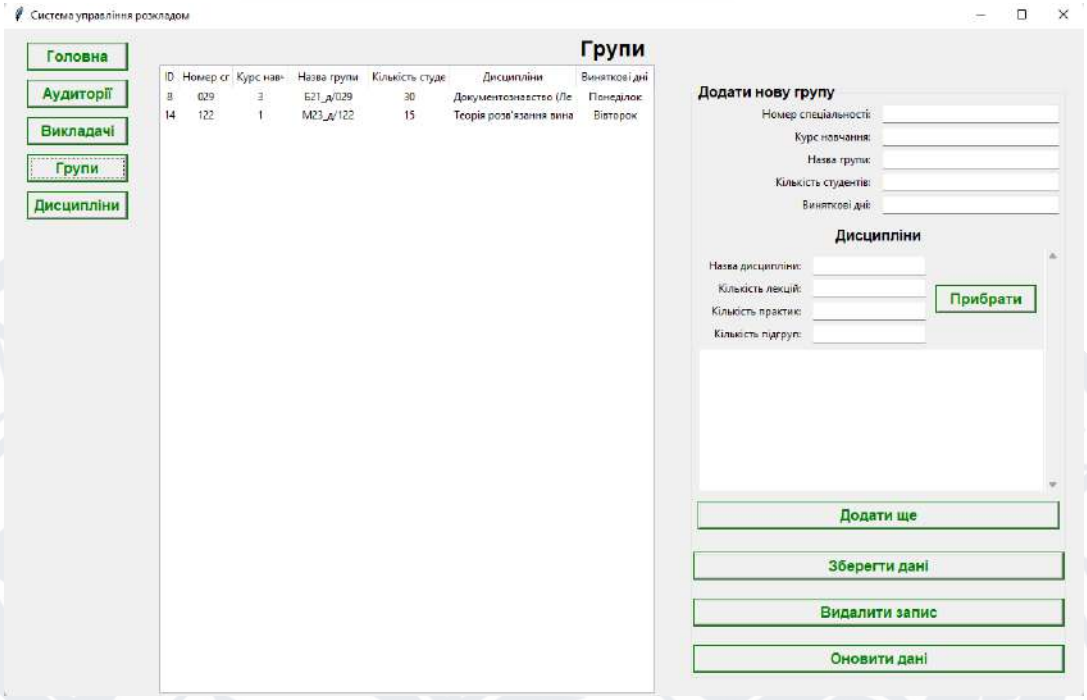


Рисунок 3.6 - Вікно керування групами студентів

- Вікно складання розкладу
  - *Кнопка "Скласти розклад"*: Запускає алгоритм.
  - *Відображення результатів*: Можливість переглянути та експортувати розклад.

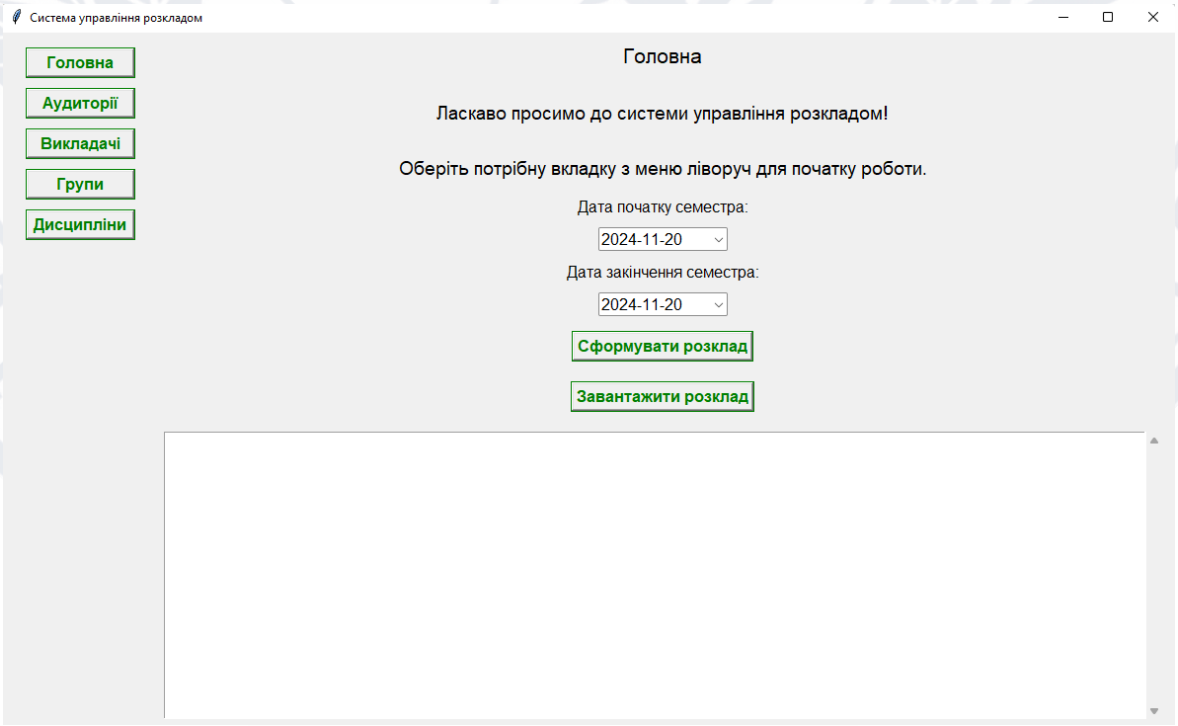


Рисунок 3.7 - Макет головного меню інтерфейсу

Особливості інтерфейсу:

- *Інтуїтивність*: Простий та зрозумілий для користувача інтерфейс.
- *Валідація введення*: Перевірка коректності введених даних з повідомленнями про помилки.
- *Доступність*: Підтримка різних роздільних здатностей екрану.

3.3 Модулі та функції системи

Основні модулі системи:

- Модуль введення та зберігання даних
  - *Функції*: Забезпечує введення даних користувачем та їх зберігання у базі даних.
  - *Класи*: *DisciplineManager, TeacherManager, AuditoriumManager, GroupManager*.
- Модуль алгоритму складання розкладу
  - *Функції*: Реалізує жадібний алгоритм з урахуванням обмежень.
  - *Клас*: *Scheduler*.
  - *Методи*: *generate_schedule(), assign_lesson(), check_constraints()*.
- Модуль перевірки обмежень
  - *Функції*: Перевіряє виконання жорстких та м'яких обмежень.
  - *Методи*: *check_teacher_availability(), check_group_availability(), check_room_availability()*.
- Модуль обробки виключних ситуацій
  - *Функції*: Виявляє та повідомляє про неможливість складання розкладу.
  - *Методи*: *record_failure(), generate_report()*.
- Модуль експорту розкладу
  - *Функції*: Експортує розклад у різні формати.
  - *Методи*: *export_to_excel(), export_to_pdf(), export_to_text()*.

Взаємодія модулів:

- Модуль введення даних надає інформацію модулю алгоритму, який використовує модуль перевірки обмежень для прийняття рішень.
- Модуль обробки виключних ситуацій взаємодіє з модулем алгоритму для обробки помилок.
- Модуль експорту отримує дані від модуля алгоритму після успішного складання розкладу.

Рисунок 3.8 - Діаграма взаємодії модулів

Детальний опис функцій:

- *generate_schedule()*: Основний метод, що запускає процес складання розкладу.
- *assign_lesson()*: Призначає конкретне заняття до часового слоту та аудиторії.
- *check_constraints()*: Перевіряє всі необхідні обмеження перед призначенням заняття.
- *export_to_excel()*: Експортує розклад у формат Excel для подальшого використання.
- *record_failure()*: Фіксує причини неможливості призначити заняття.

3.4 Реалізація алгоритму та програмних компонентів

Розробка алгоритму складання розкладу

На основі розробленої математичної моделі та описаного алгоритму, було реалізовано жадібний алгоритм для складання розкладу занять.

Алгоритм складається з наступних основних кроків:

1. Ініціалізація даних: Завантаження всіх необхідних даних з бази даних.
2. Сортування занять: Формування пріоритетного списку занять для призначення.
3. Призначення занять: Послідовне призначення занять до часових слотів та аудиторій з перевіркою обмежень.
4. Обробка м'яких обмежень: Оптимізація розкладу для мінімізації порушень м'яких обмежень.
5. Збереження результатів: Збереження згенерованого розкладу в базі даних.

Реалізація алгоритму на Python

Було створено клас `Scheduler`, який реалізує основну функціональність алгоритму (Додаток Б).

Пояснення ключових методів:

- `load_data()`: Завантажує всі необхідні дані з бази даних.
- `sort_lessons()`: Сортує заняття за визначеним критерієм пріоритету.
- `assign_lessons()`: Основний цикл призначення занять з перевіркою обмежень.
- `check_availability(lesson, timeslot)`: Перевіряє доступність викладача та групи у даному часовому слоті.
- `check_room(lesson, room, timeslot)`: Перевіряє відповідність аудиторії вимогам заняття та її доступність.
- `update_availability(lesson, timeslot, room)`: Оновлює статуси зайнятості викладача, групи та аудиторії.
- `record_failure(lesson)`: Фіксує невдалі спроби призначення занять.

- `optimize_schedule()`: Оптимізує розклад з урахуванням м'яких обмежень.
- `save_schedule()`: Зберігає згенерований розклад у базу даних.
- `generate_schedule()`: Основний метод, який запускає процес складання розкладу.

Впровадження обмежень та правил

Жорсткі обмеження були реалізовані у вигляді перевірок у методах `check_availability()` та `check_room()` (Лістинг 3.1).

```
python
def check_availability(self, lesson, timeslot):
    # Перевірка доступності викладача
    if timeslot not in lesson.teacher.availability:
        return False
    if self.is_teacher_busy(lesson.teacher, timeslot):
        return False
    # Перевірка доступності групи
    for group in lesson.groups:
        if timeslot in group.exceptions:
            return False
        if self.is_group_busy(group, timeslot):
            return False
    return True

def is_teacher_busy(self, teacher, timeslot):
    for entry in self.schedule:
        if entry['timeslot'] == timeslot and
entry['lesson'].teacher == teacher:
            return True
    return False

def is_group_busy(self, group, timeslot):
    for entry in self.schedule:
        if entry['timeslot'] == timeslot and group in
entry['lesson'].groups:
            return True
    return False
```

Рисунок 3.9. Приклад реалізації перевірки доступності викладача та групи

М'які обмеження були враховані на етапі оптимізації розкладу в методі `optimize_schedule()`. Для цього було реалізовано алгоритм локального

пошуку (Рисунок 3.10), який намагається мінімізувати порушення м'яких обмежень.

```
python
def optimize_schedule(self):
    # Для кожної групи
    for group in self.groups:
        # Отримуємо розклад групи
        group_schedule = [entry for entry in self.schedule if
group in entry['lesson'].groups]
        # Перевіряємо наявність "вікон" у кожен день
        for day in self.get_week_days():
            lessons_in_day = [entry for entry in group_schedule
if entry['timeslot'].day == day]
            lessons_in_day.sort(key=lambda                x:
x['timeslot'].number)
            # Перевірка та усунення "вікон"
            self.remove_windows(lessons_in_day)

python
def remove_windows(self, lessons_in_day):
    for i in range(len(lessons_in_day) - 1):
        current_slot = lessons_in_day[i]['timeslot']
        next_slot = lessons_in_day[i + 1]['timeslot']
        if next_slot.number - current_slot.number > 1:
            # Спробувати пересунути заняття
            self.try_reschedule(lessons_in_day[i + 1])
```

Рисунок 3.10 - Приклад реалізації уникнення "вікон" у розкладі групи

Реалізація інших м'яких обмежень, таких як постійність аудиторій та порядок проведення лекцій перед практичними, здійснювалася аналогічним чином з використанням додаткових перевірок та оптимізацій.

Механізми перевірки та валідації даних

При введенні даних через інтерфейс користувача реалізовано валідацію для забезпечення коректності та повноти інформації (Рисунок 3.11).

```
python
def validate_student_count(self, input_value):
    if not input_value.isdigit():
        messagebox.showerror("Помилка вводу", "Кількість студентів повинна бути числом.")
        return False
    elif int(input_value) <= 0:
        messagebox.showerror("Помилка вводу", "Кількість студентів повинна бути більше нуля.")
        return False
    return True
```

Рисунок 3.11 - Приклад перевірки коректності введення кількості студентів

При завантаженні даних з бази даних здійснюється перевірка наявності всіх необхідних полів та зв'язків. У випадку відсутності даних система генерує відповідне повідомлення.

Якщо виявлено некоректні дані, користувачеві надається можливість виправити помилки перед продовженням роботи.

Обробка помилок та винятків

Під час призначення занять можуть виникати ситуації, коли неможливо призначити заняття через обмеження. Ці ситуації обробляються методом `record_failure()` (Рисунок 3.12).

```
python
def record_failure(self, lesson):
    self.failures.append({
        'lesson': lesson,
        'reason': self.determine_failure_reason(lesson)
    })

def determine_failure_reason(self, lesson):
    reasons = []
    # Перевірка викладача
    if all(not self.is_teacher_available(lesson.teacher, ts) for ts in self.timeslots):
        reasons.append("Викладач недоступний")
    # Перевірка групи
    if all(not self.is_group_available(group, ts) for group in lesson.groups for ts in self.timeslots):
        reasons.append("Група недоступна")
    # Перевірка аудиторій
    if all(not self.is_room_suitable(lesson, room) for room in self.rooms):
```

```
            reasons.append("Відсутня відповідна аудиторія")
        return "; ".join(reasons)
```

Рисунок 3.12 - Приклад реалізації фіксації помилки

Після завершення складання розкладу, якщо є невдалі призначення, система відображає детальний звіт з причинами невдач (Рисунок 3.13).

```
python
def show_failures(self):
    if self.failures:
        message = "Не вдалося призначити наступні заняття:\n"
        for failure in self.failures:
            lesson_name = failure['lesson'].name
            reason = failure['reason']
            message += f"- {lesson_name}: {reason}\n"
        messagebox.showwarning("Непризначені заняття",
message)
```

Рисунок 3.13 - Приклад реалізації звіту з причинами невдач

Збереження та експорт розкладу

Після успішного складання розкладу, метод `save_schedule()` зберігає всі призначення в таблицю `schedule` бази даних (Рисунок 3.14).

```
python
def save_schedule(self):
    for entry in self.schedule:
        self.db.save_schedule_entry(
            lesson_id=entry['lesson'].id,
            teacher_id=entry['lesson'].teacher.id,
            group_ids=[group.id for group in
entry['lesson'].groups],
            room_id=entry['room'].id,
            timeslot_id=entry['timeslot'].id
        )
```

Рисунок 3.14 - Приклад реалізації збереження розкладу

Реалізовано можливість експорту розкладу у формати PDF для подальшого використання та розповсюдження.

За допомогою бібліотеки `ReportLab` створюється PDF-документ з таблицею розкладу (Рисунок 3.15).

```
    python
    from reportlab.lib.pagesizes import letter
    from reportlab.platypus import SimpleDocTemplate, Table,
TableStyle
    from reportlab.lib import colors

    def export_to_pdf(self, filename):
        doc = SimpleDocTemplate(filename, pagesize=letter)
        elements = []

        data = [["Дисципліна", "Викладач", "Групи", "Аудиторія",
"День", "Пара"]]
        for entry in self.schedule:
            data.append([
                entry['lesson'].name,
                entry['lesson'].teacher.full_name,
                ",     ".join([group.name     for     group     in
entry['lesson'].groups]),
                entry['room'].number,
                entry['timeslot'].day,
                entry['timeslot'].number
            ])

        table = Table(data)
        table.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
            ('TEXTCOLOR',(0,0),(-1,0),colors.whitesmoke),
            ('ALIGN',(0,0),(-1,-1),'CENTER'),
            ('GRID', (0,0), (-1,-1), 1, colors.black),
        ]))
        elements.append(table)
        doc.build(elements)
        messagebox.showinfo("Експорт завершено", f"Розклад успішно
експортовано до {filename}")
```

Рисунок 3.15 - Приклад використання ReportLab

Тестове середовище

Для перевірки роботи застосунку та проведення тестування було використано наступне середовище:

- Комп'ютер із наступними характеристиками:
  - Процесор: Intel Core i5-8250U 1.6 ГГц (4 ядра, 8 потоків).
  - Оперативна пам'ять: 8 ГБ DDR4.
  - Накопичувач: SSD 256 ГБ.
  - Операційна система: Windows 10 Pro 64-bit.

Методологія тестування:

- Функціональне тестування: Перевірка всіх функцій застосунку на відповідність вимогам.
- Тестування навантаженням: Перевірка роботи системи при великій кількості даних (наприклад, 1000 занять, 100 викладачів).
- Кросплатформене тестування: Запуск застосунку на різних операційних системах (Windows, Ubuntu) для перевірки сумісності.
- Тестування користувацького інтерфейсу: Оцінка зручності використання та інтуїтивності інтерфейсу.

Результати тестування:

- Функціональність: Усі основні функції працюють відповідно до вимог. Виявлені незначні помилки були виправлені.
- Продуктивність: Система здатна генерувати розклад для великої кількості даних за прийнятний час (менше 1 хвилини для 1000 занять).
- Сумісність: Застосунок успішно працює на Windows та Ubuntu без змін у коді.
- Інтерфейс: Користувачі відзначили зручність та простоту інтерфейсу після короткого ознайомлення.

Інструменти для тестування:

- Unittest: Стандартний модуль Python для написання та виконання модульних тестів.
- PyTest: Альтернативний фреймворк для тестування, який забезпечує розширені можливості.
- MySQL Workbench: Використовувався для моніторингу бази даних та аналізу виконання запитів.

ВИСНОВКИ

У першому розділі було проведено всебічний аналіз теоретико-методичних основ автоматизації складання розкладу занять у закладах вищої освіти. Розглянуто сутність та особливості процесу складання розкладу, визначено основні обмеження та проблеми, пов'язані з традиційними методами планування. Зокрема, було підкреслено складність задачі через множинність обмежень, таких як доступність викладачів, груп студентів, аудиторій та необхідного обладнання. Також було зазначено, що процес складання розкладу є динамічним і часто потребує швидкої адаптації до змінних умов.

Було проаналізовано різні математичні моделі та алгоритми, що використовуються для автоматизації цього процесу. Особливу увагу приділено розмежуванню жорстких і м'яких обмежень, що впливають на якість розкладу. У результаті критичного аналізу існуючих підходів було виявлено переваги та недоліки кожного з них, що дозволило сформулювати наукові гіпотези щодо можливості ефективного використання жадібного алгоритму для вирішення поставленої задачі з урахуванням специфічних потреб кафедри.

Таким чином, у першому розділі було закладено теоретичну основу для подальшої розробки методики та інструментальних засобів автоматизованого складання розкладу, що відповідають сучасним вимогам та спрямовані на підвищення ефективності навчального процесу.

У другому розділі було розроблено методику та створено програмні засоби для автоматизованого складання розкладу занять. Сформовано наукові гіпотези, які передбачали використання жадібного алгоритму з додатковими механізмами перевірки та оптимізації. Було детально описано формальний підхід до моделювання задачі, включаючи формалізацію вхідних даних, математичну модель та системний аналіз процесів.

У третьому розділі розроблено програмну архітектуру системи з використанням Python, Tkinter та MySQL, що забезпечує інтуїтивний інтерфейс користувача та ефективну взаємодію з базою даних. Реалізовано алгоритм

складання розкладу, який враховує жорсткі та м'які обмеження, а також механізми обробки виключних ситуацій. Система надає можливість збереження та експорту розкладу у різних форматах, що підвищує її практичну цінність.

Проведено тестування розробленого програмного забезпечення, яке підтвердило його працездатність та відповідність поставленим вимогам. Система здатна швидко генерувати розклад, враховуючи специфічні обмеження, та надавати користувачу зрозумілі повідомлення у випадку неможливості призначення занять. Таким чином, у другому розділі було успішно реалізовано методику та інструментальні засоби, які можуть бути впроваджені в навчальний процес для підвищення ефективності планування та зменшення навантаження на адміністративний персонал.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Wren A. Scheduling, timetabling and rostering a special relationship? *Practice and Theory of Automated Timetabling*. Eds. E. K. Burke, P. Ross. Berlin: Springer, 1996. P. 46–75.
2. Lewis R. A survey of metaheuristic-based techniques for University Timetabling problems. *OR Spectrum*. 2008. Vol. 30, No. 1. P. 167–190.
3. Carter M. W., Laporte G. Recent developments in practical course timetabling. *Practice and Theory of Automated Timetabling*. Eds. E. K. Burke, P. Ross. Berlin: Springer, 1996. P. 3–19.
4. Daskalaki S., Birbas T., Housos E. An integer programming formulation for a case study in university timetabling. *European Journal of Operational Research*. 2004. Vol. 153, No. 1. P. 117–135.
5. Garey M. R., Johnson D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman, 1979. 340 p.
6. Laporte G., Desroches S., Desrochers M. Computerized scheduling of school teachers // *Interfaces*. 1984. Vol. 14, No. 4. P. 68–74.
7. Even S., Itai A., Shamir A. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*. 1976. Vol. 5, No. 4. P. 691–703.
8. Burke E. K., Kingston J., de Werra D. Applications to timetabling. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.* Ed. J. Leung. Boca Raton: CRC Press, 2004. P. 445–474.
9. Marriott K., Stuckey P. J. *Programming with Constraints: An Introduction*. Cambridge: MIT Press, 1998. — 480 p.
10. Leighton F. T. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*. 1979. Vol. 84, No. 6. P. 489–506.
11. Schwindt C. *Resource Allocation in Project Management*. Berlin: Springer, 2005. 280 p.

12. Lawler E. L., Wood D. E. Branch-and-bound methods: A survey. *Operations Research*. 1966. Vol. 14, No. 4. P. 699–719.
13. Fleurent C., Ferland J. A. Object-oriented implementation of heuristic search methods for graph coloring. *Annals of Operations Research*. 1996. Vol. 63, No. 3. P. 437–461.
14. Paechter B., Cumming A., Luchian H., Petriuc M. O. Two solutions to the general timetable problem using evolutionary methods. *Parallel Problem Solving from Nature. PPSN III* Eds. Y. Davidor, H.-P. Schwefel, R. Männer. Berlin: Springer, 1994. P. 251–256.
15. Socha K., Knowles J., Sampels M. A MAX–MIN ant system for the university course timetabling problem. *Practice and Theory of Automated Timetabling*. Eds. E. K. Burke, P. De Causmaecker. Berlin: Springer, 2002. P. 1–13.
16. Glover F., Laguna M. *Tabu Search*. Boston: Kluwer Academic Publishers, 1997.— 408 p.
17. University Timetabling Application (UniTime) [Електронний ресурс]. Режим доступу: https://www.unitime.org. Дата звернення: 01.10.2024.
18. FET - Free Timetabling Software [Електронний ресурс]. Режим доступу: http://lalescu.ro/liviu/fet/. Дата звернення: 01.10.2024.
19. ASC Timetables [Електронний ресурс]. Режим доступу: https://www.asctimetables.com. Дата звернення: 01.10.2024.
20. Burke E. K., Petrovic S. Recent research directions in automated timetabling. *European Journal of Operational Research*. 2002. Vol. 140, No. 2. P. 266–280.
21. Papadimitriou C. H., Steiglitz K. *Combinatorial Optimization: Algorithms and Complexity*. New York: Dover Publications, 1998. 496 p.
22. Garey M. R., Johnson D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman, 1979. 340 p.

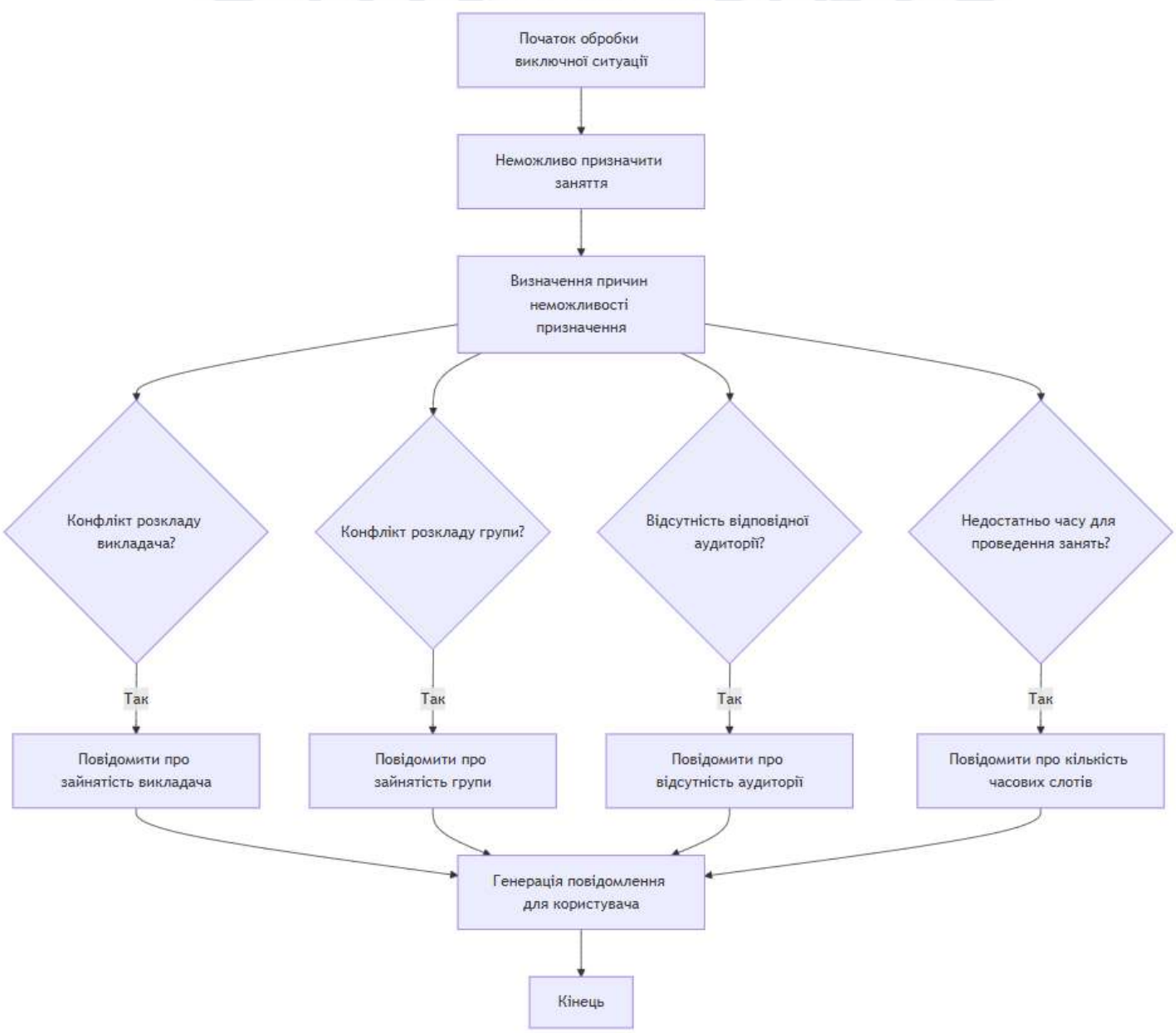
23. Blum C., Roli A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*. 2003. Vol. 35, No. 3. P. 268–308.
24. Petrovic S., Burke E. K. University timetabling. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Ed. J. Leung. Boca Raton: CRC Press, 2004. P. 1001–1028.
25. Date C. J. *An Introduction to Database Systems*. 8th ed. Boston: Addison-Wesley, 2003. 1024 p.
26. Booch G. et al. *Object-Oriented Analysis and Design with Applications*. 3rd ed. Boston: Addison-Wesley, 2007. 720 p.
27. Linthicum D. S. *Enterprise Application Integration*. Boston: Addison-Wesley, 2000. 416 p.
28. Law A. M., Kelton W. D. *Simulation Modeling and Analysis*. 3rd ed. Boston: McGraw-Hill, 2000. 784 p.
29. Creswell J. W. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. 4th ed. Thousand Oaks: SAGE Publications, 2014. 304 p.
30. Cormen T. H. et al. *Introduction to Algorithms*. 3rd ed. Cambridge: MIT Press, 2009. 1312 p.
31. Holland J. H. *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press, 1975. 183 p.
32. Kirkpatrick S., Gelatt C. D., Vecchi M. P. Optimization by simulated annealing. *Science*. 1983. Vol. 220, No. 4598. P. 671–680.
33. Lutz M. *Learning Python*. 5th ed. Sebastopol: O'Reilly Media, 2013. 1648 p.
34. Grayson J. D. *Python and Tkinter Programming*. Greenwich: Manning Publications, 2000. 672 p.
35. DuBois P. *MySQL*. 5th ed. Boston: Addison-Wesley, 2005. 1224 p.
36. Summerfield M. *Programming in Python 3*. 2nd ed. Boston: Addison-Wesley, 2010. 648 p.

37. Shipman J. W. *Tkinter 8.5 reference: a GUI for Python*. Socorro: New Mexico Tech Computer Center, 2010. 178 p.
38. MySQL Documentation [Електронний ресурс]. Режим доступу: https://dev.mysql.com/doc/. Дата звернення: 01.10.2024.
39. Horstmann C. S., Cornell G. *Core Java Volume I–Fundamentals*. 9th ed. — Upper Saddle River: Prentice Hall, 2012. 928 p.

ДОДАТКИ

ДОДАТОК А

Процес обробки виключних ситуацій



ДОДАТОК Б

Клас Scheduler

```
python
class Scheduler:
    def __init__(self, db_connection):
        self.db = db_connection
        self.lessons = []
        self.timeslots = []
        self.rooms = []
        self.schedule = []
        self.failures = []
    def load_data(self):
        # Завантаження даних з бази даних
        self.lessons = self.db.get_lessons()
        self.timeslots = self.db.get_timeslots()
        self.rooms = self.db.get_rooms()
        self.teachers = self.db.get_teachers()
        self.groups = self.db.get_groups()

    def sort_lessons(self):
        # Сортування занять за пріоритетом (наприклад, за кількістю
студентів)
        self.lessons.sort(key=lambda x: x.priority, reverse=True)

    def assign_lessons(self):
        for lesson in self.lessons:
            assigned = False
            for timeslot in self.timeslots:
                if self.check_availability(lesson, timeslot):
                    for room in self.rooms:
                        if self.check_room(lesson, room, timeslot):
                            # Призначення заняття
                            self.schedule.append({
                                'lesson': lesson,
                                'timeslot': timeslot,
                                'room': room
                            })
                            self.update_availability(lesson,     timeslot,
room)
                            assigned = True
                            break
                    if assigned:
                        break
            if not assigned:
                self.record_failure(lesson)

    def optimize_schedule(self):
        # Реалізація оптимізації з урахуванням м'яких обмежень
        pass  # Деталі реалізації будуть розглянуті в підрозділі 2.4.2

    def save_schedule(self):
        # Збереження розкладу в базу даних
        self.db.save_schedule(self.schedule)

    def generate_schedule(self):
        self.load_data()
        self.sort_lessons()
        self.assign_lessons()
        self.optimize_schedule()
        self.save_schedule()
```

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

_____

_____

Повністю вказується ПІБ та статус (освітня (освітньо-наукова) програма – для здобувачів вищої освіти, назва кваліфікаційної роботи)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)