

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Використання генетичного алгоритму для формування
розкладу занять»**

Виконав студент групи ІПЗм-23-1

БАРИБІН Володимир Романович

Керівник роботи:

ВЛАСЕНКО Олег Васильович

Рецензент:

ЛОКТИКОВА Тамара Миколаївна

Житомир - 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК

«18» вересня 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БАРИБІН Володимир Романович**

Керівник роботи: **ВЛАСЕНКО Олег Васильович**

Тема роботи: **«Використання генетичного алгоритму для формування розкладу занять»**, затверджена Наказом закладу вищої освіти від **«18» вересня 2023 р. № 479/с**

Вхідні дані до проекту (роботи): генетичний алгоритм, алгоритм збору розкладу.

Керівники / консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Керівник / консультант	Завдання видав	Завдання прийняв
1	Власенко О.В.	22.09.2023	13.11.2023
2	Власенко О.В.	14.11.2023	22.12.2023
3	Власенко О.В.	08.01.2024	20.05.2024
4	Власенко О.В.	03.09.2024	24.09.2024

РЕФЕРАТ

Кваліфікаційна робота освітнього ступеня «магістр» присвячена використанню генетичного алгоритму для формування розкладу занять. Пояснювальна записка складається з вступу, трьох розділів, списку використаної літератури та додатків. Пояснювальна записка до кваліфікаційної магістерської роботи містить 77 сторінок, 1 таблицю, 11 ілюстрацій, 26 посилань на зовнішні джерела та 7 додатків.

Метою роботи є проектування та розробка системи для формування розкладу занять за допомогою генетичного алгоритму.

У ході роботи було розроблено алгоритм для автоматизованого формування розкладу занять із використанням генетичного підходу. Вдалося досягти оптимізації процесу створення розкладу, що забезпечує мінімізацію конфліктів та підвищення ефективності розподілу ресурсів. Розроблене рішення демонструє високу гнучкість у врахуванні специфічних параметрів навчального процесу.

АЛГОРИТМ, ГЕНЕТИЧНИЙ АЛГОРИТМ, МУТАЦІЇ, СХРЕЩЕННЯ, ОПТИМІЗАЦІЯ, АВТОМАТИЗАЦІЯ, РОЗКЛАД ЗАНЯТЬ.

ABSTRACT

The master's degree qualification work is devoted to the use of a genetic algorithm for the formation of a class schedule. The explanatory note consists of an introduction, three chapters, a list of references, and appendices. The explanatory note to the master's thesis contains 77 pages, 1 table, 11 illustrations, 26 references to external sources, and 7 appendices.

The purpose of the work is to design and develop a system for generating class schedules using a genetic algorithm.

In the course of the work, an algorithm was developed for automated class schedule generation using a genetic approach. We managed to optimize the process of creating the schedule, which minimizes conflicts and increases the efficiency of resource allocation. The developed solution demonstrates high flexibility in taking into account specific parameters of the educational process.

ALGORITHM, GENETIC ALGORITHM, MUTATIONS, CROSSES, OPTIMIZATION, AUTOMATION, CLASS SCHEDULE.

ЗМІСТ

ВСТУП.....	5
РОЗДІЛ 1 ОПТИМІЗАЦІЯ ТА АЛГОРИТМИ ФОРМУВАННЯ НАВЧАЛЬНИХ РОЗКЛАДІВ.....	8
1.1 Існуючі методи формування розкладу.....	8
1.2 Проблеми формування розкладу.....	10
1.3 Технології оптимізації.....	12
1.4 Аналіз вибору алгоритму.....	21
Висновки до розділу 1.....	23
РОЗДІЛ 2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	25
2.1 Постановка задачі.....	25
2.2 Аналіз функціональних вимог.....	26
2.3 Архітектура системи.....	30
2.4 Стек технологій.....	32
Висновки до розділу 2.....	38
РОЗДІЛ 3 РОЗРОБКА ГЕНЕТИЧНОГО АЛГОРИТМУ.....	39
3.1 Опис генетичного алгоритму.....	39
3.2 Реалізація алгоритму.....	44
3.3 Покроковий опис-інструкція.....	48
Висновки до розділу 3.....	52
ВИСНОВКИ.....	54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	56

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних

ГА – генетичні алгоритми

ПЗ – програмне забезпечення

РБД – реляційна база даних

НЗ – навчальний заклад

ЗСР – задача складання розкладів

ВНЗ — вищий навчальний заклад

ВСТУП

Актуальність роботи

Складання навчального розкладу є складним завданням, що вимагає врахування численних факторів, таких як доступність ресурсів, графік роботи викладачів, специфіка потреб студентських груп та обмеження щодо тривалості занять. Традиційний підхід до вирішення цієї проблеми зазвичай базується на ручному плануванні, що часто призводить до надмірних витрат часу та сил, а також не гарантує оптимального розподілу. Натомість використання сучасних алгоритмічних підходів, таких як ГА, дозволяє ефективно вирішувати завдання, забезпечуючи гнучкість і адаптивність у процесі планування.

Отже, впровадження автоматизованих рішень на основі генетичних алгоритмів не лише спрощує процес складання розкладу, а й сприяє оптимізації навчального процесу загалом, забезпечуючи ефективніше використання часу, ресурсів та зручність для всіх учасників навчального процесу.

На відміну від статичних програмних рішень, які часто мають жорстко закладені алгоритми і можуть бути непридатними для більш складних умов, ГА дозволяють знайти збалансовані рішення навіть у випадках з великою кількістю обмежень. Існуючі системи автоматизації, як правило, орієнтовані на вузькі завдання або використовують підходи, що погано масштабуються. Запропонована система відрізняється тим, що враховує як кількісні, так і якісні обмеження, забезпечуючи високу гнучкість та адаптивність.

Одним із рішень яке є максимально наближеним до організації розкладу є сервіс smart school [1], де робота з розкладом — це мануальний процес, також у 2016 році був проведений семінар, який мав на меті презентувати автоматичний генератор розкладу, проте, він випущений так і не був, а сайт компанії не існує [2].

smart school – це програмне забезпечення, яке створене для цифровізації та часткової автоматизації навчального процесу, проте тема розкладу не є автоматизованою [1].

Відмінністю створеного застосунку є використання ГА для формування готового розкладу шляхом оптимізації випадково створеного.

Мета і задачі дослідження

Метою роботи є розробити метод формування навчального розкладу на основі ГА, який забезпечує оптимальне виконання завдань планування з урахуванням обмежень та потреб освітнього процесу.

Розроблена система повинна забезпечувати коректне формування розкладу, з врахуванням накладених обмежень та встановлених умов.

Досягнення мети включало розв'язання таких задач:

- 1) створення серверної частини для отримання запитів та запуску процесу обробки та пошуку;
- 2) обробка усіх записів БД та створення загального сховища з ймовірними записами уроків;
- 3) створення алгоритму для формування випадкового алгоритму;
- 4) реалізація ГА;
- 5) тестування алгоритмів;
- 6) створення клієнтської частини для взаємодії з користувачем.

Об'єктом дослідження є процес формування розкладу навчальних занять у закладах освіти.

Предметом дослідження є методи та алгоритми автоматизації формування розкладу занять, зокрема ГА, їх адаптація та застосування для вирішення задачі оптимізації розкладу.

Методи дослідження

У дослідженні використано методи математичного моделювання, алгоритмізації, програмування, а також емпіричні методи тестування. Основою реалізації є ГА, адаптований до задачі складання розкладу.

Наукова новизна отриманих результатів

Наукова новизна роботи полягає в адаптації ГА для задачі автоматизованого формування розкладу занять з урахуванням множини складних обмежень. Запропоновано унікальну модель представлення даних та метод оцінки придатності рішень, що дозволяє ефективно вирішувати задачу планування навіть за умов складного розподілу ресурсів.

Практичне значення отриманих результатів

Розроблена система може бути застосована в освітніх закладах для автоматизованого формування розкладу занять, що дозволяє значно зменшити витрати часу та ресурсів на процес планування. Результати роботи можуть бути інтегровані у вже існуючі інформаційні системи освітніх установ.

РОЗДІЛ 1

ОПТИМІЗАЦІЯ ТА АЛГОРИТМИ ФОРМУВАННЯ НАВЧАЛЬНИХ РОЗКЛАДІВ

1.1 Існуючі методи формування розкладу

Розвиток науки, інформаційних технологій, цінностей світового суспільства відображається у появі нових винаходів, високотехнологічних продуктів, нових підходів до управління, прийняття рішень загалом до способу розвитку, дослідження, отримання та набування нових знань.

Світова криза, яка набуває масштабів уже декілька років підряд перетворилась у глобальну проблему, яка створила поштовх до пришвидшеного розвитку науки та технологій та сформувала низку викликів людству.

Наразі, задачею українського суспільства є використання створених умов з метою інверсії проблем на можливості. Однією з таких проблем є питання форми та форматів сучасної освіти.

Зокрема дана робота концентрується на питанні автоматизації однієї із складових процесів в організації освіти в навчальних закладах, а саме процесу розробки розкладу. Для того, щоб мати змогу відповісти на питання, чи є можливість автоматизувати даний процес, та як це зробити необхідно розглянути засоби, які можна застосувати для автоматизації та оптимізації даного процесу.

Передусім, хочу зауважити, що не існує загальновідомого алгоритму, який концентрувався на побудові певних структур з нуля, отже побудова первинного розкладу - це задача, яка вимагає розробки, а не адаптації. Попри це, для побудови оптимального розкладу необхідно застосувати шлях оптимізації, який зумовлює наявність хоч якихось вхідних даних, тобто об'єкта оптимізації.

Формування розкладу є складною багатофакторною задачею, існуючі методи можна поділити на три основні групи: ручні, напівавтоматичні та автоматизовані.

Ручні методи історично є найпоширенішими, особливо в невеликих навчальних закладах. Вони передбачають використання паперових схем або електронних таблиць (наприклад, Excel) для створення розкладу.

До переваг цих методів можна віднести гнучкість у врахуванні нестандартних ситуацій, простоту реалізації, оскільки не потребують спеціалізованого програмного забезпечення або особливих знань. Щодо недоліків, можна виділити наступні пункти:

- значна витрата часу;
- висока ймовірність людських помилок;
- обмежена масштабованість, для великих закладів чи складних структур розкладів ручні методи колосально втрачають свою ефективність.

Напівавтоматичні методи – це методи використовують прості програмні інструменти для допомоги у складанні розкладу, наприклад, шаблони чи базові програми, що перевіряють конфлікти.

До їх особливостей можна віднести: повідомлення про наявність конфліктів (але їх вирішення залишається за користувачем), використання простих алгоритмів для перевірки доступності ресурсів тощо. До прикладу: використання Google Sheets або Microsoft Excel з макросами для автоматизації частини задач, використання програм з простою логікою для перевірки наявності викладачів, аудиторій, і так далі.

Перевагами таких методів є: легкість адаптації для конкретного закладу, часткове зниження людських помилок, в порівнянні з ручними методами. Попри це, такі способи також мають свої недоліки, а саме: не вирішують проблему оптимізації розкладу, а лише полегшують ручну працю, не здатні обробляти великий обсяг обмежень чи забезпечувати автоматичне вирішення конфліктів.

Автоматизовані методи до формування розкладу базуються на використанні програмного забезпечення, яке автоматично генерує оптимальний розклад на основі заданих обмежень.

Можна виділити такі способи вирішення цієї задачі: традиційними алгоритмами та сучасними підходами.

До традиційних підходів можна віднести жадібний алгоритм, метод перебору, евристичні методи тощо. Проте, при використанні таких підходів є декілька проблем, які знижують рівень якості результатів, відповідно зводять ефективність даних підходів до рівня напіваавтоматичних методів, а саме: високі обчислювальні витрати при використанні традиційних алгоритмів на великих даних, складність інтеграції складних обмежень (наприклад, пріоритетність певних занять чи спеціальних аудиторій) тощо.

Таким чином, кращим рішенням є використання саме сучасних підходів, які дозволяють вирішити таку задачу більш ефективно без суттєвої втрати якості. До таких підходів належать: мурашиний алгоритм, генетичний алгоритм, машинне навчання та штучний інтелект.

Ці підходи дозволяють обробляти великі обсяги даних з великою кількістю умов та обмежень, забезпечувати оптимальний підхід до вирішення конфліктів та оптимізувати час і ресурси.

Через втрату актуальності ручних та напіваавтоматичних методів, а саме причини масштабування процесів сучасної освіти, було вирішено автоматизувати частину ресурсозатратного процесу, зокрема побудови розкладу.

1.2 Проблеми формування розкладу

Одне з найважливіших питань в організації навчального процесу будь-якого НЗ – задача складання розкладів занять. Це завдання комбінаторного типу, яке характеризується доволі великою розмірністю і містить значну кількість умов та обмежень [3].

Множинність критеріїв цієї задачі і складність об'єкта, для якого будується математична модель, обумовлює необхідність дослідження об'єкта для збільшення функціональних можливостей алгоритмів складання розкладів і, як наслідок, зменшення використовуваної пам'яті і часу розв'язання задачі [6].

Для визначення основних проблем формування розкладу необхідно почати з основ, а саме визначення об'єктів навчального процесу.

Розклад — це документ, що регламентує трудовий ритм і впливає на творчу віддачу вчителів (викладачів); розглядається як фактор оптимізації навчального процесу [7]. Він складається з набору занять, кожне заняття має свого викладача, групу і час, коли воно проводиться.

Викладач — особа, яка має певну спеціалізацію, і може проводити дисципліни, які відповідають її спеціалізації або входять в коло її знань та умінь.

Група або клас — набір навчальних груп здобувачів освіти, які закріплені до певних потоків, об'єднані однією спеціальністю (якщо ВНЗ).

Дисципліна — предмет, який закріплений за певною спеціальністю та потоком.

Аудиторія — це приміщення де проходять заняття з різних дисциплін, кожна аудиторія має певне призначення, від лекційних до лабораторій, а також визначену кількість мість.

Варто зазначити, що розклад складається з огляду на навчальний план, котрий встановлено для певної спеціальності або класу. Таким чином, саме на основі навчального плану встановлюється кількість предметів та «які» дисципліни будуть опановуватись в межах певного навчального року.

Процес побудови розкладу розпочинається підготовкою необхідної інформації: отримання набору спеціальностей та їх потоків, далі для кожної потоку отримуються дисципліни та групи, які закріплені до потоку, наступним етапом проходить формування записів викладач-дисципліна-група-аудиторія.

Якщо попередні етапи, тобто отримання даних та часткове формування записів були питанням «конвеєрної роботи», то на етапі формування повноцінного запису постає складність: збалансоване залучення викладача до викладання, ємність аудиторії, відсутність колізій для різних груп, потоків, часових меж та викладачів тощо; і складність, з людської точки зору, полягає в утриманні усієї цієї інформації в голові або на допоміжних пристроях, якщо мова йде про напівавтоматичні системи, то складність, звичайно, зменшується, проте, все одно, задача розставлення та підбору правильних позицій залишається за людиною, а з точки зору машини складність полягає у обробці усіх даних, дотримання вимог та видача оптимального результату.

Задача може бути розв'язана багатьма способами, до прикладу, повним перебором (що є неефективним рішенням через високу ресурсозатратність), методами теорії розкладів і методами оптимізації. Така задача складна для вирішення її вручну, і, зазвичай, вирішується базуючись на досвіді попередніх років [6].

Тож на даний момент більшість навчальних закладів просто використовують шаблони попередніх років (це, зазвичай, працює при незмінних аргументах, наприклад сталому викладацькому складу), або використовують напівавтоматичні системи, які допомагають систематизувати певний досвід, ввести деякі правила, які допомагають виявляти помилки і спрощувати їх пошуки та виправлення.

1.3 Технології оптимізації

Алгоритми оптимізації працюють шляхом ітеративного пошуку та оцінки різних потенційних рішень, щоб визначити найоптимальніший результат на основі заданих критеріїв. Процес зазвичай включає наступні етапи [4]: ініціалізація, генерація рішень, оцінка, відбір та завершення.

Алгоритм стартує з початкового набору кандидатів на рішення. Нові потенційні рішення генеруються за допомогою спеціальних методів, таких як

мутація, рекомбінація або збурення. Кожне рішення оцінюється на основі завчасно визначених критеріїв, щоб визначити його якість. Алгоритм відбирає найкращі рішення з поточного набору для подальшої ітерації. Процес продовжується до досягнення умови завершення, наприклад, досягнення певної кількості ітерацій або задовільного рішення.

Існує велика кількість алгоритмів оптимізації, проте при попередньому огляді області дослідження, було виділено декілька з них для аналізу:

- евристичні методи;
- мурашиний алгоритм;
- генетичний алгоритм.

Евристичні методи. Серед методів вирішення оптимізаційних задач одним із універсальних та зрозумілих є метод, що може бути застосований у випадках, коли множина можливих рішень M обмежена. Йдеться про метод повного перебору, який полягає у розгляді всіх потенційних варіантів. Цей підхід гарантує отримання результату за умови, що множина M є скінченною (що часто трапляється у дискретному програмуванні), а також доступний ефективний алгоритм для генерації елементів із M і розрахунку значення цільової функції для кожного з них [5].

Таким чином, ми стикаємося з загальноінтелектуальною процедурою. У нових ситуаціях, коли система намагається спланувати свої дії, вона може спробувати перетворити задачу планування у задачу оптимізації. Для цього необхідно визначити множину допустимих рішень M і цільову функцію $f(x)$. Якщо це зробити вдається, тоді метод повного перебору у більшості випадків забезпечить знаходження оптимального рішення [5].

Однак у такого підходу є значний недолік: у багатьох практичних ситуаціях кількість можливих варіантів для перебору може бути настільки великою, що реалізувати цей метод за розумний проміжок часу стає неможливим. У зв'язку з цим розробляються нові методи та алгоритми, які дозволяють скоротити кількість варіантів для аналізу [5].

При цьому не завжди обов'язково прагнути до пошуку саме оптимального рішення. У багатьох випадках достатньо знайти субоптимальне (наближене до найкращого) або просто допустиме рішення. Для цього розробляються правила, які дозволяють уникнути застосування повного перебору і при цьому забезпечити прийнятний результат для більшості задач. Саме на цьому базується концепція евристичних алгоритмів [5].

Водночас далеко не кожна інтелектуальна задача дозволяє явно звести її до оптимізаційної. Часто це ускладнюється неможливістю формалізувати цільову функцію чи обмеження [5].

Процедуру систематизованого перебору на основі послідовного аналізу ймовірних опцій та оцінки їх результатів прийнято називати евристичний пошук [5].

Одним з важливих варіантів евристичного пошуку є метод послідовних поліпшень. Метод послідовних поліпшень, як частина евристичного пошуку, являється інструментом для розв'язання задач, де повний перебір є неможливим через експоненційний ріст обчислювальної складності. У контексті задачі складання розкладу основну ідею цього методу можна сформулювати так: якщо ми маємо певний поточний стан розкладу S_k , то за допомогою визначеної процедури переходимо до нового стану $S_{(k+1)}$, який задовільняє більше обмежень або має кращу цільову функцію [8].

Метод послідовних поліпшень реалізується через ітеративний пошук локально оптимального рішення. На кожній ітерації ми [8]:

1. Оцінюємо поточний розклад S_k за цільовою функцією $f(S_k)$, яка враховує кількість виконаних обмежень і ступінь виконання.

2. Знаходимо зміну ΔS_k , яка покращує значення цільової функції, тобто:
$$f(S_{(k+1)}) = f(S_k) + \Delta f, \Delta f > 0$$

3. Переходимо до нового стану $S_{(k+1)} = S_k + \Delta S$

Задачу складання розкладу можна інтерпретувати як задачу пошуку в дереві рішень. Кожна вершина цього дерева відповідає конкретному стану розкладу. На кожному кроці алгоритм має вибір: додати певне заняття у відповідний слот (визначивши викладача, аудиторію і час). Наприклад, дерево рішень для розкладу з 3 дисциплінами $D=d_1, d_2, d_3$ і 2 часовими слотами може виглядати так [8]:

- Корінь дерева: порожній розклад;
- гілки першого рівня: розклад із першим заняттям у слотах h_1 або h_2 ;
- гілки другого рівня: додавання другого заняття до одного з доступних слотів;
- і так далі до заповнення всього розкладу.

Кількість можливих комбінацій для заповнення розкладу залежить від: кількості дисциплін l , кількості груп n , кількості часових слотів q , кількості викладачів m , кількості аудиторій p . Це дає загальну кількість варіантів:

$$|S|=l*n*m*p*q$$

Зі збільшенням кількості параметрів перебір стає обчислювально недоцільним, що обґрунтовує застосування евристичних методів [8].

Мурашиний алгоритм. Марко Доріго в своїй докторській запропонував «систему мурах» [9]. Наслідуючи природну поведінку колоній мурах, мурашині алгоритми застосовують багатоагентні системи, де агенти діють за дуже простими правилами. Ці алгоритми ефективно вирішують складні комбінаторні задачі та задачі оптимізації [10].

Мурашиний алгоритм — це метаевристичний метод, натхненний поведінкою реальних мурашок під час пошуку найкоротших шляхів між гніздом і джерелом їжі. Основна ідея полягає у використанні віртуальних "мурашок", які поступово будують рішення задачі, залишаючи на своєму шляху "феромони". Інтенсивність феромону використовується для визначення привабливості кожного можливого варіанта, і алгоритм спрямовує пошук у напрямку найкращих знайдених рішень.

Принцип роботи даного підходу укладається у декілька етапів [11]:

1. Ініціалізація.
2. Побудова рішення.
3. Оновлення феромонів.
4. Ітерація.

Розглянемо кожен з них.

В процесі ініціалізації створюється граф задачі, де вершини представляють можливі рішення, потім встановлюється початкова кількість феромонів на всіх ребрах графа.

Побудова рішення проходить наступним чином: кожна «мураха» починає з початкової вершини і поступово проходить через граф приймаючи рішення на основі ймовірності переходу [11]:

$$P_{ij} = \frac{\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}{\sum_{k \in \text{доступні}} \tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}$$

де:

- τ_{ij}^{α} – рівень феромону на ребрі між вершинами i та j ;
- η_{ij}^{β} – евристична оцінка;
- α і β – параметри, що визначають вагу феромону та евристики.

Після завершення проходу усіма мурахами феромони оновлюються:

$$\tau_{ij} \leftarrow (1-p) * \tau_{ij} + \sum_{\text{мурах}} \Delta\tau_{ij},$$

де:

- p – коефіцієнт випаровування феромону;
- $\Delta\tau_{ij}$ – додаток до феромону, якщо мураха пройшла через ребро i, j (зазвичай пропорційний якості розв'язку).

Цей процес повторюється доти, доки не досягнуто оптимального або прийнятного рішення.

Задача складання розкладу для навчальних закладів може бути перетворена на графову задачу, де вершини представляють: часові слоти, пари

викладач-група, аудиторії. Ребра графа відображають можливі зв'язки між цими елементами, а їхні ваги — "вартість" чи "зручність" вибору. Як приклад, кожна вершина відповідає одному з можливих компонентів розкладу, ребра між вершинами показують можливості переходу. Кожна мураха будує повний розклад для всіх занять, проходячи через граф і вибираючи можливі комбінації «група-час-аудиторія» з урахуванням поточного рівня феромонів і евристичної оцінки (рис. 1.1) [11].

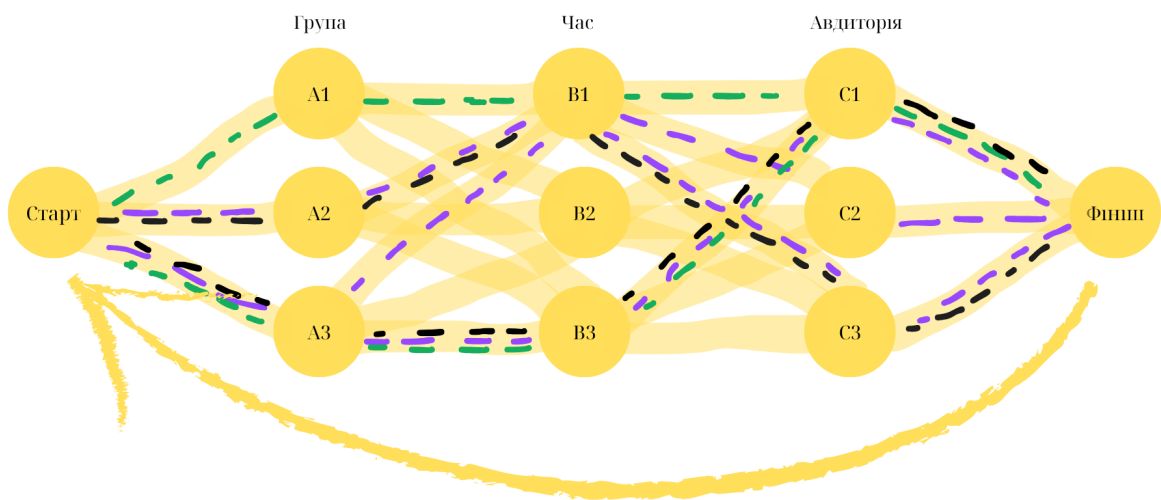


Рис. 1.1. «Мурашиний» граф

Привабливість ребра може визначатися, наприклад, близькістю аудиторії до попередньої, наявністю вільного часу у викладача чи мінімізацією конфліктів. Для кожного знайденого розкладу оцінюється його якість [11]:

- кількість задоволених обмежень;
- мінімізація перевантажень викладачів або груп;
- оптимальність використання ресурсів.

Найкращі рішення посилюють відповідні ребра графу. Після кількох ітерацій алгоритм видає оптимальний розклад або близьке до оптимального рішення.

Генетичний алгоритм — це оптимізаційний метод, заснований на принципах природного відбору та дарвінівській теорії. Він моделює процес біологічної еволюції для пошуку найкращих рішень складних задач. Алгоритм підтримує популяцію можливих рішень (індивідуумів) і еволюціонує їх через кілька поколінь за допомогою генетичних операцій, таких як кросовер і мутація. Кожен індивідуум отримує оцінку, яка відображає його відповідність критеріям оптимізації. Найкращі індивідууми відбираються для подальшого розмноження, що поступово призводить до поліпшення рішень [4].

ГА надзвичайно корисні для вирішення оптимізаційних задач з великими просторами можливих рішень або складними обмеженнями. Вони відмінно підходять для таких завдань, як планування, задача комівояжера, а також оптимізація параметрів у машинному навчанні [4].

Завдання представляється у вигляді масиву, подібного до хромосоми, де кожний елемент масиву називається "хромосомою". Початкова популяція створюється випадковим чином з певної кількості початкових елементів, які називаються "особами". Кожна особа оцінюється за допомогою функції відповідності, що визначає її шанси на виживання. Після цього, на основі отриманих значень відповідності, обираються особи для схрещування (селекція). Генетичні оператори, як-от схрещування (crossover) і мутація (mutation), застосовуються до осіб, створюючи наступне покоління. Це нове покоління також проходить оцінку за допомогою генетичних операторів, і процес селекції та мутації повторюється. Таким чином, моделюється еволюційний процес, що триває кілька життєвих циклів (поколінь), поки не буде досягнуто критерій зупинки алгоритму (рис. 1.2). Критерієм зупинки може бути [12]:

- знаходження глобального, або надоптимального вирішення;
- вичерпання числа поколінь, що відпущені на еволюцію;
- вичерпання часу, відпущеного на еволюцію.

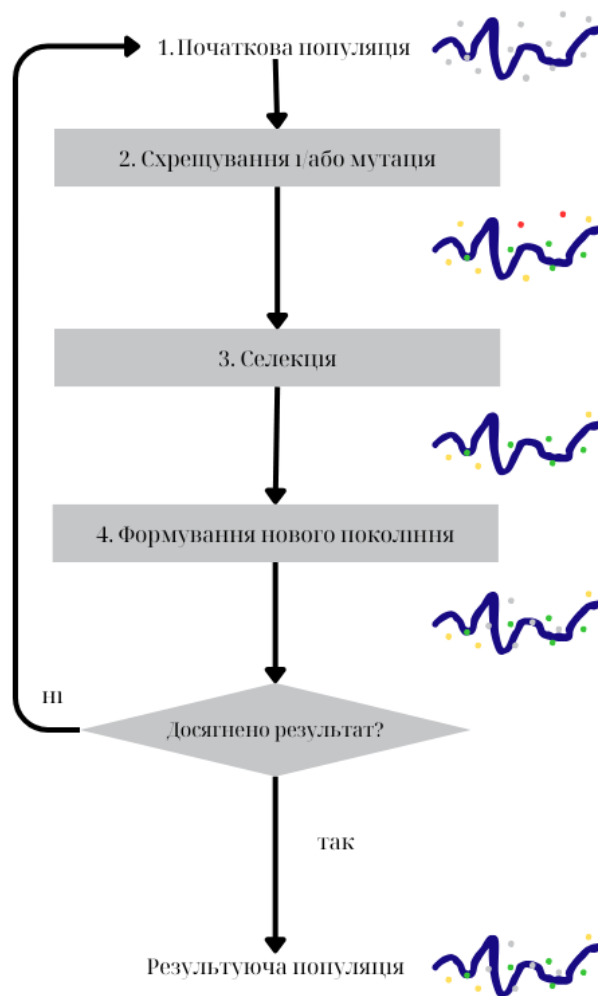


Рис. 1.2. Опис ГА

Задача складання розкладу для навчальних закладів є складною оптимізаційною задачею з великою кількістю обмежень. Генетичний алгоритм дозволяє ефективно знайти збалансоване рішення шляхом ітеративного вдосконалення популяції розкладів.

Етапи застосування ГА для складання розкладу:

- хромосома;
- популяція;
- фітнес-функція;
- схрещування;
- мутація;
- зупинка.

Кожна хромосома відповідає можливому розкладу. Хромосома x представляється як вектор параметрів:

$$x = [g_1, g_2, g_3, \dots],$$

де g_l – ген, що відповідає конкретному елементу розкладу.

Початкова популяція генерується випадковим чином, створюючи кілька варіантів розкладу. Початкова популяція P_0 – це набір N можливих хромосом:

$$P_0 = \{x_1, x_2, x_3, \dots\}.$$

Фітнес-функція оцінює якість розкладу, до прикладу враховує конфлікти між генами, рівномірний розподіл кожного виду генів тощо:

$$f(x) = w_1 * conf(x) + w_2 * classutil(x) + w_3 * teacherload(x),$$

де:

- $conf(x)$ – кількість конфліктів у розкладі;
- $classutil(x)$ – коефіцієнт використання аудиторій;
- $teacherload(x)$ – баланс завантаження вчителів;
- w_1, w_2, w_3 – вагові коефіцієнти.

Основна задача – максимізувати $f(x)$.

Наступним етапом відбувається відбір, використовується метод рулетки. Ймовірність вибору хромосоми пропорційна її фітнес-значенню:

$$p(x_i) = \frac{f(x_i)}{\sum_{j=1}^N f(x_j)}$$

Далі, два розклади обмінюються частинами, щоб створити нові розклади. Відбувається процес схрещування:

$$x_1 = [g_1^{(1)}, g_2^{(1)}, \dots, g_k^{(1)}, g_{k+1}^{(2)}, \dots, g_n^{(2)}],$$

$$x_2 = [g_1^{(2)}, g_2^{(2)}, \dots, g_k^{(2)}, g_{k+1}^{(1)}, \dots, g_n^{(1)}].$$

Мутація робить випадкові зміни в популяціях; випадкова зміна g_l у хромосомі x :

$$g_l = g'_l, \text{ де } g'_l \in G_l,$$

G_l – допустимі значення для гена g_l .

Нове покоління P_{t+1} утворюється як комбінація:

$$P_{t+1} = \text{selection}(P_t) + \text{crossover}(P_t) + \text{mutation}(P_t)$$

Алгоритм завершується, коли досягнуто прийняттого рівня фітнесу або заданої кількості ітерацій:

$$\max(f(x_i)) \geq F_{\text{target}} \text{ або } t = t_{\text{max}}.$$

Генетичний алгоритм дозволяє знаходити близькі до оптимальних розв'язки для складних задач, таких як складання розкладу, навіть за великої кількості обмежень.

1.4 Аналіз вибору алгоритму

В даному розділі буде розглянуто, проаналізовано та обрано технологію, яка буде використана в процесі оптимізації розкладу задля формування кінцевого та робочого результату.

Мурашиний алгоритм. Принцип роботи:

- симулює поведінку мурашиної колонії під час пошуку їжі;
- основа алгоритму — пошук оптимальних маршрутів через оновлення «феромонів».

До переваг можна віднести ефективність в задачах маршрутизації та деяких комбінаційних задач. Щодо недоліків: чутливий до початкових параметрів, зростання кількості можливих комбінацій значно ускладнює процес, важко врахувати складні обмеження.

Як висновок, мурашиний алгоритм добре працює для задач, де важливий пошук найкоротшого маршруту або мінімізація певного показника, але його складно адаптувати для задач із великою кількістю гнучких обмежень, характерних для генерації розкладу.

Метод послідовних поліпшень. Метод поступово покращує поточний зв'язок через перехід до локального кращого стану. Використовує структуру дерева перебору рішень.

Перевагами є:

- простота реалізації;
- прямий пошук у просторі рішень, який можна адаптувати до обмежень.

Недоліки:

- схильність до потрапляння в локальні мінімуми;
- неефективність для задач із великою кількістю змінних і обмежень;
- потребує великої кількості обчислювальних ресурсів при зростанні розмірності задачі.

Метод послідовних поліпшень не є оптимальним для складних задач, оскільки має високий ризик застрягти на локально оптимальному рішенні. Для задачі генерації розкладу, яка потребує глобальної оптимізації, цей підхід не є доцільним.

Генетичний алгоритм. Алгоритм базується на природному відборі й еволюції, створює популяцію можливих розкладів і покращує їх через механізми відбору, схрещування і мутації.

Перевагами даного алгоритму є: глобальний пошук, гнучкість, масштабованість, ітеративне вдосконалення.

Метод ефективний у великому просторі рішень із багатьма локальними мінімумами. Він легко враховує складні обмеження, наприклад, доступність викладачів, типи занять (лекції, практичні), місткість аудиторій тощо. Підхід добре працює навіть із великою кількістю груп, викладачів і предметів, і поступово покращує розклад, забезпечуючи прогрес із кожною генерацією.

Попри це, даний спосіб також має свої недоліки. Алгоритм вимогливий до обчислювальних ресурсів при великій кількості змінних та чутливий до налаштування параметрів.

Генетичний алгоритм є найкращим вибором завдяки здатності шукати глобально оптимальні рішення навіть для задач із великою кількістю змінних та обмежень. Він забезпечує гнучкість і адаптивність, що особливо важливо для створення навчального розкладу.

Порівняльна таблиця (табл. 1.1):

Таблиця 1.1

Критерій	Мурашиний алгоритм	Метод послідовних поліпшень	Генетичний алгоритм
Глобальна оптимізація	Середня	Низька	Висока
Гнучкість	Низька	Низька	Висока
Чутливість до обмежень	Висока	Висока	Низька
Складність реалізації	Середня	Низька	Середня
Обчислювальні витрати	Високі	Високі	Помірні
Адаптація до змін	Складна	Складна	Легка

Висновки до розділу 1

У розділі проведено огляд існуючих методів формування навчальних розкладів, їх переваг та недоліків, а також розглянуто сучасні технології оптимізації, які застосовуються для автоматизації цього процесу.

Ручні та напіваавтоматичні методи, які широко використовуються у навчальних закладах, характеризуються низькою ефективністю, схильністю до помилок і значними часовими витратами. Водночас, автоматизовані підходи демонструють значно кращу продуктивність, особливо для задач із великою кількістю змінних та обмежень.

У межах технологій оптимізації проаналізовано три основні методи:

- Метод послідовних поліпшень є простим у реалізації, але обмеженим через схильність до потрапляння в локальні мінімуми та високу обчислювальну складність для складних задач.

- Мурашиний алгоритм показує себе ефективним у комбінаційних задачах, але його складно адаптувати до великої кількості складних обмежень, характерних для генерації розкладу.

— Генетичний алгоритм вирізняється своєю здатністю здійснювати глобальний пошук, адаптивністю до змін у вихідних даних, а також ефективністю при вирішенні задач із великим простором рішень.

На основі проведеного аналізу зроблено висновок, що генетичний алгоритм є оптимальним вибором для задачі автоматизації формування навчального розкладу. Його здатність враховувати численні обмеження, такі як доступність викладачів, типи занять, місткість аудиторій і специфіку предметів, робить його найбільш ефективним рішенням для цієї задачі. Крім того, він забезпечує масштабованість і дозволяє поступово покращувати якість розкладу з кожною ітерацією, досягаючи глобального оптимуму.

Таким чином, у подальшому дослідженні саме генетичний алгоритм буде використано як основний інструмент для побудови автоматизованої системи формування навчального розкладу.

РОЗДІЛ 2

ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Постановка задачі

Задача автоматизованого формування навчального розкладу полягає у створенні оптимального розподілу навчальних занять у часі й просторі з урахуванням заданих обмежень та вимог. Основною метою є забезпечення ефективної організації навчального процесу, мінімізації конфліктів і максимального задоволення потреб усіх зацікавлених сторін, зокрема студентів, викладачів і адміністрації навчального закладу.

Система повинна виконувати наступні завдання:

1. Генерація розкладу: автоматичне створення розкладу для семестру, забезпечення рівномірного розподілу навантаження серед викладачів і студентів.
2. Урахування жорстких обмежень: уникнення конфліктів у розкладі, відповідність типу занять до типу аудиторій, урахування місткості аудиторій залежно від кількості студентів.
3. Урахування гнучких обмежень: побажання викладачів щодо часу проведення занять, розподіл вихідних та зручних днів для студентів.
4. Оптимізація розкладу: максимальне врахування гнучких обмежень, мінімізація "вікон" у розкладі студентів та викладачів.
5. Ефективність та масштабованість: адаптація до змін у параметрах, можливість розширення для формування розкладу на різні періоди.

Система повинна базуватися на використанні оптимізаційних алгоритмів, які забезпечать створення розкладу у прийнятні часові рамки навіть для великих навчальних закладів із великою кількістю обмежень. Для цього передбачається використання генетичного алгоритму, що поєднує ефективність глобального пошуку з гнучкістю в адаптації до різних обмежень.

2.2 Аналіз функціональних вимог

Спершу необхідно провести ідентифікацію системи. Цей процес включає визначення та опис ключових характеристик, які вирізняють дану систему серед інших. Також до цього належать функціональні вимоги до програми. Вони визначають, які функції та можливості повинно мати програмне забезпечення (ПЗ) [13]. Для візуалізації подано діаграму прецедентів (рис. 2.1).

Дана система повинна мати в собі наступні функції:

- додавання нових даних;
- редагування існуючих даних;
- видалення записів;
- генерація розкладу;
- перегляд розкладу;

Актори: адміністратор в ролі єдиного користувача.

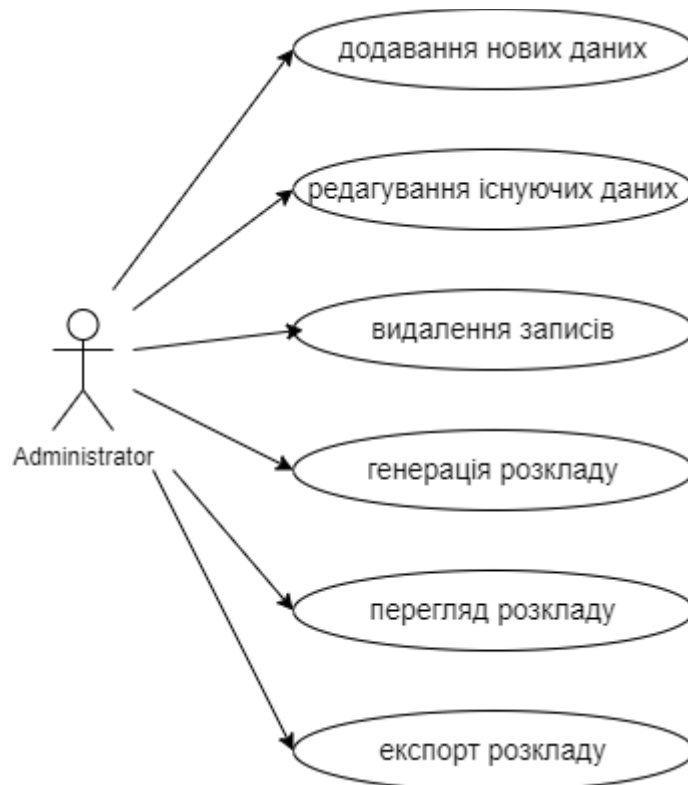


Рис. 2.1. Діаграма прецедентів

Для моделювання та візуалізації зовнішньої поведінки системи без звернення до її внутрішньої реалізації, використано діаграму «чорна скриня» (рис. 2.2).

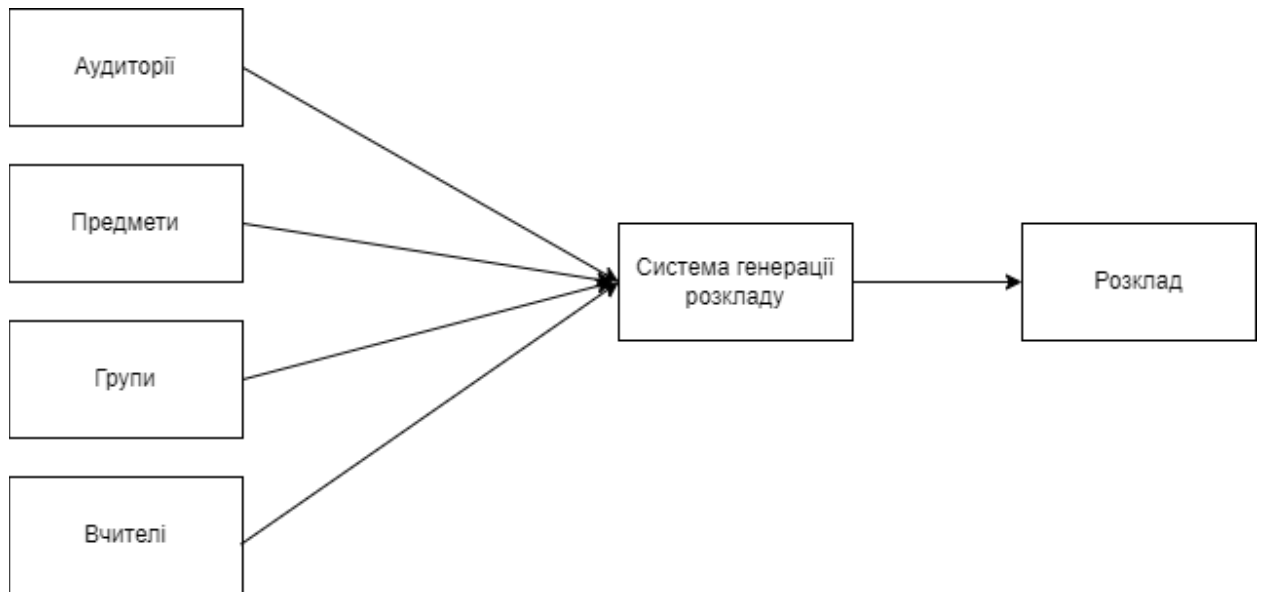


Рис. 2.2. Діаграма «чорна скриня»

Ця діаграма дозволяє поверхнево розглядати систему, що відома своїми вхідними та вихідними даними, при цьому приховуючи свої внутрішні процеси [14].

Наступний етап полягає в створенні функціональної моделі системи (рис. 2.3). Цей тип моделювання зосереджується на описі функцій та їх взаємозв'язків у системі. Він показує, як система функціонує і які завдання виконує для досягнення своїх цілей [15].

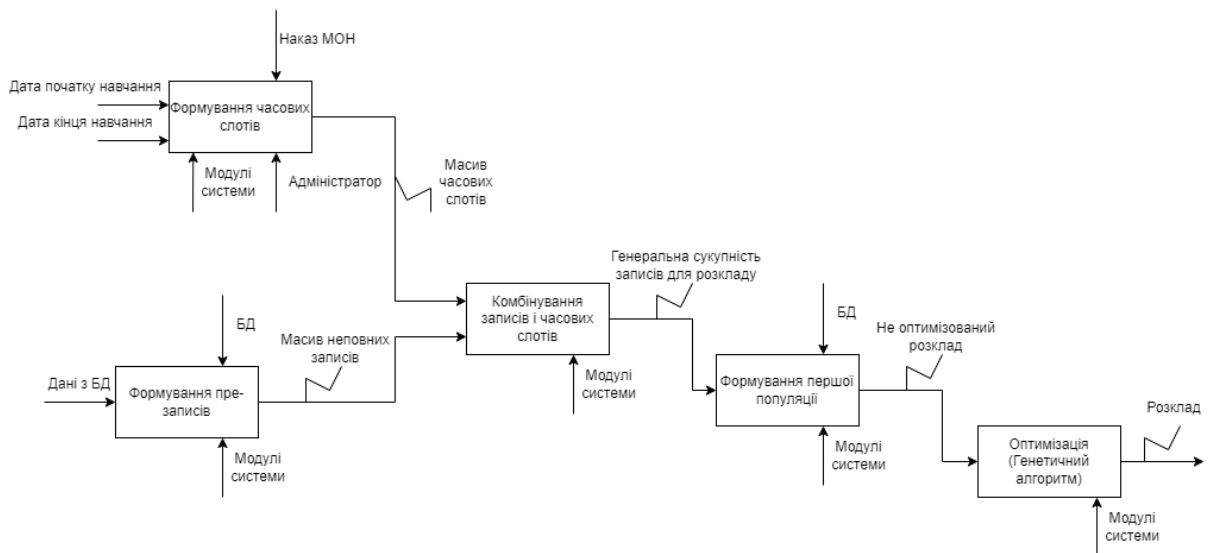


Рис. 2.3. Функціональна модель системи

Запит створюється шляхом натиснення відповідної кнопки, перед цим необхідно внести певні оперативні дані. Запит надсилається від користувача до системи, де підлягає обробці даних, які були введені. Після отримання запиту та даних система починає формувати часові слоти необхідні для подальшої роботи алгоритму. Після цього відбувається формування пре-записів - об'єктів, які містять інформацію про потенційні уроки, які будуть розміщені в розкладі. Як будуть сформовані 2 набори даних, починається процес злиття, при якому записи з різних масивів об'єднуються та утворюють записи, які готові до використання у формуванні розкладу.

Діаграма послідовності (Sequence Diagrams) показує, як взаємодіють об'єкти, що використовуються у сценарії, але не те, як вони пов'язані один з одним. Діаграми послідовності також часто використовуються для того, щоб показати взаємодію компонентів інтерфейсу користувача або компонентів програмного забезпечення [16].

Повідомлення, які надсилаються від одного об'єкта до іншого, зображені у вигляді горизонтальних стрілок. Порядок повідомлень представлений у послідовності зверху вниз і зліва направо, починаючи з першого повідомлення

у верхньому лівому кутку сторінки, а наступні повідомлення розташовуються праворуч і нижче [16].

Подано діаграму послідовностей, щоб зобразити «яким чином» передаються повідомлення між вузлами системи (рис. 2.4).

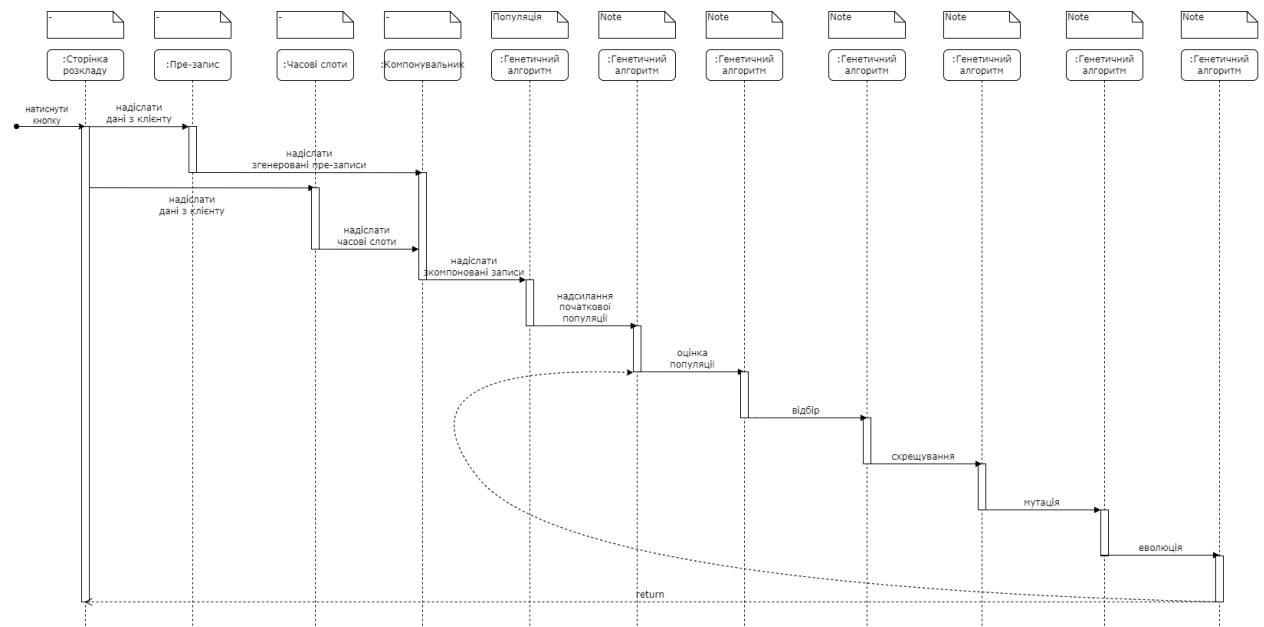


Рис. 2.4. Діаграма послідовності

Функціональний аналіз передбачає розбір, розуміння та описування функцій, функціональних вимог і завдань, що має виконувати система або програмне забезпечення. Цей процес допомагає визначити, що саме має робити система, які функції вона повинна забезпечити та які вимоги задовольнити [13].

2.3 Архітектура системи

Застосунок будується на базі клієнт-серверної архітектури (рис. 2.5), де фронтенд створюється за допомогою Next.js, а серверна частина реалізується з використанням Node.js.

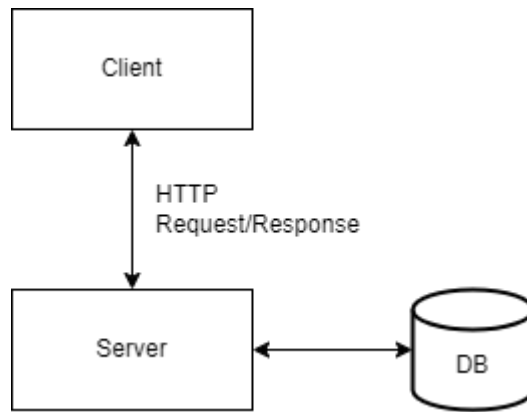


Рис. 2.5. Модулі системи. Клієнт-серверна архітектура

Клієнт-серверна архітектура є широко використовуваним підходом в розробці програмного забезпечення, де система розділяється на дві основні складові частини: клієнтську та серверну.

Клієнтська частина (Frontend) – це частина програмного забезпечення, через яку користувач взаємодіє зі системою. Клієнтська сторона відповідає за представлення інформації користувачеві та обробку його запитів. У веб-додатках це може бути фронтенд (наприклад, веб-сторінка або мобільний додаток), який відображає вміст із сервера, забезпечує взаємодію з користувачем та передає запити на сервер для обробки [17].

Серверна частина (Backend) – це частина, яка обробляє запити від клієнтської сторони, виконує бізнес-логіку, доступ до даних та надає відповіді клієнту. Серверна частина може включати в себе веб-сервер, додаткові служби та базу даних. У випадку веб-додатків, серверна сторона може бути реалізована за допомогою фреймворків або технологій, таких як Next.js, Node.js, Django, Ruby on Rails тощо [17].

Взаємодія між клієнтом та сервером відбувається за допомогою мережових протоколів, таких як HTTP або WebSocket. В даній системі це HTTP. Клієнт відправляє запити на сервер, передаючи дані або створюючи запити на ресурси, і сервер обробляє ці запити, виконує необхідні операції та надсилає відповіді назад.

Клієнт та сервер можуть комунікувати за допомогою різних типів запитів і відповідей, таких як GET (отримання ресурсів), HEAD (запитує метадані ресурсу у вигляді заголовків), POST (створення ресурсів), PUT або PATCH (оновлення ресурсів), OPTION (запитує дозволені параметри зв'язку) та DELETE (видалення ресурсів). Запити можуть містити дані, передані клієнтом, такі як параметри запиту або тіло запиту, яке містить дані для обробки на сервері.

Клієнт-серверна архітектура дозволяє розділити відповідальності між клієнтом та сервером, спрощує розробку, підтримку та масштабування системи. Клієнтська сторона займається представленням інформації та взаємодіє з користувачем, тоді як серверна сторона відповідає за обробку запитів, збереження даних та надання відповідей [17].

При розробці застосунку використовувався функціональний підхід [18]. Перевагами такого підходу є:

- незмінність даних: дозволяє уникати небезпечних побічних ефектів і спрощує розробку;
- функції першого класу: що означає, що вони можуть бути передані як аргументи до інших функцій, повернуті з функцій та збережені в змінних. Це дає змогу реалізувати потужні техніки програмування, такі як замикання і карування функції;
- відсутність побічних ефектів: це означає, що вони не змінюють стану системи або зовнішніх ресурсів. Таким чином, сприяє більш простому тестуванню і розробці;
- імутабельність: використовуються імутабельні структури даних. Це спрощує керування станом програми, дозволяє легше відтворювати стан систем.

Ці особливості роблять функціональне програмування ефективною парадигмою для розробки обчислювальних систем.

2.4 Стек технологій

Для розробки застосунку, було використано популярний фреймворк для створення веб-додатків Express.js.

Express.js – це швидкий, гнучкий і мінімалістичний веб-фреймворк для Node.js. Це фактично інструмент, який спрощує створення веб-додатків та API за допомогою JavaScript на стороні сервера. Express має відкритий вихідний код, який розробляється і підтримується фондом Node.js [19].

Ключові особливості Express.js:

- Проміжне ПЗ та маршрутизація;
- гнучкість та кастомізація;
- мінімалістичний дизайн;
- шаблони;
- статичне обслуговування файлів;
- Node.js інтеграція.

Даний фреймворк спрощує процес розробки Node-додатків. Він використовує JavaScript як мову програмування і забезпечує ефективний спосіб створення веб-додатків та API. За допомогою Express можна легко обробляти маршрути, запити та відповіді, що значно спрощує процес створення надійних та масштабованих додатків. Крім того, це легкий і гнучкий фреймворк, який легко освоїти і який постачається з опціями проміжного програмного забезпечення [19].

Мова програмування обрана як засіб розробки: Typescript.

Typescript – це мова програмування, яка є розширенням JavaScript, що надає статичну типізацію і додаткові функції для розробки великих та складних програмних проєктів. Вона компілюється в чистий JavaScript, що дозволяє використовувати її в будь-якому середовищі, де підтримується JavaScript [20].

Переваги мови TypeScript є: статична типізація підтримка сучасних стандартів JavaScript та масштабованість.

TypeScript дозволяє визначати типи для змінних, параметрів функцій та інших елементів коду. Це дозволяє виявляти помилки на етапі компіляції, що сприяє покращенню якості коду та забезпеченню більшої надійності програмного забезпечення. Крім того, статична типізація допомагає забезпечити кращу підтримку редактора коду з відображенням підказок та автодоповненням [20].

Окрім цього, TypeScript підтримує всі сучасні функції та стандарти JavaScript, такі як ES6, ES7 і ES8 [20].

Більш того, TypeScript допомагає зробити код більш зрозумілим та легко підтримуваним. Він надає можливість визначати інтерфейси, типи та моделі даних, що полегшує розуміння структури програми. Крім того, TypeScript підтримує розділення коду на модулі та використання просторів імен, що допомагає уникнути конфліктів та покращити організацію проекту [20].

Також, TypeScript дозволяє ефективно працювати з великими проектами та командами розробників. Завдяки статичній типізації, ви зможете виявляти та виправляти помилки на ранніх етапах розробки, а також зручно взаємодіяти зі складними структурами даних [20].

У якості місця для зберігання інформації було обрано реляційну БД MySQL.

MySQL – це реляційна система управління базами даних (СКБД) з відкритим вихідним кодом, яка дозволяє користувачам ефективно зберігати, керувати та отримувати структуровані дані. Вона широко використовується для різних додатків, від невеликих проектів до великих веб-сайтів і рішень корпоративного рівня [21].

Базова структура клієнт-серверної структури включає в себе один або кілька пристроїв, підключених до сервера через певну мережу. Кожен клієнт може зробити запит з графічного інтерфейсу користувача (GUI) на своєму

екрані, а сервер надасть бажаний результат, якщо обидва кінці розуміють інструкцію, основні процеси, які відбуваються в середовищі MySQL, однакові, а саме [21]:

- MySQL створює базу даних для зберігання та маніпулювання даними, визначаючи взаємозв'язок кожної таблиці;
- клієнти можуть робити запити, вводячи певні SQL-запити в MySQL;
- серверна програма відповість запитуваною інформацією, і вона з'явиться на стороні клієнта.

Далі буде представлено схему бази даних (рис. 2.6), яка розроблялась в рамках проєкту для зберігання користувацької інформації, а також результатів застосунку.

- Student Group: для збереження студентських груп та їх кількості.
- Study Stream: дає змогу групувати студентів за навчальними потоками для обліку навчального навантаження.
- Study Stream Subject: для управління навчальним навантаженням і формування розкладу.
- Study Stream Group: для об'єднання груп з навчальним потоком.
- Time Slot: для розміщення занять за часом протягом дня.
- Schedule: є основною сутністю для зберігання готового розкладу.

В базі присутні такі зв'язки між таблицями:

1. Teacher → Teacher Subject → Subject

- тип зв'язку: багато-до-багатьох через таблицю teacher_subject;
- опис: один викладач може викладати кілька предметів, а один предмет може викладатися різними викладачами. Таблиця teacher_subject використовується для зберігання цього зв'язку.

2. Subject → Subject Type.

- тип зв'язку: один-до-одного;
- опис: кожен предмет має визначений тип (наприклад, лекція, практичне заняття).

3. Classroom → Classroom Type.

- тип зв'язку: один-до-одного;
- опис: кожна аудиторія має свій тип (наприклад, лекційна зала, лабораторія).

4. Study Stream → Study Stream Subject → Subject.

- тип зв'язку: багато-до-багатьох через таблицю study_stream_subject;
- опис: один навчальний потік має певний набір предметів, а кожен предмет може викладатися у різних потоках. Таблиця study_stream_subject також зберігає години (лекційні, практичні) для кожного предмета у конкретному потоці.

5. Study Stream → Study Stream Stgroup → Student Group.
 - тип зв'язку: багато-до-багатьох через таблицю study_stream_stgroup;
 - опис: один навчальний потік може включати кілька студентських груп, і навпаки, одна група може входити в різні потоки (наприклад, факультативні предмети).
6. Schedule → Teacher.
 - тип зв'язку: багато-до-одного;
 - опис: кожен запис у розкладі прив'язаний до викладача, який веде заняття.
7. Schedule → Subject.
 - тип зв'язку: багато-до-одного;
 - опис: кожен запис у розкладі відповідає конкретному предмету.
8. Schedule → Student Group.
 - тип зв'язку: багато-до-одного;
 - опис: кожен запис у розкладі прив'язаний до певної студентської групи.
9. Schedule → Classroom.
 - тип зв'язку: багато-до-одного;
 - опис: кожен запис у розкладі містить інформацію про аудиторію, в якій проводиться заняття.
10. Schedule → Time Slot.
 - тип зв'язку: багато-до-одного;
 - опис: кожне заняття у розкладі має визначений часовий слот (дата, час початку та завершення).
11. Teacher Subject → Study Stream Subject.
 - тип зв'язку: один-до-багатьох;
 - опис: викладачі прив'язуються до предметів, які вони ведуть у певних потоках.

Ці сутності обрані, виходячи з вимог автоматизації процесу створення навчального розкладу. Вони відповідають основним аспектам освітнього процесу: викладачі, предмети, групи студентів, навчальні приміщення, а також часові рамки занять. Взаємозв'язки між сутностями дозволяють моделювати реальну структуру навчального закладу та створювати релевантний розклад.

MySQL була обрана як система керування базами даних (СКБД) через простоту використання, швидкість, підтримку транзакцій та доступність. Також MySQL є широко поширеною, добре задокументованою та має активну спільноту.

Висновки до розділу 2

Другий розділ документа присвячений реалізації програмного забезпечення для генерації розкладу для НЗ. У розділі описано постановку задачі, яка полягає у створенні програми, яка формує примітивного розкладу, який буде підлягати оптимізації за допомогою генетичного алгоритму.

Для реалізації програми було обрано мову програмування TypeScript, яка є розширенням JavaScript та надає статичну типізацію та додаткові функції для розробки великих та складних програмних проєктів. Також було використано фреймворк Next.js для створення веб-додатків, який надає можливості серверного рендерингу та статичного генерації контенту. Було описано БД, її схему та обґрунтовано вибір такого підходу до зберігання інформації.

У розділі також описано модулі системи та клієнт-серверну архітектуру, яка була використана для реалізації програми.

РОЗДІЛ 3

РОЗРОБКА ГЕНЕТИЧНОГО АЛГОРИТМУ

3.1 Опис генетичного алгоритму

Генетичний алгоритм шукає оптимальне рішення, моделюючи процес еволюції. Замість аналізу всіх можливих рішень, він працює з множиною можливих варіантів (популяцією) та поступово вдосконалює їх через серію ітерацій (поколінь).

Процес роботи ГА можна описати в сім етапів, де останній є логічним, тобто є завершенням:

- ініціалізація популяції;
- оцінка;
- селекція;
- схрещування;
- мутація;
- породження нового покоління;
- завершення.

Початкова популяція складається з набору кандидатів. Кожен кандидат описується за допомогою генетичного представлення — хромосоми, яка є сукупністю параметрів (генів). У нашій задачі хромосома може бути розкладом, а генами хромосоми будуть: аудиторія, викладач, група, час, дисципліна.

Кожен індивід оцінюється за допомогою фітнес-функції, яка визначає "якість" рішення. Наприклад, у даній задачі формування розкладу фітнес-функція враховує такі критерії:

- відсутність конфліктів між групами, викладачами та аудиторіями;
- дотримання обмежень щодо доступності викладачів;
- оптимальне використання ресурсів.

Кандидати з вищими значеннями фітнес-функції мають більше шансів бути обраними для участі у створенні нового покоління. Вибрані батьки комбінуються для створення нових індивідів (нащадків). Схрещування дозволяє об'єднати "сильні" сторони двох рішень. Наприклад: у хромосомах, що описують розклад, можна обмінюватися інформацією про час чи аудиторії.

$$C_1=(P_1[1:k],P_2[k+1:n])$$

$$C_2=(P_2[1:k],P_1[k+1:n]), \text{ де}$$

k – точка схрещування, P_1, P_2 – індивіди, котрих сильні гени, будуть передані до наступних поколінь.

Для уникнення застрягання у локальному мінімумі, вносяться випадкові зміни в хромосоми. Наприклад, у задачі розкладу це може бути перестановка занять у межах певного тижня.

Нове покоління формується з нащадків і кращих індивідів попереднього покоління (елітний відбір). Алгоритм завершується, якщо було досягнуто визначну кількість поколінь, або досягнуто оптимального рішення.

Далі подано псевдокод ГА, який буде певною мірою відображати реальну кодову базу.

На початку визначаються основні параметри алгоритму. Розмір популяції встановлює кількість рішень, які одночасно обробляються. Максимальна кількість поколінь визначає, скільки ітерацій алгоритм виконуватиме. Ймовірності мутації та схрещування впливають на частоту змін у популяції, а кількість елітних індивідів визначає, скільки найкращих рішень зберігатимуться в кожному поколінні.

Фрагмент коду 3.1

```
function GeneticAlgorithm():
```

```
    // Параметри алгоритму
```

```
    populationSize ← N
```

```
    maxGenerations ← M
```

```
    mutationRate ←  $\mu$ 
```

```
    crossoverRate ←  $\rho$ 
```

```
    // Розмір популяції
```

```
    // Максимальна кількість поколінь
```

```
    // Ймовірність мутації
```

```
    // Ймовірність схрещування
```

```
elitismCount ← E // Кількість елітних індивідів
```

Алгоритм починається з ініціалізації популяції, де створюється набір випадкових рішень. У кожній ітерації для всіх індивідів популяції обчислюється фітнес-функція, яка оцінює якість кожного рішення. Потім виконується селекція, де обираються батьки для подальшого схрещування. Цей етап забезпечує, що кращі рішення мають більше шансів передати свої "гени" наступному поколінню.

Фрагмент коду 3.2

```
// Ініціалізація
population ← InitializePopulation(populationSize)

generation ← 0
while generation < maxGenerations do:
    // Обчислення фітнес-функції для кожного індивіда
    for individual in population do:
        individual.fitness ← EvaluateFitness(individual)

    // Селекція
    selectedParents ← Selection(population)
```

Схрещування відбувається між парами батьків, обраних на попередньому етапі. Для цього використовується ймовірність схрещування. У випадку успішного схрещування утворюються два нових "нащадки" шляхом комбінації генів батьків. Якщо схрещування не відбувається, обрані батьки просто копіюються до наступного покоління.

Фрагмент коду 3.3

```
// Схрещування
offspring ← []
for i from 1 to populationSize/2 do:
    parent1, parent2 ← SelectPair(selectedParents)
    if Random() < crossoverRate:
        child1, child2 ← Crossover(parent1, parent2)
    else:
        child1 ← parent1
        child2 ← parent2
    offspring.add(child1)
```

```
offspring.add(child2)
```

На етапі мутації у випадково обраних генів окремих індивідів відбуваються зміни. Цей процес додає випадковість до алгоритму, допомагаючи уникати локальних мінімумів у пошуку.

Фрагмент коду 3.4

```
// Мутація
for child in offspring do:
    if Random() < mutationRate:
        Mutate(child)
```

Після цього формується нове покоління, яке складається з елітних індивідів попереднього покоління та нових нащадків. Цикл повторюється до досягнення заданої кількості поколінь.

Фрагмент коду 3.5

```
// Формування нового покоління
newPopulation ← []
// Додавання елітних індивідів
elite ← SelectElite(population, elitismCount)
newPopulation.add(elite)

// Додавання нащадків
newPopulation.add(offspring)

population ← newPopulation

generation ← generation + 1
```

Після завершення алгоритму вибирається найкращий індивід з фінальної популяції. Цей індивід представляє оптимальне рішення задачі.

Фрагмент коду 3.6

```
// Повертаємо найкраще рішення
bestIndividual ← FindBestSolution(population)
return bestIndividual
```

Допоміжні функції забезпечують ініціалізацію популяції, обчислення фітнесу, відбір батьків, реалізацію схрещування, мутації та пошук найкращого рішення. Кожна функція виконує свою специфічну задачу, яка інтегрується в основний алгоритм.

Фрагмент коду 3.7

```
// Допоміжні функції
function InitializePopulation(size):
    population ← []
    for i from 1 to size do:
        individual ← GenerateRandomSolution()
        population.add(individual)
    return population

function EvaluateFitness(individual):
    // Оцінюємо якість розкладу: враховуємо конфлікти, доступність
    викладачів тощо
    return ComputeFitness(individual)

function Selection(population):
    // Відбір батьків для схрещування (наприклад, турнірний відбір)
    return TournamentSelection(population)

function Crossover(parent1, parent2):
    // Одноточковий кросовер
    crossoverPoint ← RandomIndex()
    child1 ← CombineGenes(parent1[:crossoverPoint],
parent2[crossoverPoint:])
    child2 ← CombineGenes(parent2[:crossoverPoint],
parent1[crossoverPoint:])
    return child1, child2

function Mutate(individual):
    // Випадкова зміна одного або кількох генів
    geneIndex ← RandomIndex()
    individual.genes[geneIndex] ← GenerateRandomGene()

function SelectElite(population, count):
    // Вибираємо "найкращі" рішення на основі фітнес-функції
    return SortByFitness(population).take(count)
```

```
function FindBestSolution(population):
    // Повертаємо індивід з найвищим значенням фітнес-функції
    return MaxByFitness(population)
```

3.2 Реалізація алгоритму

Для того, щоб основний алгоритм отримав на вхід структуру, яку він буде обробляти необхідно побудувати підготовчу логіку, яка обробить збережену інформацію та складе її в структуру. Цими процесами займаються декілька функцій, які доступні за контролером ScheduleController та роутом /schedule/generate, а саме: preparePreScheduleRecord, combineTimeSlotsWithSchedule, generateLimitedSchedule, – розглянемо їх детальніше.

Підготовчий алгоритм. Функція preparePreScheduleRecord (додаток А) створює запис вчитель-група-предмет-авдиторія, керуючись простими умовами, наприклад: «група з 35 людей, не може навчатись в аудиторії з місткістю 30 людей» або «вчитель математики не може викладати біологію». Перевірка умов відбувається по ключам, оскільки вони містять достатньо інформації для того, щоб відрізнити записи (див. рис. 2.6). Функція повертає об'єкт розкладу, який відповідає інтерфейсу ISchedule (інтерфейс відповідний до моделі з однією різницею — відсутність ключа), проте з відсутнім значенням часового слоту. Часові слоти генеруються окремо, іншим контролером та за іншим роутом, а саме: TimeSlotController та /time-slot/generate/. Для того, щоб згенерувати часові слоти необхідно знати тривалість дисципліни, початок та кінець навчання (в межах семестру), початок та кінець навчання (в межах дня).

Наступним етапом буде об'єднання часових слотів та попередньо-підготовлених записів з функції preparePreScheduleRecord цим займається функція combineTimeSlotsWithSchedule (додаток Б). Варто зауважити, що при об'єднанні даних записів утворюються усі можливі комбінації з пар часовий

слот-запис. Це необхідно для того, щоб мати повну варіативність записів у різних часових проміжках та потім знадобиться для формування повного розкладу (готового до оптимізації).

Остання функція в підготовчому алгоритмі є `generateLimitedSchedule` (додаток В). Дана функція займається створенням початкової популяції для генетичного алгоритму, водночас її можна віднести як до підготовчої частини до основного алгоритму так і частиною ГА. Варто відзначити, що саме тут використовується попередньо створена повнота варіацій записів за часом, це здійснено для набору записів в структуру, які будуть задовільняти вимогу по часу, який виділений на конкретну дисципліну за навчальним планом.

ГА в коді представлений множиною різних функцій, які відповідають певним етапам, котрі були описані в даній роботі раніше, при цьому зібраними в одній основній — `geneticAlgorithm`.

Дана функція складається з викликів кожного з етапів, окрім ініціалізації первинної популяції, оскільки вона була створена попередньо. Для початку визначимось з поняттями, які будуть використовуватись в описі.

Ген — це елемент, який входить до складу хромосоми і визначає певну характеристику, тобто несе певну кількість даних. В межах даної роботи генами є записи розкладу, тобто «вчитель-клас-група-дисципліна-час».

Хромосома — це елемент популяції, котрий складається з набору генів, тобто це розклад, який буде використовуватись в процесі еволюції популяцій.

Популяція — це набір хромосом (різних розкладів), які змінюються з покоління в покоління. Кожне нове покоління, це набір хромосом, які увібрали в себе усі найкращі гени з попередньої покоління (популяції). Кількість поколінь задано всередині системи, проте, в майбутньому, є перспектива в тому, щоб дати користувачам визначати це самостійно.

Першою функцією, яка запускатиметься на початку кожного покоління є `evaluateFitness` (додаток Г). Дана функція використовується для оцінки хромосом, тобто перевірки, щоб гени відповідали поставленим умовам,

зокрема, щоб вчителі не були в один час в різних місцях, щоб маленькі аудиторії не були поставлені для великих груп, щоб одна і та ж аудиторія не експлуатувалась різними викладачами і групами, тощо. Також, там передбачені м'які обмеження, котрі додають балів до визначеної хромосоми, наприклад навантаження на одного вчителя протягом одного дня. По результатам, цієї функції ми отримуємо масив оцінок, які індексно відповідають масиву хромосом. Якщо в даному масиві знаходимо розклад з оцінкою ≥ 0 , то завершуємо алгоритм та подаємо дану хромосому, як результат.

Далі, — етап вибору батьків. На цьому етапі необхідно було обрати спосіб вибору батьків та реалізувати його. Було знайдено декілька способів, вибору, а саме: вибір за принципом рулетки, турнірний вибір, вибір за ранжуванням, відбір з жорстким копіюванням та стабілізуючий відбір.

Відбір за принципом рулетки працює наступним чином:

1. Кожна хромосома отримує "вагу", пропорційну її значенню придатності (fitness).
2. Уявляємо собі колесо рулетки, де кожен сектор відповідає хромосомі, а розмір сектора — значенню придатності.
3. Вибір особини здійснюється випадковим чином: колесо обертається, і обирається хромосома, на якій зупинилась стрілка.

Перевагами такого підходу є простота у реалізації та більша імовірність кращих рішень, попри це є і свої недоліки — даний підхід уразливий до ефекту «домінування»: якщо одна хромосома має дуже велику вагу, то вона може повністю захопити популяцію [22].

Турнірний вибір базується на організації «змагань» між хромосомами: випадково вибирається підмножина хромосом із популяції, з цієї підмножини обирається найкраща хромосома (з найвищою оцінкою), процес повторюється до моменту поки усі хромосоми не пройдуть турнір. Перевагами даного підходу є гнучкість, а також низький ризик «домінування». До недоліків

можна віднести: збільшення імовірності «домінування» при великих підмножинах та випадковий вибір при надто малих підмножинах [23].

У методі відбору за ранжуванням усі хромосоми ранжуються за рівним їх придатності. Таким чином, до кожної хромосоми присвоюється ранг — найкраща хромосома отримує ранг 1, трішки гірша — 2, і так далі. Імовірність вибору базується не на абсолютному значенні оцінки, а на рангу хромосоми, при цьому розподіл імовірності часто є лінійним або експоненціальним, щоб кращі хромосоми все ж мали більший шанс. До переваг даного методу відносять: рівномірний відбір та запобігання домінування. Недоліками є уповільнення зближення до оптимального рішення, якщо різниця у оцінках хромосом невелика [24].

Проаналізувавши ці методи, було прийнято рішення обрати турнірний метод вибору (додаток Г).

Після обрання батьків необхідно провести операцію схрещування, щоб породити нову популяцію. Є декілька способів проведення цієї операції: одноточкове схрещування, двоточкове схрещування та уніформне схрещування.

Одноточкове схрещування полягає у розділенні двох батьківських хромосом у випадково вибраній точці, після чого об'єднуються ці частини, щоб створити два нових нащадки [25]. Наприклад:

Батько 1 = 10101 | 0011

Батько 2 = 11010 | 1100

Нашадки: 10101 | 1100 та 11010 | 0011.

Двоточкове схрещування визначаються трьома сегментами і середній сегмент обмінюється між батьками [26]. Наприклад:

Батько 1 = 101 | 010 | 011

Батько 2 = 110 | 111 | 100

Нашадки: 101 | 111 | 011 та 110 | 010 | 100

Уніформне схрещування — це метод, при якому кожен ген нащадка обирається випадково з однієї з батьківських хромосом [26]. Наприклад:

Батько 1 = 10101

Батько 2 = 11010

Нащадки: 11101 та 10010

Одноточкове схрещування добре працює для задач, де порядок чи структура даних важливі. Двоточкове схрещування дозволяє передавати більше інформації від обох батьків. Уніформне підходить для задач із випадковим розподілом ознак, що сприяє більшій варіативності.

Було обрано уніформне схрещування (додаток Д) для того, щоб отримати більшу варіативність, оскільки початкові хромосоми абсолютно не оптимізовані.

Останньою дією є мутація (додаток Е). Мутація полягає в тому, щоб випадковим чином змінювати частини хромосом (генів).

Даний ГА обгорнутий в рекурсію (додаток Є), для того, щоб обробляти покоління різних популяцій.

3.3 Покроковий опис-інструкція

Для взаємодії з сервісом необхідно мати відповідний клієнт, ним може виступати розроблена Front-end частина системи, використати Postman або писати запити власноруч через термінал. В межах даної роботи використовувався Postman (рис. 4.1).

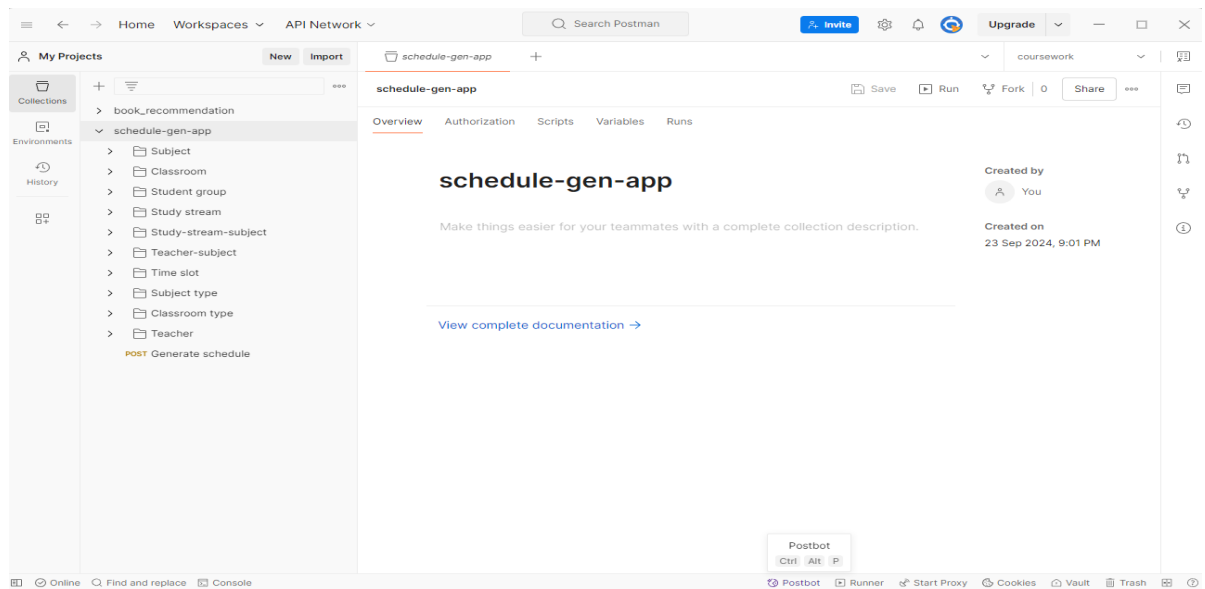


Рис. 4.1. Postman

Далі, необхідно занести дані в таблиці, для розробників клієнтської частини подано усі роути в поточній версії системи:

- $\{\text{host}\}/\text{api}/\text{teacher}/\text{create}$ (POST);
- $\{\text{host}\}/\text{api}/\text{teacher}/\text{update}/:\text{id}$ (PATCH);
- $\{\text{host}\}/\text{api}/\text{teacher}/\text{delete}/:\text{id}$ (DELETE);
- $\{\text{host}\}/\text{api}/\text{teacher}/:\text{id}$ (GET);
- $\{\text{host}\}/\text{api}/\text{teacher}/$ (GET);
- $\{\text{host}\}/\text{api}/\text{subject}/\text{create}$ (POST);
- $\{\text{host}\}/\text{api}/\text{subject}/$ (GET);
- $\{\text{host}\}/\text{api}/\text{subject}/\text{update}/:\text{id}$ (PATCH);
- $\{\text{host}\}/\text{api}/\text{subject}/\text{delete}/:\text{id}$ (DELETE);
- $\{\text{host}\}/\text{api}/\text{subject}/:\text{id}$ (GET);
- $\{\text{host}\}/\text{api}/\text{subject}/\text{type}/\text{create}$ (POST);
- $\{\text{host}\}/\text{api}/\text{subject}/\text{type}/$ (GET);
- $\{\text{host}\}/\text{api}/\text{subject}/\text{type}/\text{update}/:\text{id}$ (PATCH);
- $\{\text{host}\}/\text{api}/\text{subject}/\text{type}/\text{delete}/:\text{id}$ (DELETE);
- $\{\text{host}\}/\text{api}/\text{subject}/\text{type}/:\text{id}$ (GET);

- {host}/api/student-group/ (GET);
- {host}/api/student-group/create (POST);
- {host}/api/student-group/update/:id (PATCH);
- {host}/api/student-group/delete/:id (DELETE);
- {host}/api/student-group/:id (GET);
- {host}/api/classroom/ (GET);
- {host}/api/classroom/create/ (POST);
- {host}/api/classroom/update/:id (PATCH);
- {host}/api/classroom/delete/:id (DELETE);
- {host}/api/classroom/:id (GET);
- {host}/api/classroom/type/ (GET);
- {host}/api/classroom/type/create/ (POST);
- {host}/api/classroom/type/update/:id (PATCH);
- {host}/api/classroom/type/delete/:id (DELETE);
- {host}/api/classroom/type/:id (GET);
- {host}/api/study-stream/create (POST);
- {host}/api/study-stream/update/:id (PATCH);
- {host}/api/study-stream/delete/:id (DELETE);
- {host}/api/study-stream/:id (GET);
- {host}/api/study-stream/ (GET);
- {host}/api/study-stream-subject/ (GET);
- {host}/api/study-stream-subject/create (POST);
- {host}/api/study-stream-subject/update/:id (PATCH);
- {host}/api/study-stream-subject/delete/:id (DELETE);
- {host}/api/study-stream-subject/:id (GET);
- {host}/api/teacher-subject/ (GET);
- {host}/api/teacher-subject/create (POST);
- {host}/api/teacher-subject/update/:id (PATCH);
- {host}/api/teacher-subject/delete/:id (DELETE);

- {host}/api/teacher-subject/:id (GET);
- {host}/api/time-slot/generate (POST);
- {host}/api/schedule/generate (POST).

Після того, як було додано усю інформацію, необхідно згенерувати часові слоти на навчальний термін (4.1):

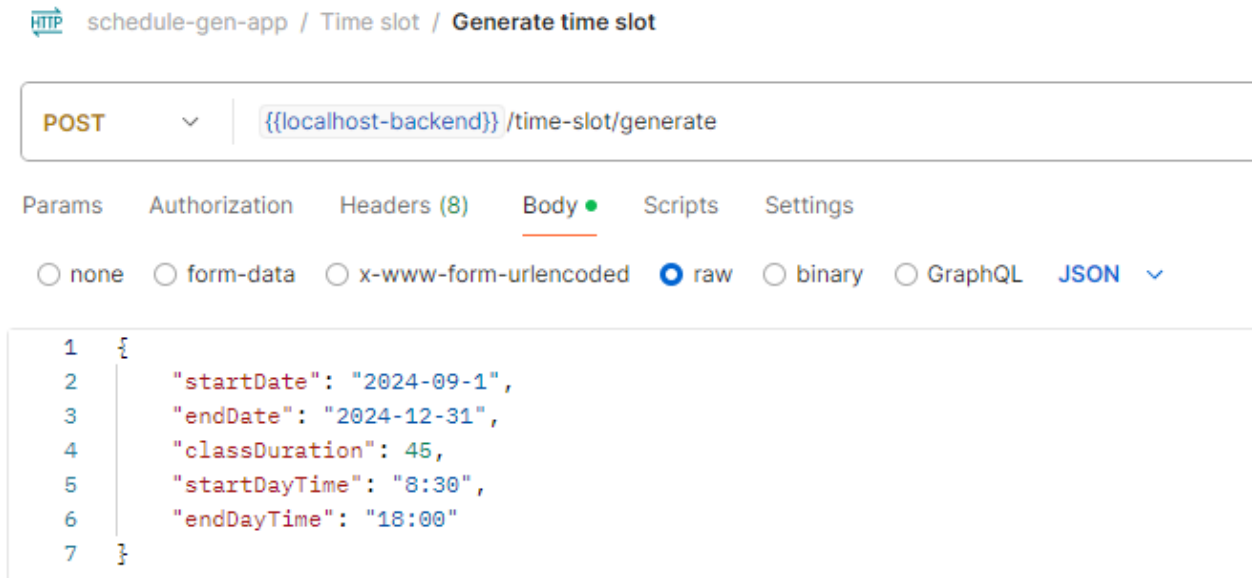


Рис. 4.1 Генерація часових слотів

Часові слоти зберігаються в базі даних, далі можна генерувати розклад (рис. 4.2).

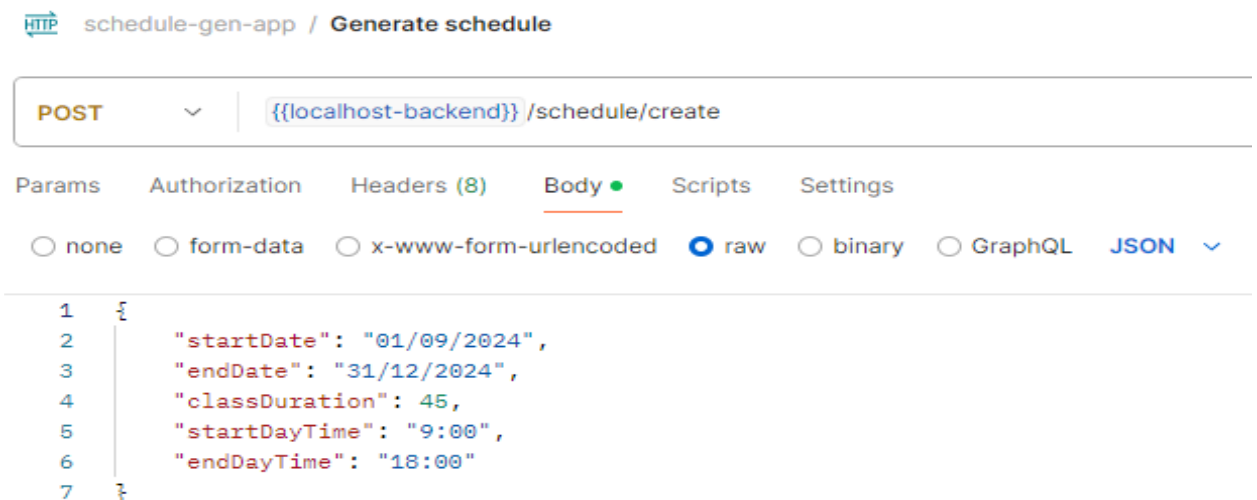


Рис. 4.2. Генерація розкладу

Процес генерації на даний момент складає приблизно 1 годину (тестові дані), але це залежить від кількості елементів для яких необхідно зробити розклад, тобто складність наразі складає $O(n)$.

Розклад зберігається в базі даних також, тож необхідно буде їх дістати GET запитом та висвітлити на front-end частині застосунку, якщо такий існує або вивести іншим зручним для користувача шляхом.

Висновки до розділу 3

У цьому розділі було розроблено генетичний алгоритм для автоматизованого формування розкладу, який враховує вимоги та обмеження навчального процесу. Генетичний алгоритм базується на концепціях еволюційної оптимізації, таких як селекція, схрещування, мутація та елітний відбір. Цей підхід дозволяє ефективно знаходити оптимальні рішення, мінімізуючи конфлікти між ресурсами (групами, викладачами, аудиторіями) та забезпечуючи дотримання заданих критеріїв якості.

Окрім основної логіки генетичного алгоритму, було реалізовано підготовчий етап, який забезпечує формування структури даних для подальшої обробки. Це включає перевірку відповідності груп і аудиторій, генерацію часових слотів та створення початкової популяції.

Розроблений алгоритм демонструє гнучкість і можливість адаптації до специфічних вимог, таких як тривалість занять, графіки викладачів і обмеження навчального плану. Завдяки модульному підходу реалізація забезпечує легку модифікацію й масштабування, що робить її придатною для застосування у різних навчальних закладах.

Також у цьому розділі було представлено технічну реалізацію взаємодії з розробленим сервісом, включаючи API маршрути, використані для CRUD-операцій з основними сутностями системи. Детально описано процеси введення даних у базу, створення часових слотів, генерації розкладу, а також

збереження й отримання результатів за допомогою REST API.

У роботі використовувався інструмент Postman для тестування та відлагодження запитів, що демонструє зручність перевірки функціональності сервісу. Крім того, подано перелік усіх доступних маршрутів API, що забезпечує структурований підхід до інтеграції клієнтської частини із сервером. Ці маршрути охоплюють ключові об'єкти, такі як викладачі, предмети, групи студентів, аудиторії, навчальні потоки, часові слоти та розклад.

Процес генерації часових слотів і розкладу реалізовано таким чином, щоб зберігати результати в базі даних для подальшого використання. Складність алгоритму генерації розкладу, яка на даний момент оцінюється як $O(n)$, є прийнятною для тестових даних, однак її можна оптимізувати для роботи з великими наборами даних.

Генерований розклад може бути відображений на клієнтській частині системи, якщо така передбачена, або переданий у зручному форматі для користувача. Це демонструє універсальність і практичність розробленої системи.

Загалом, у розділі описано процеси, які забезпечують повний цикл роботи з даними в системі: від створення та збереження до генерації й отримання кінцевого результату. Це підтверджує високу функціональність API, а також готовність системи до інтеграції з іншими компонентами або до розширення її функціональних можливостей.

ВИСНОВКИ

У ході виконання роботи було розроблено автоматизовану систему для генерації розкладу занять з використанням генетичного алгоритму. Основною метою цієї розробки було створення ефективного та адаптивного інструменту, здатного вирішувати завдання формування розкладів, що відповідають різноманітним критеріям і обмеженням, які виникають у навчальних закладах.

Розробка включала кілька основних етапів: аналіз предметної області, моделювання генетичного алгоритму, розробку підготовчих алгоритмів, а також інтеграцію всіх компонентів у єдину програмну систему.

Генетичний алгоритм було обрано через його здатність ефективно працювати в умовах багатокритеріальної оптимізації та великої кількості обмежень. Алгоритм моделює природний еволюційний процес, включаючи такі етапи, як селекція, схрещування, мутація та елітний відбір. Цей підхід дозволив створювати оптимальні або близькі до оптимальних рішення навіть у складних умовах.

Підготовчі алгоритми, що включали обробку вхідних даних та створення початкової популяції, забезпечили правильне формування базових елементів для роботи основного алгоритму. Ці елементи враховували всі вимоги до розкладу, зокрема: уникнення конфліктів між викладачами, групами та аудиторіями; дотримання часових і ресурсних обмежень; оптимальне розподілення навчального часу.

У процесі реалізації було створено низку функцій і модулів, які взаємодіяли через структурований API. Зокрема, модулі підготовки даних, генерації часових слотів та основного алгоритму були побудовані так, щоб забезпечити їхню масштабованість і можливість повторного використання.

Результатом роботи стала програмна система, яка успішно формує розклади, відповідні заданим критеріям. Вона має гнучку архітектуру, що дозволяє адаптувати її до специфічних потреб навчальних закладів різного типу. Крім того, система може бути вдосконалена в майбутньому, зокрема

шляхом додавання нових обмежень, критеріїв оптимізації та інтеграції з іншими інформаційними системами.

Загалом, розробка автоматизованої системи на основі генетичного алгоритму підтвердила ефективність цього підходу до вирішення складних оптимізаційних завдань, проте вона також має свої недоліки, а саме висока ресурсозатратність для обчислення — чим більше вхідних даних, тим довше працює алгоритм. Робота може слугувати основою для подальших досліджень та практичного впровадження в реальних умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Можливості для адміністрації – smart school. Smart-School. — [Електронний ресурс]. — Режим доступу: <https://smart-school.com.ua/mozhlivosti-dlya-administratsii/>.
2. Автоматизоване складання розкладу для вишів. Освіта.UA. — [Електронний ресурс]. — Режим доступу: <https://osvita.ua/vnz/53319/>.
3. Островська Г., Мічківський С. Формування розкладу заліково-екзаменаційної сесії у вищих навчальних закладах. Збірники наукових праць професорсько-викладацького складу ДонНУ імені Василя Стуса. — [Електронний ресурс]. — Режим доступу: <https://jpvs.donnu.edu.ua/article/view/7243>.
4. Що таке алгоритм оптимізації — терміни та визначення у сфері кібербезпеки. VPN Unlimited – Fast & Secure VPN service. — [Електронний ресурс]. — Режим доступу: <https://www.vpnunlimited.com/ua/help/cybersecurity/optimization-algorithm>.
5. Повний перебір. StudFiles. — [Електронний ресурс]. — Режим доступу: <https://studfile.net/preview/5745312/page:2/#4>.
6. Снитюк В., Сіпко О. Технологія еволюційного формування розкладів у закладах вищої освіти: монографія. Київ: ФОП Піча Ю. В., 2022. 135 с.
7. Сіпко О. Моделі та еволюційний метод складання розкладу занять у вищому навчальному закладі. КНУ ім. Шевченка. — [Електронний ресурс]. — Режим доступу: <https://mydisser.com/en/catalog/view/sipko-olena-mikolaivna-modeli-ta-evolyuciyniy-metod-skladannya-rozkladu-zanyat-u-vischomu-navchalno.html>.
8. Шурпенкова Р. К., Демко І. І. Евристичні методи генерування, оцінки і вибору альтернатив. Бухгалтерський облік, аналіз та аудит. 2016. — [Електронний ресурс]. — Режим доступу: [https://ird.gov.ua/sep/sep20162\(118\)/sep20162\(118\)_151_ShurpenkovaR,DemkoI.pdf](https://ird.gov.ua/sep/sep20162(118)/sep20162(118)_151_ShurpenkovaR,DemkoI.pdf)

9. Dorigo M., Di Caro G., Stützle T. Ant algorithms: підручник. 8th ed. Brussels: IRIDIA, 2000. Vol. 16: Ant algorithms. 105 p.
10. Ant system: optimization by a colony of cooperating agents: підручник. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 1996. Vol. 26: Ant system: optimization by a colony of cooperating agents. 12 p.
11. Dorigo M. Ant colony optimization: підручник. MIT Press, Cambridge, Mass., 2004. 305p.
12. Katoch, S., Chauhan, S.S. & Kumar, V. A review on genetic algorithm: past, present, and future. Multimed Tools Appl 80, 8091–8126 (2021). <https://doi.org/10.1007/s11042-020-10139-6>
13. Функціональні вимоги та нефункціональні вимоги - Інший. myservername.com. — [Електронний ресурс]. — Режим доступу: <https://uk.myservername.com/features-functional-requirements>.
14. Описання систем за допомогою моделі «чорної скрині». StudFiles. — [Електронний ресурс]. — Режим доступу: <https://studfile.net/preview/9247717/page:8/>.
15. Побудова функціональної моделі системи. StudFiles. — [Електронний ресурс]. — Режим доступу: <https://studfile.net/preview/381095/page:14/>
16. Махум Z. Діаграма послідовності (Sequence Diagrams). Махум Zosym. — [Електронний ресурс]. — Режим доступу: <https://www.maxzosim.com/sequence-diagrams/>.
17. Berson A. Client/server architecture. 2nd ed. New York : McGraw-Hill, 1996. 569 p.
18. Беннет Л. Що таке функціональне програмування? Підручник із прикладом. Guru99. — [Електронний ресурс]. — Режим доступу: <https://www.guru99.com/uk/functional-programming-tutorial.html>.
19. GeeksforGeeks. Express.js Tutorial - GeeksforGeeks. GeeksforGeeks. — [Електронний ресурс]. — Режим доступу: <https://www.geeksforgeeks.org/express-js/>.

20. TypeScript for JavaScript Programmers. TypeScript. — [Электронный ресурс]. — Режим доступа: <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html>.
21. What is mysql and how does it work. Hostinger Tutorials. — [Электронный ресурс]. — Режим доступа: <https://www.hostinger.com/tutorials/what-is-mysql>.
22. De Luca G. Roulette selection in genetic algorithms. Baeldung.cs. — [Электронный ресурс]. — Режим доступа: <https://www.baeldung.com/cs/genetic-algorithms-roulette-selection>
23. GeeksforGeeks. Tournament Selection (GA) - GeeksforGeeks. GeeksforGeeks. — [Электронный ресурс]. — Режим доступа: <https://www.geeksforgeeks.org/tournament-selection-ga/>.
24. Genetic Algorithms Parent Selection. — [Электронный ресурс]. — Режим доступа: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm.
25. Datta S. Genetic algorithms: order one crossover. Baeldung.Cs. — [Электронный ресурс]. — Режим доступа: <https://www.baeldung.com/cs/ga-order-one-crossover>.
26. Two-Point and Three-Point Crosses. Thompson Rivers University. — [Электронный ресурс]. — Режим доступа: <https://opengenetics.pressbooks.tru.ca/chapter/mapping-with-three-point-crosses/>.

ДОДАТКИ

Функція preparePreScheduleRecord

```
function preparePreScheduleRecords(  
  studyStreams: IStudyStreamModal[],  
  studyStreamGroups: IStudyStreamGroupModal[],  
  subjects: ISubjectModal[],  
  subjectDetails: ISubjectDetailsModal[],  
  classrooms: IClassroomModal[],  
  studyStreamSubjects: IStudyStreamSubjectModal[],  
  teacherSubjects: ITeacherSubjectModal[]  
) : ISchedule[] {  
  const scheduleRecords: ISchedule[] = [];  
  
  studyStreams.forEach((stream) => {  
    // Отримуємо групи, які належать до поточного потоку  
    const streamGroups = studyStreamGroups.filter(  
      (sg) => sg.study_stream_id === stream.id  
    );  
  
    const streamSubjects = studyStreamSubjects.filter(  
      (s) => s.study_stream_id === stream.id  
    );  
  
    streamSubjects.forEach((streamSubject) => {  
      const subject = subjects.find((s) => s.id ===  
streamSubject.subject_id);  
  
      if (!subject) {  
        console.warn(`Subject with ID ${streamSubject.subject_id} not  
found.`);  
        return;  
      }  
  
      const details = subjectDetails.find(  
        (d) => d.subject_name_id === subject.id  
      );  
  
      if (!details) {  
        console.warn(`Details for subject ID ${subject.id} not found.`);  
        return;  
      }  
    });  
  });  
}
```

```

    }

    const availableTeachers = teacherSubjects.filter(
      (t) => t.subject_id === subject.id
    );

    if (availableTeachers.length === 0) {
      console.warn(`No teachers found for subject ID ${subject.id}.`);
      return;
    }

    const suitableClassrooms = classrooms.filter(
      (c) => c.classroom_type_id === details.classroom_type_id
    );

    if (suitableClassrooms.length === 0) {
      console.warn(`No classrooms found for subject ID ${subject.id}.`);
      return;
    }

    // Створюємо записи для кожної групи, пов'язаної з потоком
    streamGroups.forEach((streamGroup) => {
      availableTeachers.forEach((teacher) => {
        suitableClassrooms.forEach((classroom) => {
          scheduleRecords.push({
            teacher_id: teacher.teacher_id as number,
            subject_id: subject.id as number,
            group_id: streamGroup.student_group_id as number,
            classroom_id: classroom.id as number,
          });
        });
      });
    });

    return scheduleRecords;
  }

```

Функція combineTimeSlotsWithSchedule

```
function combineTimeSlotsWithSchedule(  
  timeSlots: ITimeSlotModal[],  
  scheduleRecords: ISchedule[]  
): ISchedule[] {  
  const combinedSchedule: ISchedule[] = [];  
  
  scheduleRecords.forEach((schedule) => {  
    timeSlots.forEach((timeSlot) => {  
      combinedSchedule.push({  
        teacher_id: schedule.teacher_id,  
        subject_id: schedule.subject_id,  
        group_id: schedule.group_id,  
        classroom_id: schedule.classroom_id,  
        time_slot_id: timeSlot.id,  
      });  
    });  
  });  
  
  return combinedSchedule;  
}
```

Функція generateLimitedSchedule

```

function timeToHours(time: string): number {
  const [hours, minutes] = time.split(':').map(Number);
  return hours + minutes / 60;
}

function generateLimitedSchedule(
  studyStreams: IStudyStreamModal[],
  studyStreamSubjects: IStudyStreamSubjectModal[],
  studyStreamStGroups: IStudyStreamGroupModal[], // Додано для отримання
  group_id
  combinedSchedule: ISchedule[],
  lessonDuration: number // Тривалість уроку в хвиликах
): ISchedule[] {
  const result: ISchedule[] = [];

  studyStreams.forEach((stream) => {
    // Отримуємо всі групи, пов'язані з потоком
    const groupsForStream = studyStreamStGroups.filter(
      (stGroup) => stGroup.study_stream_id === stream.id
    );

    groupsForStream.forEach((group) => {
      const groupId = group.student_group_id;

      // Витягуємо предмети для поточного потоку
      const subjectsForStream = studyStreamSubjects.filter(
        (s) => s.study_stream_id === stream.id
      );

      subjectsForStream.forEach((subjectForStream) => {
        const lectureHours = timeToHours(
          (subjectForStream.lecture_hours as string) || '00:00:00'
        );
        const practicalHours = timeToHours(
          (subjectForStream.practical_hours as string) || '00:00:00'
        );
        const hours = lectureHours + practicalHours;
      });
    });
  });

  return result;
}

```

```

const lessonDurationHours = lessonDuration / 60;
const requiredLessons = Math.ceil(hours / lessonDurationHours);

// Витягуємо всі записи з combinedSchedule для групи та предмета
const eligibleLessons = combinedSchedule.filter(
  (lesson) =>
    lesson.group_id === groupId &&
    lesson.subject_id === subjectForStream.subject_id
);

const shuffledLessons = eligibleLessons.sort(() => Math.random() -
0.5);

for (
  let i = 0;
  i < requiredLessons && i < shuffledLessons.length;
  i++
) {
  result.push(shuffledLessons[i]);
}
});
});
});

return result;
}

```

Функція evaluateFitness

```

const PENALTY_HARD_CONSTRAINT = 1000;
const BONUS_SOFT_CONSTRAINT = 10;

function evaluateFitness(
  schedule: ISchedule[],
  studyStreamSubjects: IStudyStreamSubjectModal[],
  classrooms: IClassroomModal[],
  teachers: ITeacherModal[],
  studentGroups: IStudentGroupModal[],
  timeSlots: ITimeSlotModal[],
  lessonDuration: number,
  workHours: { start: string; end: string },
  avoidEarlyLate: boolean
): number {
  let fitness = 0;

  // Map subjects to their required hours
  const subjectHoursMap = new Map(
    studyStreamSubjects.map((subject) => [
      subject.subject_id,
      Number(subject.lecture_hours) + Number(subject.practical_hours),
    ])
  );

  // Hard constraints
  const timeConflicts = new Set();
  const teacherConflicts = new Set();
  const groupConflicts = new Set();
  const classroomConflicts = new Set();

  for (const record of schedule) {
    const { teacher_id, group_id, classroom_id, time_slot_id, subject_id }
    =
      record;

    // Check for time conflicts within groups, teachers, and classrooms
    const key = `${time_slot_id}`;
    if (!timeConflicts.has(key)) {
      timeConflicts.add(key);
    }
  }
}

```

```

    } else {
      fitness -= PENALTY_HARD_CONSTRAINT;
    }

    const teacherKey = `${teacher_id}-${time_slot_id}`;
    if (!teacherConflicts.has(teacherKey)) {
      teacherConflicts.add(teacherKey);
    } else {
      fitness -= PENALTY_HARD_CONSTRAINT;
    }

    const groupKey = `${group_id}-${time_slot_id}`;
    if (!groupConflicts.has(groupKey)) {
      groupConflicts.add(groupKey);
    } else {
      fitness -= PENALTY_HARD_CONSTRAINT;
    }

    const classroomKey = `${classroom_id}-${time_slot_id}`;
    if (!classroomConflicts.has(classroomKey)) {
      classroomConflicts.add(classroomKey);
    } else {
      fitness -= PENALTY_HARD_CONSTRAINT;
    }

    // Check if classroom capacity meets group size
    const classroom = classrooms.find((c) => c.id === classroom_id);
    const group = studentGroups.find((g) => g.id === group_id);
    if (classroom && group && classroom.capacity < group.student_amount) {
      fitness -= PENALTY_HARD_CONSTRAINT;
    }

    // Check if required hours for the subject are met
    if (subjectHoursMap.has(subject_id)) {
      const requiredHours = subjectHoursMap.get(subject_id) || 0;
      subjectHoursMap.set(
        subject_id,
        Math.max(0, requiredHours - lessonDuration)
      );
    }
  }
}

```

```

// Penalty for unmet subject hours
for (const [_, remainingHours] of subjectHoursMap) {
  if (remainingHours > 0) {
    fitness -= PENALTY_HARD_CONSTRAINT;
  }
}

// Soft constraints
for (const record of schedule) {
  const { teacher_id, group_id, time_slot_id } = record;

  // Find the corresponding time slot
  const timeSlot = timeSlots.find((slot) => slot.id === time_slot_id);
  if (timeSlot) {
    const slotHour = parseInt(
      (timeSlot.start_time as string).split(':')[0],
      10
    );
    const workStartHour = parseInt(workHours.start.split(':')[0], 10);
    const workEndHour = parseInt(workHours.end.split(':')[0], 10);

    if (
      avoidEarlyLate &&
      (slotHour < workStartHour || slotHour >= workEndHour)
    ) {
      fitness -= BONUS_SOFT_CONSTRAINT;
    } else {
      fitness += BONUS_SOFT_CONSTRAINT;
    }
  }
}

// Balance teacher workload
if (timeSlot) {
  const day = timeSlot.date;
  const teacherDayLoad = schedule.filter((s) => {
    const slot = timeSlots.find((ts) => ts.id === s.time_slot_id);
    return slot && slot.date === day && s.teacher_id === teacher_id;
  }).length;

  if (teacherDayLoad <= 5) {
    fitness += BONUS_SOFT_CONSTRAINT;
  } else {

```

```

    fitness -= BONUS_SOFT_CONSTRAINT;
  }
}

// Balance group schedule
const groupSchedule = schedule.filter((s) => s.group_id === group_id);
const uniqueDays = new Set(
  groupSchedule.map((s) => {
    const slot = timeSlots.find((slot) => slot.id === s.time_slot_id);
    return slot?.date;
  })
);
if (uniqueDays.size > 3) {
  fitness += BONUS_SOFT_CONSTRAINT;
}
}

return fitness;
}

```

Функція tournamentSelection

```
function tournamentSelection(  
  fitnessScores: number[],  
  populationSize: number,  
  tournamentSize: number  
): number[] {  
  const selectedParents: number[] = [];  
  
  while (selectedParents.length < populationSize) {  
    // Randomly select participants for the tournament  
    const tournamentParticipants = Array.from({ length: tournamentSize },  
    () =>  
      randomInt(0, fitnessScores.length)  
    );  
  
    // Find the participant with the highest fitness in the tournament  
    let bestParticipant = tournamentParticipants[0];  
    for (const participant of tournamentParticipants) {  
      if (fitnessScores[participant] > fitnessScores[bestParticipant]) {  
        bestParticipant = participant;  
      }  
    }  
  
    // Add the best participant to the selected parents  
    selectedParents.push(bestParticipant);  
  }  
  
  return selectedParents;  
}
```

Функція uniformCrossover

```
function uniformCrossover(  
  chromosomes: ISchedule[][],  
  selectedParentIndices: number[]  
) : ISchedule[][] {  
  const newGeneration: ISchedule[][] = [];  
  
  // Паруємо батьків для схрещування  
  for (let i = 0; i < selectedParentIndices.length; i += 2) {  
    if (i + 1 >= selectedParentIndices.length) {  
      // Якщо кількість батьків непарна, останній просто копіюється  
      newGeneration.push([...chromosomes[selectedParentIndices[i]]]);  
      continue;  
    }  
  
    const parent1 = chromosomes[selectedParentIndices[i]];  
    const parent2 = chromosomes[selectedParentIndices[i + 1]];  
    const offspring1: ISchedule[] = [];  
    const offspring2: ISchedule[] = [];  
  
    // Схрещування генів  
    for (let geneIndex = 0; geneIndex < parent1.length; geneIndex++) {  
      if (Math.random() < 0.5) {  
        offspring1.push(parent1[geneIndex]);  
        offspring2.push(parent2[geneIndex]);  
      } else {  
        offspring1.push(parent2[geneIndex]);  
        offspring2.push(parent1[geneIndex]);  
      }  
    }  
  
    newGeneration.push(offspring1, offspring2);  
  }  
  
  return newGeneration;  
}
```

Функція mutateChromosome

```

function mutateChromosome({
  chromosome,
  teachers,
  classrooms,
  timeSlots,
  teacherSubjects,
  subjectDetails,
  studentGroup,
}: MutationInput): ISchedule {
  const mutationType = Math.floor(Math.random() * 3); // 0: teacher, 1:
classroom, 2: timeSlot

  switch (mutationType) {
    case 0: {
      // Mutate teacher_id
      const subjectTeachers = teachers.filter((teacher) =>
        teacherSubjects.some(
          (teacherSubject) =>
            teacherSubject.subject_id === chromosome.subject_id &&
            teacherSubject.teacher_id === teacher.id
        )
      );
      if (subjectTeachers.length > 0) {
        const randomTeacher =
          subjectTeachers[Math.floor(Math.random() * subjectTeachers.length)]
;
        chromosome.teacher_id = randomTeacher.id;
      }
      break;
    }
    case 1: {
      // Mutate classroom_id
      const subjectClassroomDetails = subjectDetails.filter(
        (detail) => detail.subject_id === chromosome.subject_id
      );
      const group = studentGroup.find(
        (group) => group.id === chromosome.group_id
      );

```

```

const validClassrooms = classrooms.filter(
  (classroom) =>
    classroom.capacity >= (group as IStudentGroupModal)?.student_amount
  &&
    subjectClassroomDetails.some(
      (detail) => detail.classroom_type === classroom.classroom_type
    )
);
if (validClassrooms.length > 0) {
  const randomClassroom =
    validClassrooms[Math.floor(Math.random() * validClassrooms.length)]
;
  chromosome.classroom_id = randomClassroom.id;
}
break;
}
case 2: {
  // Mutate time_slot_id
  if (timeSlots.length > 0) {
    const randomTimeSlot =
      timeSlots[Math.floor(Math.random() * timeSlots.length)];
    chromosome.time_slot_id = randomTimeSlot.id;
  }
  break;
}
}

return chromosome;
}

```

Функція geneticAlgorithm

```

function geneticAlgorithm(
  population: ISchedule[][],
  teachers: ITeacherModal[],
  subjects: ISubjectModal[],
  classrooms: IClassroomModal[],
  studentGroups: IStudentGroupModal[],
  studyStreams: IStudyStreamModal[],
  studyStreamSubjects: IStudyStreamSubjectModal[],
  timeSlots: ITimeSlotModal[],
  teacherSubjects: ITeacherSubjectModal[],
  subjectDetails: ISubjectDetailsModal[],
  classDuration: number,
  startDayTime: DATE,
  endDayTime: DATE,
  POPULATION_SIZE: number,
  GENERATION_AMOUNT: number
): ISchedule[] {
  let genAmountVar = GENERATION_AMOUNT;
  const calculation = (population: ISchedule[][], generation: number) =>
  {
    let i = 0;
    const grades = population.map((chromosome) => {
      console.log('fitnes', i++);
      return evaluateFitness(
        chromosome,
        studyStreamSubjects,
        classrooms,
        teachers,
        studentGroups,
        timeSlots,
        classDuration,
        { start: startDayTime as string, end: endDayTime as string },
        true
      );
    });

    const biggestGrade = Math.max(...grades);
    if (generation === 0 || biggestGrade >= 0) {

```

```

        const index = grades.findIndex((value, index) => value ===
biggestGrade && index);
        return population[index];
    }

    const selectedParents = tournamentSelection(grades, POPULATION_SIZE,
4);
    const newGeneration = uniformCrossover(population, selectedParents);
    const mutatedGeneration = newGeneration.map((chromosome) =>
        mutateSchedule(
            chromosome,
            teachers,
            classrooms,
            timeSlots,
            teacherSubjects,
            subjectDetails,
            studentGroups
        )
    );

    return calculation(mutatedGeneration, --genAmountVar);
}

return calculation(population, genAmountVar);
}

```