МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки
Факультет Комп'ютерних наук
Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Дослідження методів генерації практичних навчальних завдань на основі різних рівнів складності

Виконав:
Студент 2 курсу групи ІПЗм-20-2
Малікін Д. Є.
(прізвище, ініціали)
Спеціальність 121 — Інженерія програмного забезпечення
Тип програми Освітньо-наукова
Керівник ст. викладач каф. ПІ, к.т.н. Кириченко І. В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. Кафедри _____ З. В. Дудар

2022 р.

Харківський національний університет радіоелектроніки

Факультет___Комп'ютерних наук_____

Кафедра___Програмної інженерії_____

Рівень вищої освіти_____другий (магістерський)_____

Спеціальність _____121 - Інженерія програмного забезпечення_____

Тип програми _____освітньо-наукова програма_____

Освітня програма _____Інженерія програмного забезпечення_____

ЗАТВЕРДЖУЮ:

Зав. Кафедри _____

«_____» _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Малікіну Дмитру Євгеновичу

(прізвище, ім'я, по батькові)

1. Тема роботи: «Дослідження методів генерації практичних навчальних завдань на основі різних рівнів складності»

затверджена наказом університету « 24 » березня 2022 р

2. Термін подання студентом роботи до екзаменаційної комісії « 10 » травня 2022 р.

3. Вихідні дані до роботи: *методи генерації практичних завдань, методи кластеризації, мова програмування Python, бібліотеки numpy, pydantic, scikit-learn, середовище розробки PyCharm, методичні вказівки, пояснювальна записка*

4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз предметної галузі і постановка задачі, опис запропонованого алгоритму, реалізація програмної системи, порівняння методів кластеризації у розподілі завдань за рівнями складності*

КАЛЕНДАРНИЙ ПЛАН

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналітичний огляд	24.01.2022	виконано
2	Аналіз існуючих методів і алгоритмів	03.02.2022	виконано
3	Постановка задачі	23.02.2022	виконано
4	Вибір методів створення алгоритму	16.02.2022	виконано
5	Реалізація прототипу алгоритму	23.02.2022	виконано
6	Проведення експерименту та аналіз результатів	13.03.2022	виконано
7	Підготовка пояснювальної записки	13.04.2022	виконано
8	Перевірка роботи на антиплагіат	10.05.2022	виконано
9	Нормоконтроль	11.05.2022	виконано
10	Рецензування	13.05.2022	виконано
11	Підготовка презентації та доповіді	13.05.2022	виконано
12	Попередній захист	14.05.2022	виконано
13	Занесення роботи в електронний архів	14.05.2022	виконано
14	Допуск до захисту у зав. кафедри	19.05.2022	виконано

Дата видачі завдання «24» січня 202_ р.

Студент. _____
(підпис)

Керівник _____ _ст. викладач каф. ПІ, к.т.н. Кириченко І. В._
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 96 с., 6 рис., 7 табл., 19 джер., 12 додатків.

E-LEARNING, MBOK, BIG DATA, КЛАСТЕРИЗАЦІЯ, ГЕНЕРАЦІЯ, РІВЕНЬ СКЛАДНОСТІ, АЛГОРИТМ

Об’єктом дослідження є вплив динамічних практичних завдань з різним рівнем складності на результат засвоєння знань. Метою роботи є дослідження методів та алгоритмів генерації завдань на базі різних рівнів складності.

Методи дослідження базуються на обчисленні метрик, оцінці складності практичних завдань та порівнянні методів кластеризації отриманих наборів завдань для їх розподілу за рівнями складності.

У результаті роботи запропоновано алгоритм, що створює набори завдань, використовуючи кластеризацію для їх розподілу за рівнями складності. Також алгоритм може бути модифікований у подальшому для використання у різних сферах навчання.

E-LEARNING, MOOC, BIG DATA, CLUSTERING, GENERATION, DIFFICULTY LEVEL, ALGORITHM

The object of research is the impact of dynamic practical tasks with different difficulty levels on the result of knowledge acquisition.

The aim of the practice is to study methods and algorithms for generating tasks based on different difficulty levels.

Research methods are based on metrics calculation, scoring the difficulty of practical tasks and comparison of clustering methods for dividing task sets into difficulty levels.

The result of research is the proposed algorithm that creates sets of tasks, using clustering for their distribution by levels of complexity. Also, the algorithm can be further modified for the use in various education areas.

Я, Малікін Дмитро Євгенович, студент групи ІПЗм-20-2, здобувач вищої освіти на другому (магістерському) рівні, кафедра «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів генерації практичних навчальних завдань на основі різних рівнів складності», що буде представлена до ЕК для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Знання – це те, що люди здавна бажають безперестанно отримувати. Наука і технології з часом досягли значного рівня розвитку. Та не всім людям необхідно отримувати ці знання лише на професійному рівні, закінчуючи спеціальні заклади. А й навіть тим, хто отримує вищу освіту, іноді варто освіжити накопичений багаж знань.

Враховуючи такі проблеми, що постали перед значною часткою людства, були створені перші платформи для онлайн-навчання. У певних випадках така система повністю замінює очне навчання, а деякі сертифікати про закінчення онлайн-курсів стають важливим досягненням для пошуку роботи.

Велика частина існуючих онлайн-курсів містить лише теоретичну складову, тобто людина читає чи слухає лекції. Для певних галузей цього може бути достатньо, але для більшості спеціалізацій навчання слід вміщувати також і практичну складову. На сьогодні дистанційне навчання є актуальною сферою наукових досліджень. На відміну від традиційної освіти з учителем у аудиторії, онлайн-навчання пропонує переваги можливостей засвоєння знань у своєму власному темпі та постійного доступу до ресурсів знань.

Крім традиційної освіти, досить популярна також ніша для навчання на різних платформах МВОК (масових відкритих онлайн курсів), де люди можуть покращити свої знання та навички чи отримати їх у цікавих їм сферах. Незважаючи на те, що в сучасному світі люди мають достатньо можливостей для навчання, і навчальні заклади, і онлайн-платформи мають недолік, який виражається в певній сталій навчальній програмі. Для людей, які з тих чи інших причин можуть показати різний рівень засвоєння знань, передбачені практичні завдання одного рівня. Здебільшого це пов’язано з тим, що результатом навчання має бути відповідність учня певному рівню, який перевіряється комплексом іспитів, заліків тощо. Але це не вирішує проблеми розуміння окремих принципів, які ведуть до вирішення поставлених на іспиті завдань. Дозволяючи зрозуміти, як саме потрібно вирішувати завдання, а не просто знати, яка відповідь підходить для цього завдання, збільшує переваги від самого процесу навчання.

Зокрема, від того, які саме завдання отримані учнем, значного впливу зазнає його мотивація. Вона, у свою чергу, впливає на рівень ентузіазму, з яким учень буде розв’язувати завдання та наскільки ефективним буде засвоєння знань після таких завдань.

У даному дослідженні пропонується розглянути варіант навчання користувача з використанням набору практичних завдань, згенерованих з різними рівнями складності залежно від успішності виконання завдань користувача на попередніх рівнях. Таким чином, користувачі, які показують хороші результати у вирішенні завдань, отримають складніші завдання для підвищення кваліфікації. А користувачі, які мають певні труднощі з розумінням, будуть вирішувати простіші завдання та з більшою ймовірністю зможуть краще засвоїти теорію.

Такий підхід може принести користь переважно для систем МВОК. Онлайн-курси – гарне місце для тестування такого рішення, оскільки процес впровадження такого підходу в освітню систему може зайняти багато часу.

Таким чином, робота є актуальною як з практичної, так і з наукової точки зору, оскільки дозволяє розглянути застосування динамічної генерації завдань для навчальних програм, таким чином полегшуючи роботу для тих людей, що цим займаються. Зокрема, може бути досліджена достатньо велика кількість програмних методів як для власне генерації завдань так і застосування даного механізму у існуючих навчальних системах.

Метою дослідження є вирішення науково-практичної задачі аналізу та оцінки ефективності методів для динамічного створення практичних завдань з використанням різних рівнів складності. Для досягнення поставленої мети необхідно виконати такі завдання:

– проаналізувати існуючі методи та алгоритми генерації завдань динамічної складності;

– виконати теоретичний опис власного алгоритму для створення практичних завдань;

– реалізувати програмно даний алгоритм;

– створити тестовий приклад завдання та перевірити працездатність алгоритму при генерації наборів параметрів для цього завдання;

– проаналізувати працездатність алгоритму, визначити його недоліки та подальші кроки для покращення.

Об’єктом дослідження є вплив складності практичних завдань, поставлених перед учнем, на його мотивацію та ефективність навчання з використанням таких завдань.

Предмет дослідження – методи генерації практичних завдань на основі різних рівнів складності.

Методами дослідження є оцінка складності завдання з вимірами часу та пам’яті, а також порівняння методів кластеризації, що використовуються для розподілу завдань між рівнями за складністю.

У результаті дослідження одержано алгоритм, що генерує завдання та розподіляє їх за рівнями складності за допомогою методів кластеризації. Подальшого розвитку отримали дослідження використання прогресу учня у навчанні для надання йому нових практичних завдань.

Результати дослідження можуть бути використані для впровадження у навчальні системи, а також окремо як дані для подальшого розвитку досліджень методів кластеризації чи застосування графів чи дерев прогресу учня. Подальший розвиток може отримати й сам алгоритм з дослідженням методів виявлення та обчислення метрик та оцінки складності.

Результати кваліфікаційної роботи були представлені на двох наукових конференціях, тези доповіді [1] та стаття [2] опубліковані у збірниках цих конференцій:

– Дванадцята міжнародна науково-технічна конференція «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління»;

– 6th International Conference on Computational Linguistics and Intelligent Systems (Scopus) May 12–13, 2022, Gliwice, Poland.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЦІЛЕЙ ДОСЛІДЖЕННЯ

Однією з основних проблем, чому виконання практичних завдань онлайн-курсу в їх звичному розумінні може бути неефективним, є те, що набір завдань є статичним. Він не змінюється залежно від того, хто його виконує. Усі учасники курсу вирішують однаковий набір завдань. Здебільшого самі курси можуть містити лише одне завдання на розділ. Тому, щоб прочитати теоретичний матеріал і завершити практику, хтось може витратити 5 хвилин, а іншим доведеться кілька разів повертатися до початку, щоб нарешті розв’язати завдання. На платформах масових відкритих онлайн-курсів (МВОК) різноманітність студентів дуже висока, оскільки будь-хто може отримати доступ до потрібного курсу. Тому при складанні комплексу практичних завдань слід враховувати індивідуальність учнів та його конкретні успіхи й прогрес в навчанні.

Якщо учням поставлені нерутинні завдання на належному рівні, їхні когнітивні навички можна значно покращити. Наявність рівня складності завдання, що відповідає прогресу учня у розв’язанні завдань такого типу, може допомогти у покращенні їх мотивації. Учень, який знає, як розв’язувати задачу на низькому рівні, швидше за все, зможе розв’язувати подібне завдання на вищому рівні і з більшою ймовірністю буде зацікавлений у самому процесі такого розв’язування. Навпаки, якщо студент стикається з занадто складною проблемою, це може вплинути на його мотивацію і, отже, призвести до зниження ефективності процесу навчання [3].

Важливим аспектом завдань, які можуть вплинути на мотивацію студентів до взаємодії з ними, є їх зміст. Персонально релевантний зміст завдання може викликати ситуативний інтерес, що, у свою чергу, може призвести до посилення уваги, постійних зусиль і насолоди і в кінцевому підсумку призведе до кращого навчання. Те ж саме стосується конкретного рівня складності завдання. Складність проблеми учнем сприймається особисто і тому є релевантною [4].

Мотивація студентів є невід'ємним елементом високоякісної освіти [5]. Умотивований студент має більше шансів розв’язувати завдання з високим рівнем

ентузіазму. Для підвищення мотивації учнів до виконання завдань слід враховувати різні рівні складності цих завдань. Дослідники показали, що складність завдання може впливати на саморегуляцію учнів. Труднощі можуть також вплинути на стратегію та тактику вирішення задач, поставлених перед учнями. Занадто легкі чи навпаки складні проблеми давали б набагато менше користі від їх розв'язування [6].

Методи штучного інтелекту (ШІ) є потужним інструментом для підтримки освітян-практиків та інноваційних технологій [7]. Але все ж обмін знаннями між викладачами є однією з головних умов успішного вдосконалення процесів навчання. Тому поєднання інструментів штучного інтелекту та обміну знаннями між освітянами може підвищити поточний рівень ефективності навчання саме в МООС.

Серед існуючих варіантів реалізації такого підходу до створення навчальних завдань слід виділити алгоритм графу знань. Алгоритм розбиває знання на кілька груп: готові до вивчення, неготові до вивчення та вже засвоєні. Такий підхід генерує завдання на основі успішності виконання або необхідних для цього виконання навичок учня [8].

Підсумовуючи вищезазначені твердження, висока мотивація студентів є однією з головних умов успішного навчання. На мотивацію значно впливає зміст і рівень персоналізації завдань. Крім того, чим більше динаміки у зміні рівня складності завдань, тим більше ентузіазму проявляє студент під час їх вирішення.

Різні методи штучного інтелекту вже використовуються для покращення процесу освіти, що демонструє хороший рівень застосування в цій конкретній сфері. Існують методи генерування завдань на основі раніше розв'язаних завдань і загального досвіду та знань учнів у межах курсу.

Сучасні технології штучного інтелекту та машинного навчання (МН) мають великий вплив на життя кожного. Оскільки освіта є однією з головних потреб людей, впровадження методів ШІ та МН призведе до значного покращення освітнього процесу.

Таким чином, метою даного дослідження можна визначити спробу отримати алгоритм генерації практичних завдань на основі різного рівня складності та поточного прогресу учнів. При цьому, слід розуміти, що алгоритм повинен мати здатність із застосуванням малої кількості модифікацій бути пристосованим до використання у різноманітних сферах навчання.

У рамках дослідження необхідно вирішити такі задачі:

– розглянути існуючі методи та алгоритми генерації завдань динамічної складності, проаналізувати їх;

– розробити власний алгоритм для створення практичних завдань, описати його математичну модель;

– реалізувати програмно запропонований алгоритм;

– провести експеримент з генерацією набору завдань для тестового прикладу;

– проаналізувати результати експерименту, визначити недоліки алгоритму та подальші кроки для покращення.

Дослідження не передбачає, як саме запропонований алгоритм буде інтегрований у системи навчання, та платформи МВОК зокрема. Налаштування алгоритму та навчання певних моделей при використанні на конкретних платформах та у конкретних областях навчання є окремою зоною відповідальності учителів та власників курсів.

2 МАТЕМАТИЧНА МОДЕЛЬ АЛГОРИТМУ

2.1 Загальний опис алгоритму

Деякий алгоритм G повинен мати можливість генерувати набори завдань із певним рівнем складності, враховуючи цей рівень як частину вхідних даних.

Припустимо, що є певне завдання T з параметрами $A = \{a_1, .., a_n\}$. Параметри можуть набувати різних форм. Розглядаючи такі завдання у контексті математичних прикладів, можна припустити, що T співвідноситься з певним завдання, наприклад, квадратичним рівнянням, що має бути розв'язаним. Відповідно, існує множина $x = \{x_1, .., x_n\}$, що є розв'язкою завдання. На даному етапі можна опустити можливі випадки, коли, наприклад, корені рівняння відсутні.

У рівнянні (1) множина A буде містити $\{a, b, c\}$.

$$ax^2 + bx + c = 0. \quad (1)$$

Тоді, можна стверджувати, що в залежності від різних значень A буде змінюватись його складність T. У термінології інформаційних технологій, складність алгоритму не зміниться. Якщо алгоритм виконує, наприклад, n операцій над вхідними даними розміром n, то і складність залишається лінійною.

Проте, для людини відносно простіше виконати обчислення над 2-розрядними числами, ніж з 3-розрядними. Саме у цьому розумінні описується складність завдання в даній роботі.

Маючи завдання T та параметри A, також необхідно визначити алгоритм F такий, що:

$$F\big(T(A)\big) = x, \quad (2)$$

тобто алгоритм для розв'язування завдання.

Також існують такі типи завдань, коли у межах одного завдання треба надати відповідь на декілька запитань. Тоді на вхід також необхідно подавати множину

алгоритмів $F = \{F_1, ..., F_n\}$, кожен з яких приводить до знаходження розв'язку $X = \{X_1, .., X_n\}$.

Нехай l – це рівень складності, що подається як вхідний для запропонованого алгоритму G. Тоді даний алгоритм може бути сформульований так:

$$G(l, T, F) = \{A_l, X\}. \quad (3)$$

Звичайно, алгоритми розв'язування більшості сучасних математичних прикладів давно визначені, тому у автора курсу не має потреби описувати алгоритм розв'язування, наприклад, квадратного рівняння. Система, що реалізує цей алгоритм, має бути здатна вирішувати основні відомі задачі (не тільки в математиці) без їх явного визначення.

Таким чином, пропонується розробити алгоритм, який приймає на вхід T, F і l, та генерує певну множину A, що належить до l рівня складності задачі T.

Слід зазначити, що для запропонованого алгоритму необхідно визначити верхню та нижню межі для кожного значення A_n, щоб мати можливість зробити чіткий розподіл між рівнями складності задачі.

Інший спосіб – визначити константні значення для a_n. Можливий варіант з розширенням алгоритму для генерування нескінченної кількості рівнів складності, але в даній роботі він не розглядається.

Крім того, для X можна додати конкретні обмеження. Прикладом такого випадку є те, що завдання з рівнянням має обов'язково мати розв'язок із дійсними числами.

Після отримання A_{max} і A_{min} необхідно створити набори A в цих межах. Ці набори можна визначити як набори, що містять максимальне та мінімальне значення A_n відповідно, або їх можна знайти динамічно, оцінюючи кожен набір.

Припустимо, що кількість рівнів складності дорівнює L. У простішому варіанті достатньо знайти крок арифметичної прогресії для кожного члена множини A - a_{nmin}, …, a_{nmax} з кількістю членів L.

Представимо запис функції оцінки S складності набору згенерованих параметрів. Функція має застосовуватись до наборів A_{max} та A_{min}:

$$S_{min} = S(A_{min});\ S_{max} = S(A_{max}). \quad (4)$$

Таким чином, усі набори A будуть мати значення оціненої складності у такому проміжку:

$$S(A) \in [S_{min}, S_{max}]. \quad (5)$$

Постає питання, як саме функція S має обчислювати складність набору A.

2.2 Обчислення складності набору параметрів

Оскільки обчислювальна здатність апаратного забезпечення може значно відрізнятися, а час виконання $F(T(A_{max}))$ цілком можливо буде майже однаковий з часом виконання $F(T(A_{min}))$, то потрібно розглядати дещо інший підхід до обчислення складності завдання.

Якщо час виконання над граничними наборами є однаковим, можна виконувати зміри часу декількох запусків з одними вхідними даними. Процес може бути автоматизований у залежності від отриманої різниці між часом виконання F_{max} та F_{min}. Якщо ця різниця менше певного значення ε, тоді кількість запусків збільшується. Фактична кількість виконань алгоритму має бути визначена експериментально.

Іншою метрикою для оцінки завдання може стати кількість виділеної пам’яті процесом під час виконання. Хоча, на прикладі з квадратичним рівнянням, кількість пам’яті буде значно малою, для певних типів завдань така метрика може бути дуже корисною у процесі оцінки.

Отже, дві базові метрики, які можна розглядати у якості прикладу для оцінки складності набору A:

– t – час, витрачений на виконання алгоритму над згенерованим набором.

– m – виділена пам’ять для вирішення завдання.

Тоді, S_{min} може бути обчислене так:

$$S_{min} = t_{min} * w_t + m_{min} * w_m. \quad (6)$$

Відповідно, S_{max} обчислюється так:

$$S_{max} = t_{max} * w_t + m_{max} * w_m, \quad (7)$$

де w_t , w_m – це ваги відповідних метрик, що визначають, як саме кожна з метрик впливає на отриману оцінку.

Ваги також можуть бути знайдені експериментально при застосуванні у конкретному випадку.

Звісно, інші метрики можуть бути використані у залежності від завдання T, тож насправді функція оцінки визначена сума добутків значення кожної метрики на відповідну вагу:

$$S = \sum_{i=1}^{n} M_i * w_i, \quad (8)$$

де M_i (i = 1, .., n) це послідовність метрик, та w_i – це відповідні їм ваги.

2.3 Розподіл на рівні складності

Найпростішим рішенням поставленої задачі є надання можливості розробнику курсу визначити конкретні межі кожного A_l. Це спрацює для певних випадків використання, коли задача T є досить простою, для якої людина може легко визначити кілька рівнів складності. Оскільки це не зрозуміло для всіх проблем, слід розглядати інший підхід до розділення наборів завдань. Один з таких підходів – це кластеризація.

Кластеризація – це метод групування набору об’єктів за їхньою схожістю. Тоді об'єкти в одному кластері повинні бути більш схожими за деякими ознаками [9].

Спочатку слід створити набори A. Їх можна знайти як певні комбінації значень a_1, .. a_n, враховуючи, що a_n можна визначити як:

– значення з проміжку a_{nmin}, .., a_{nmax}

– конкретне значення зі сталої множини параметрів.

Результуючі набори A можуть бути розглянуті як вектори значень. Такі вектори мають бути нормалізовані. Тим же способом має бути нормалізованим також значення оцінки складності. Нормалізований вектор $\hat{A}$ для вектору $\vec{A}$ може бути знайдений так:

$$\hat{A} = \frac{\vec{A}}{\|\vec{A}\|}, \tag{9}$$

де $\|\vec{A}\|$ це довжина вектору $\vec{A}$.

Після процесу нормалізації, вектори можуть бути використані як вхідні дані для певного алгоритму кластеризації. Нехай кількість кластерів дорівнює L – кількості рівні складності. У якості функції дистанції $D(A_1, A_2)$ між двома векторами A_1 та A_2 використовується функція, що порівнює відповідні $S(A_1)$ так $S(A_2)$. Як результат даного процесу, набори параметрів мають бути розподілені між L кластерами та певна модель кластеризації буде навчена робити рішення, до якого кластеру будь-який новий набір A має бути розподілений.

Отже, опишемо узагальнено повний алгоритм G.

1. На вхід подаються функція F, розв’язок X та кількість рівнів складності L.

2. Значення a_n обмежуються границями a_{nmin} та a_{nmax}.

3. Генеруються набори A, а також, якщо зазначено, A_{min} та A_{max}.

4. Фільтруються набори A так, щоб лишились ті, що підходять для певних розв’язків X.

5. Знаходяться оцінки S_{min}, S_{max}.

6. Обчислюються оцінки усіх інших наборів.

7. Нормалізуються вектори *A*.

8. Нормалізовані вектори передаються на вхід алгоритму кластеризації.

9. За потреби генерується новий набір *A*, та з використанням моделі кластеризації, призначається до певного кластеру *l*.

10. За потреби отримується рівень складності *l* у якості вхідного значення та генерується новий набір *A*, що може бути призначений до даного кластеру.

Для реалізації даного алгоритму експериментально мають бути визначені метрики та відповідні їх ваги. Також необхідно визначити, який саме алгоритм кластеризації буде показувати кращий результат у залежності від поставленої перед алгоритмом мети.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ

Для реалізації системи необхідна наявність двох основних складових – безпосередньо генератору наборів завдань та моделі кластеризації наборів для їх поділу на рівні складності. Генератор має реалізовувати основні етапи для створення наборів завдань. Він приймає на вхід сам алгоритм для розв’язання завдання, обмеження до вхідних параметрів та отриманого результату. Дані обмеження мають застосовуватися у процесі валідації отриманих значень. Модель кластеризації отримує на вхід параметри, які будуть використані у процесі поділу наборів завдань. На діаграмі послідовності представлено, як саме буде відбуватись взаємодія між компонентами системи (рисунок 3.1).

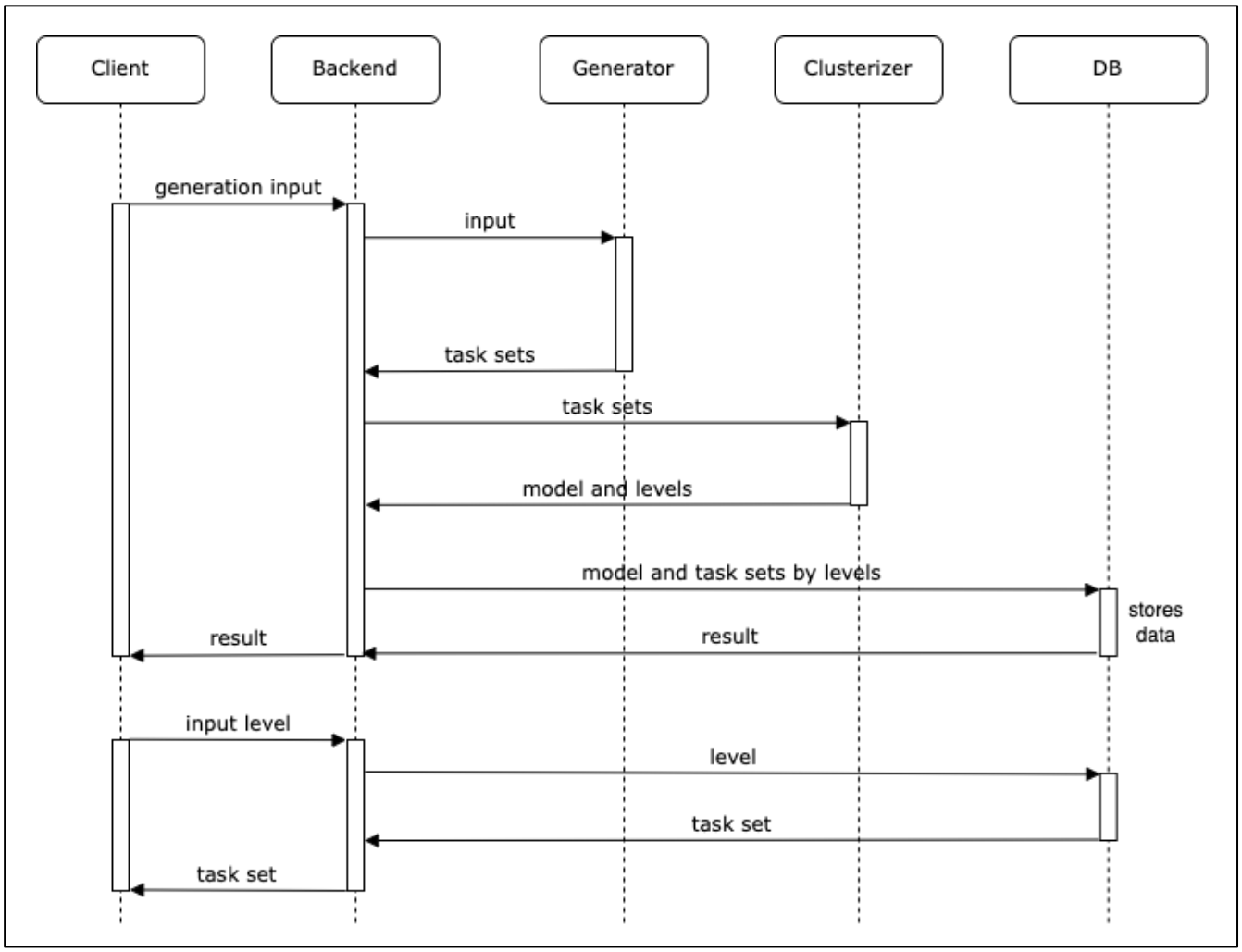

Рисунок 3.1 – Діаграма послідовності

Отримані набори завдань, а також їх належність до певного рівня складності потім зберігається у базі даних для подальшого повернення у систему навчання. Навчена модель також має зберігатися для подальшого використання чи донавчання. Таким чином, загалом система повинна мати певний єдиний сервіс для поєднання складових, до яких входять:

– генератор завдань;
– модель кластеризації;
– сховище даних.

Слід зазначити, що на діаграмі не виділено окремим компонентом сервіс для оцінки наборів завдань. У першій реалізації він є частиною власне генератора завдань, який під час їх створення одночасно виконує оцінку набору, оскільки може використовувати заданий алгоритм для перевірки часу та пам’яті, витрачених на виконання. У подальшому слід винести його окремо від генератора, оскільки він може використовуватись як самостійний сервіс з можливістю використання машинного навчання.

Також у тестовій реалізації не було використане сховище даних, а готові набори використовувались лише для візуалізації.

Для реалізації було обрано мову програмування Python, як одну з найбільш популярних та пристосованих до застосування у сфері машинного навчання та data science. Генерація завдань забезпечується на рівні бібліотек самої мови Python, а також бібліотек numpy [10], яка надає оптимізований інструментарій для структур даних та математичних розрахунків і pydantic [11] для підтримки валідації. Для забезпечення кластеризації було обрано бібліотеку scikit learn [12], що містить велику кількість вже реалізованих методів кластеризації. Перелік залежностей вказаний у додатку Л.

3.1 Реалізація генератору параметрів завдань

Як було зазначено вище, для генератора завдань було вирішено обмежитись лише стандартною бібліотекою python, окрім сторонньої бібліотеки numpy, яка

надає змогу використати оптимізовані алгоритми та структури даних для обчислень.

Загалом, генератор має реалізовувати наступні кроки алгоритму.

1. Отримати на вхід алгоритм розв’язання завдання, та параметри валідації для наборів завдань та результатів.

2. Створити набори завдань з комбінацій значень, границі яких задані у вхідних даних. Виконати валідацію наборів, відсіюючи ті, що їй не задовольняють.

3. Виконати алгоритм завдання на кожному з наборів даних, отримавши метрики для цих наборів.

4. Виконати валідацію результатів виконання алгоритму, відсіюючи ті набори, результати розв’язання яких не задовольняють задані умови.

5. Оцінити складність наборів даних, що пройшли валідацію, за вже отриманими метриками.

6. Повернути множину наборів завдань з відповідними їм результатам та оцінкам, як результат роботи.

Для проведення експерименту було реалізовано дещо спрощену систему, для якої завдання будуть створені програмно, а не отримуються як вхідне значення.

Був реалізований базовий клас, що має два абстрактні методи. Наведемо реалізацію даного класу:

```
class Task:
    @staticmethod
    @abc.abstractmethod
    def algorithm(data: TaskData) -> TaskResult:
        pass

    @staticmethod
    @abc.abstractmethod
    def generate(*args, **kwargs) -> List[TaskData]:
        pass
```

Метод generate має створювати набори завдань, що підходять під задані умови. Метод algorithm власне розв’язує завдання, повертаючи отриманий результат.

Додатково реалізовані базові класи для вхідних даних та результату виконання алгоритму, що не мають конкретного функціоналу, проте надаються для полегшення анотацій типів у прямих реалізаціях завдань:

```
class TupleModel(BaseModel):
    def tuple(self):
        return tuple(self.dict().values())

class TaskData(BaseModel): pass
class TaskResult(TupleModel): pass
```

Клас TupleModel тут представляє функціонал для спрощеного представлення даних у вигляді послідовності значень. Уже від нього успадковуються базові класи для вхідних даних TaskData та для результату TaskResult.

Також були реалізовані базові класи для окремих типів вхідних даних:

```
class GenerationInput(BaseModel): pass
class GenerationDataInput(TupleModel):
    __cache__ = {}

    @abc.abstractmethod
    def generate(self):
        pass
```

Серед конкретних реалізацій такого класу є NumberStepInput, призначений для створення ряду чисел, якому на вхід подається нижня та верхня границі, а також крок між числами:

```
class NumberInput(GenerationDataInput): pass
class NumberStepInput(NumberInput):
    min: float
    max: float
    step: float
    exclude: Optional[Tuple[float]] = tuple()

    def generate(self):
        key = self.tuple()

        value_in_cache = NumberStepInput.__cache__.get(key)
        if value_in_cache:
            return value_in_cache

        value = [
            element
```

```
            for element in np.arange(self.min, self.max + self.step,
self.step)
            if element not in self.exclude
        ]
        NumberStepInput.__cache__[key] = value
        return value
```

Клас містить певну оптимізацію, що дозволяє для ідентичних вхідних параметрів використовувати вже згенерований набір для уникнення повторного виконання генерації.

Валідація як вхідних параметрів завдання, так і результату виконується на рівні описаної моделі. Таким чином, генератор на верхньому рівні під час створення наборів завдань має можливість виявити помилку валідації та проігнорувати даний набір даних.

3.2 Реалізація сервісу оцінки складності параметрів

Для оцінки використовується реалізація з двома базовими метриками – за часом виконання та об’ємом використаної пам’яті. Допоміжний клас отримує на вхід набір вагів для кожної метрики:

```
class Scorer:
    def __init__(self, *weights):
        self.weights = weights
```

Далі клас виконує переданий алгоритм завдання на вхідних параметрах та виконує оцінку складності:

```
class Scorer:
    @staticmethod
    def score(self, algorithm, data):

        before = time.time()
        m_score = max(memory_usage((executor, (algorithm, data))))
        after = time.time()
        t_score = after - before
        return *data.tuple(), self.get_score(m_score, t_score)
```

Де метод get_score підсумовує значення метрик, помножені на відповідну вагу:

```
def get_score(self, *scores):
    score = 0

    for metric_score, weight in zip(scores, self.weights):
        score += float(metric_score) * weight
    return score
```

Функція executor, представлена у методі score, виконує декілька викликів алгоритму. Кількість викликів у подальшому має конфігуруватися:

```
RANGE = 10
def executor(algorithm, data):
    for _ in range(RANGE):
        algorithm(data)
```

Таким чином, у результаті оцінки кожного набору параметрів повертається власне параметри завдання, а також набір оцінок за кожною з метрик.

Слід зазначити, що використання збору метрик у даному випадку є тісно пов’язаним з реалізацією. У подальшому даний функціонал також має бути реалізований так, щоб метрики могли легко конфігуруватись.

3.3 Реалізація сервісу кластеризації завдань

Сервіс кластеризації приймає у свій конструктор модель кластеризації, кількість кластерів, назву для подальших візуалізацій, функцію score_getter, використання якої буде описано далі, а також додаткові параметри для моделі.

Після цього виконується ініціалізація моделі:

```
class Clusterizer:
    def __init__(
            self,
            name: str,
            model_cls: Any,
            n_clusters: int,
            *args,
            **kwargs,
    ):
        self.name = name
        self.n_clusters = n_clusters
        self.model = model_cls(n_clusters=n_clusters, *args,
**kwargs)
```

Даний клас у своєму методі clusterize приймає на вхід отримані набори завдань з їх оцінками, та виконує навчання моделі:

```
def clusterize(self, x: Sequence):
    y = self.model.fit_predict(x)
    result = self.process(x, y)
    return result
```

Метод process оброблює отримані кластери, сортуючи їх за середнім значенням оцінки, видаючи у результаті рівні складності відсортовані власне за складністю завдань у них:

```
def process(self, x: Sequence, y: Sequence) -> dict:
    result = defaultdict(list)
    actual = {}
    for x, y in zip(x, y):
        result[y].append(list(x))

    avgs = []
    ys = list(range(self.n_clusters))
    for i in ys:
        x = result[i]
        x_avg = self.avg(x)
        avgs.append(x_avg)

    sorted_avgs = list(sorted(avgs))
    for i, x_avg in enumerate(sorted_avgs):
        index = avgs.index(x_avg)
        actual[i] = result[index]

    return actual
```

Для знаходження середнього значення складності рівня використовується метод avg. Він викликає функцію score_getter на кожному наборі параметрів, яка, приймаючи на вхід цей набір параметрів, повертає оцінку складності:

```
def avg(self, x: Sequence) -> float:
    return sum([self.score_getter(el) for el in x]) / len(x)
```

Для перевірки системи були використані 3 алгоритми, які реалізовані у бібліотеці scikit learn: k-середніх, агломеративний та спектральний. Використання методів буде описаному у подальшому тексті.

3.4 Реалізація менеджеру генерації завдань

Менеджер генерації завдань – це єдиний механізм, що керує створенням наборів параметрів, їх валідацією, оцінкою та кластеризацією.

У конструкторі такий менеджер приймає усі необхідні параметри для запуску подальшого процесу, ініціює класи для оцінки та кластеризації:

```
class Manager:
    def __init__(
            self,
            task_cls: Type[Task],
            generator_input: Any,
            clustering_name: str,
            clustering_model: Any,
            levels: int,
            score_getter: Callable,
            clustering_args: tuple,
            clustering_kwargs: dict,
            scorer_weights: tuple
    ):

        self.task_cls = task_cls
        self.generator_input = generator_input
        self.clusterizer = Clusterizer(
            name=clustering_name,
            model_cls=clustering_model,
            n_clusters=levels,
            score_getter=score_getter,
            *clustering_args,
            **clustering_kwargs
        )

        self.scorer = Scorer(*scorer_weights)
```

У методі run даного класу виконується увесь процес генерації параметрів завдань та їх розподілу за рівнями складності:

```
def run(self):
    inputs = self.task_cls.generate(**self.generator_input)

    valid_data = []
    for data in inputs:
        result = self.task_cls.algorithm(data)
        if result is not None:
            valid_data.append(data)

    scores = [
        normalize(self.scorer.score(self.task_cls.algorithm, data))
        for data in valid_data
```

```
    ]

    levels = self.clusterizer.cluster(scores)
    return levels
```

Таким чином, менеджер спочатку генерує усі можливі набори параметрів, після чого знаходить розв’язки для цих наборів. Усі набори, для яких розв’язок задовольняє задані умови, оцінюються, після чого виконується їх розподіл за рівнями складності. Ці рівні і повертаються у результаті роботи менеджера.

3.5 Реалізація тестового прикладу

Для перевірки системи було реалізовано тестовий приклад із завданням, пов’язаним з розв’язанням квадратного рівняння за допомогою знаходження дискримінанта.

Для алгоритму, що розв’язує завдання мають бути передані три параметри рівняння (*a, b, c*), які використовуються у знаходженні коренів за наступною формулою:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{10}$$

У якості тестового прикладу для описаних можливостей алгоритму, для параметрів мають виконуватись такі умови: ціле число у проміжку від -10 до 10, а також *a* не дорівнює 0. Для результату ж умови такі: рівняння має мати хоча б один корінь, корені мають бути цілими числами.

Для того, щоб використати створені механізми, описані у попередніх пунктах, треба реалізувати додаткові класи – клас вхідних даних та клас результату виконання завдання. Для наведеного прикладу клас вхідних даних виглядає таким чином:

```
class Data(TaskData):
    a: int
    b: int
    c: int
```

Клас результату перевіряє, що корені є цілими числами, а також, що результат має хоча б один корінь. Реалізація класу виглядає таким чином:

```
class Result(TaskResult):
    x1: int
    x2: Optional[int]

    @root_validator
    def validate_values(cls, values):
        x1 = values.get('x1')
        if int(x1) != x1:
            raise ValidationError()

        x2 = values.get('x2')
        if x2 is not None and int(x2) != x2:
            raise ValidationError()

        return values
```

Клас завдання, який є нащадком класу Task, має реалізувати абстрактні методи. Метод generate виглядає таким чином:

```
@staticmethod
def generate(
        *,
        a: NumberInput,
        b: NumberInput,
        c: NumberInput,
) -> List[TaskData]:
    data_sets = product(*(
        param.generate()
        for param in (a, b, c)
    ))

    return [
        Data(**dict(zip(["a", "b", "c"], data_set)))
        for data_set in data_sets
    ]
```

Кінцева множина наборів завдань утворюється комбінаціями з заданих числових рядів.

Метод algorithm – це реалізація власне алгоритму розв’язання рівняння:

```
@staticmethod
@validated_task
```

```
def algorithm(data: Data) -> Result:
    if data.a == 0:
        return Result()

    d = data.b ** 2 - 4 * data.a * data.c
    if d < 0:
        return Result()

    d_sqrt = math.sqrt(d)
    x1, x2 = (
        (-data.b + sqrt_) / (2 * data.a) for sqrt_ in (d_sqrt, -
d_sqrt)
    )

    if d == 0:
        x2 = None

    return Result(x1=x1, x2=x2)
```

Декоратор validated_task, що використовується тут, призначений для виконання валідації над результатом виконання алгоритму, та фільтрації тих результатів, що не задовольняють заданим умовам. Його реалізація виглядає таким чином:

```
def validated_task(func):
    def wrapper(*args, **kwargs):
        try:
            return func(*args, **kwargs)

        except ValidationError as e:
            return None

    return wrapper
```

Приклад власне використання наведених реалізацій з урахуванням їх передачі до класу менеджера з використанням моделі кластеризації k-середніх є таким:

```
inputs = dict(
    a=IntegerStepInput(min=-10, max=10, step=1, exclude=(0,)),
    b=IntegerStepInput(min=-10, max=10, step=1),
    c=IntegerStepInput(min=-10, max=10, step=1),
)

manager = Manager(
    task_cls=QuadraticEquation,
    generator_input=inputs,
    clustering_name="K-Means",
```

```
        clustering_model=KMeans,
        levels=10,
        score_getter=itemgetter(3),
        clustering_args=(),
        clustering_kwargs={},
        scorer_weights=(0.3, 0.7)
    )

    result = manager.run()
```

У даному фрагменті наявна валідація параметрів завдань, що була описана раніше. Зокрема, це належність параметрів до проміжку цілих чисел від -10 до 10, а також те, що значення a не може дорівнювати нулю (оскільки квадратне рівняння перестає бути таким при даному значенні).

Менеджер отримує реалізований клас самого завдання та зазначені вище обмеження для вхідних параметрів. Також менеджеру передається модель для кластеризації, у даному випадку у якості прикладу зазначено методі k-середніх з бібліотеки scikit-learn.

Функція score_getter у даному випадку це екземпляр itemgetter. Під час створення даний механізм отримує певний ключ (у даному випадку – індекс елемента у послідовності). А під час виклику отримує саму послідовність, що містить набір вхідних параметрів та оцінку. Таким чином, виклик itemgetter(3)(sequence) повертає елемент послідовності sequence з індексом 3, за яким у даному випадку і знаходиться оцінка набору параметрів.

Отримання оцінки з послідовності виконано саме таким чином, оскільки реалізація з отриманням останнього елемента послідовності унеможливлює подальше використання декількох оцінок, що було описано у теоретичному огляді алгоритму.

Останніми параметрами менеджер отримує конфігурації моделі кластеризації та ваги для кожної з метрик, після чого виконує повний процес створення завдань та їх розподілу за складністю.

Повна реалізація модулів наведена у додатках Д–К.

3.6 Подальші модифікації та покращення системи

У попередніх пунктах описано реалізацію прототипу системи, що використовує запропонований алгоритм. Система на даному етапі не передбачає сховище даних для створених наборів завдань та рівнів складності, оскільки для експерименту та для візуалізації достатньо мати отриманий результат у пам’яті. Структура даних у цьому випадку застосування є досить неоднорідною, зокрема набори можуть складатися з різноманітних типів параметрів. Тому визначити можна лише приблизну структуру (рисунок 3.2).

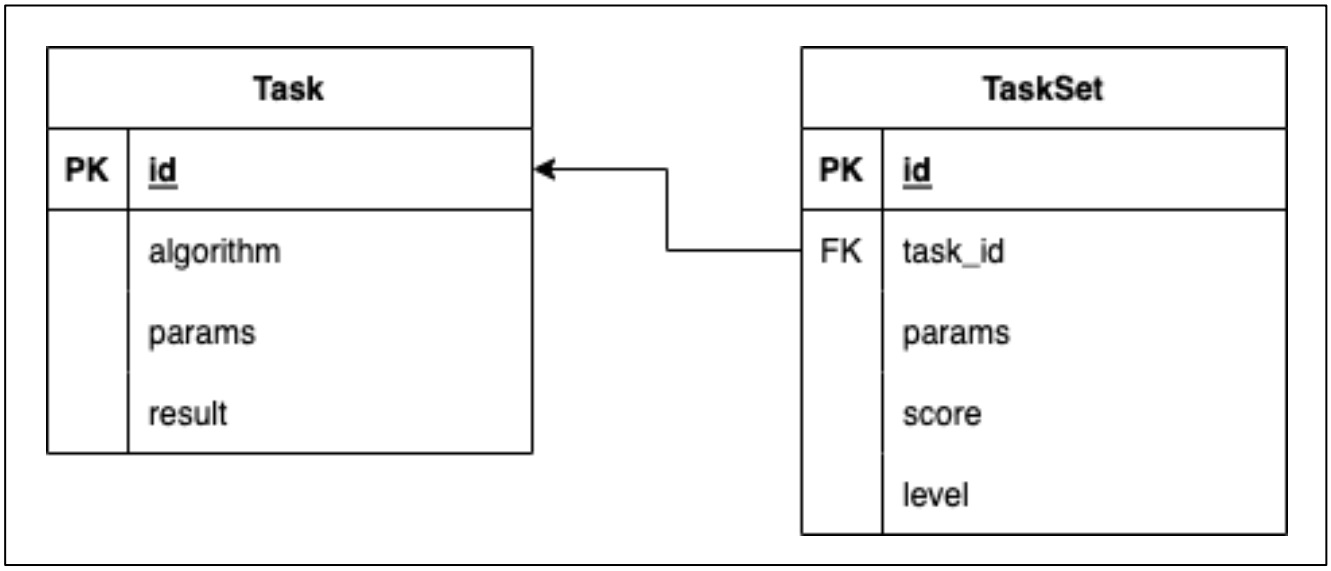

Рисунок 3.2 – Структура даних

У якості алгоритму у даній структурі можна зберігати посилання на файл, що містить його реалізацію. Поле Task.params – це певний опис того, які взагалі параметри можуть бути у завдання, зокрема границі можливих значень, тощо. На відміну від цього, поле TaskSet.params містить вже конкретні значення згенерованого набору завдань.

Через неоднорідність наборів параметрів, не лише за типами даних, а і за їх кількістю, можливим варіантом вибору для сховища даних є NoSQL системи [13]. Але, якщо представити набір параметрів, як досить просту послідовність даних, цілком можна використовувати і реляційні бази даних, зокрема, PostgreSQL [14], що надає можливості для зберігання таких неоднорідних структур у одному полі.

Для подальшої модифікації та покращення системи, у першу чергу, має бути реалізований механізм конфігурації та використання додаткових метрик, а також має бути створений сервіс для збереження та зчитування результату виконання алгоритму у сховищі. Далі ж можна розглянути модифікації сервісу надання оцінки із запровадженням функціоналу, що буде описаний у подальших розділах.

Також отриманий сервіс має у подальшому надавати певний інтерфейс, з яким будуть взаємодіяти клієнтські системи, передаючи усі необхідні вхідні дані, та отримуючи набори даних за певним рівнем складності у якості відповіді.

Опис інших теоретичних можливостей покращення програмної реалізації буде наведений у наступних розділах, враховуючи, зокрема, результати проведення експерименту.

4 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТУ

Для перевірки описаного алгоритму був створений прототип з квадратним рівнянням у якості прикладу завдання, що має бути розв'язане. У першу чергу має бути реалізований алгоритм, що розв'язує завдання. Квадратне рівняння може бути розв'язане декількома способами. Для прикладу було використано алгоритм із застосування дискримінанту. Формула для знаходження розв'язків рівняння виглядає таким чином:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{11}$$

Отже, набір параметрів *A* може бути описаний як (*a, b, c*).

Кроки проведення експерименту такі:

1. Реалізувати алгоритм за вказаною формулою.
2. Згенерувати набори A, що задовольняють певні умови.
3. Відфільтрувати набори, розв'язки для яких не задовольняють умови.
4. Оцінити кожен набір параметрів.
5. Розподілити набори за рівнями складності.
6. Оцінити працездатність алгоритму, порівняти різні методи кластеризації.

Отже, наступним кроком після реалізації алгоритму для розв'язання завдань є генерація наборів завдань. Для даного випадку це може бути виконано шляхом ітерації через послідовності, обмежені певними границями. Для експерименту, усі параметри мають знаходитись у проміжку [-10, 10], а також бути цілими числами. Зокрема, для параметру *a* є додаткова перевірка на те, що його значення не дорівнює нулю, оскільки таке рівняння не буде квадратним.

Для розв'язків рівнянь задані умови такі: рівняння має мати хоча б один розв'язок (він може бути нулем), і розв'язки мають бути також цілими числами. Таким чином, у результаті генерації та фільтрації було отримано 298 наборів параметрів.

Наступним кроком є знаходження оцінок для кожного з наборів параметрів. У рамках експерименту, були використані дві метрики – час виконання алгоритму та об’єм пам’яті, витрачений при цьому виконанні. За описаною у попередньому розділі формулою були знайдені оцінки для наборів з використанням ваги 0.3 та 0.7 для витрати пам’яті та часу відповідно.

Мінімальна та максимальна оцінка для створених наборів представлена у таблиці 4.1.

Таблиця 4.1 – Мінімальна та максимальна оцінка наборів параметрів

Набір параметрів	Оцінка
(-9, 0, 0)	4.765
(1, 5, 4)	5.321

Як можна бачити з таблиці, рівняння $9x^2 = 0$ оцінено алгоритмом, як таке, що можна розв’язати простіше у порівнянні з рівнянням $x^2 + 5x + 4 = 0$.

Далі необхідно виконати кластеризацію отриманих наборів даних. Для проведення експерименту були використані 3 методи: k-середніх, агломеративний та спектральний. Реалізація даних методів не є метою дослідження, тому були використані готові реалізації. Вимога до методу кластеризації – можливість власноруч визначити кількість кластерів. Методи порівнювались з базовими параметрами, передавалась лише бажана кількість кластерів. Для експерименту було обрано значення кількості кластерів, що дорівнює 10.

Після того, як отримані кластери, вони мають бути відсортовані за середнім значенням оцінки у межах кожного з них.

Перший алгоритм – це метод k-середніх, що є найбільш популярним на сьогодні. Його мета – розподілити вхідні дані на кластери, де кожне вхідне значення належить до кластеру з найближчим середнім значенням.

Кількість наборів параметрів у кожному з кластерів після виконання даного методу представлена на рисунку 4.1.

Як можна бачити з рисунку, кластери мають приблизно схожу кількість наборів.

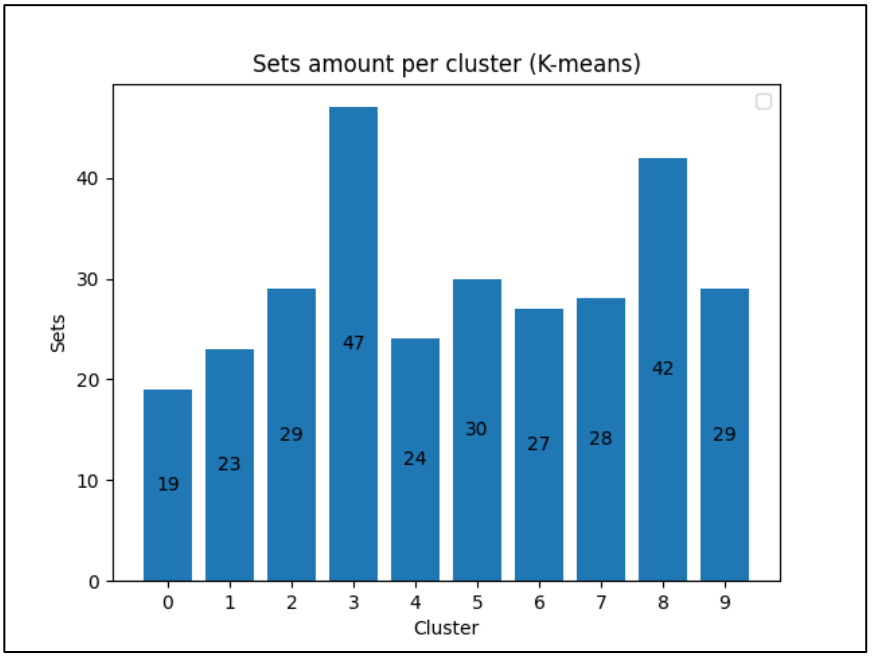

Рисунок 4.1 – Розміри кластерів після використання методу k-середніх

Приклади наборів у кластері 0 та 9 (найлегшому та найважчому рівнях складності за оцінкою алгоритму) представлені у таблиці 4.2. Середня оцінка для рівня 0 – це 4.782, для рівня 9 – 4.986.

Таблиця 4.2 – Приклади наборів після використання методу k-середніх

Рівень	Набір параметрів	Оцінка
0	(-9, 0, 0)	4.765
0	(-5, 0, 5)	4.798
0	(-7, 7, 0)	4.782
9	(-1, 3, 4)	4.842
9	(-3, 6, 9)	4.796
9	(1, 5, 4)	5.321

Дані приклади містять випадок, у якому набір параметрів з рівня 9 має нижчу оцінку, ніж набори з рівня 0. Це можливо, оскільки модель кластеризації отримує на вхід не лише оцінку, а і самі параметри. Таким чином, враховується схожість не лише за оцінкою, а і за вмістом набору.

Рівень 0 також показує, що набори, що містять нулі, мають нижчу оцінку, що відображає, які рівняння є порівняно легшими для розв'язування для людини.

Наступний метод кластеризації – агломеративний, який є підкласом ієрархічної кластеризації. Такі алгоритми намагаються побудувати ієрархію кластерів. Кожне вхідне значення у даному методів на початку знаходиться у власному кластері. У подальшому пари кластерів поєднуються, піднімаючись за ієрархією.

Кількість наборів, розподілених агломеративним методом, відображена на рисунку 4.2. Кількість наборів є також порівняно однаковою у різних кластерах.

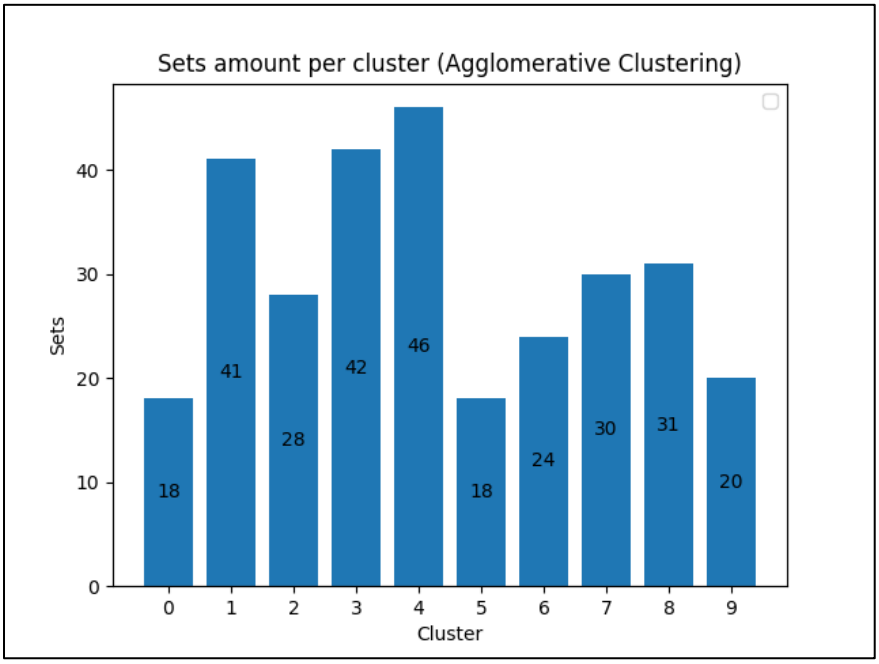

Рисунок 4.2 – Розміри кластерів після використання агломеративного методу

У таблиці 4.3 відображені приклади параметри з рівнів 0 та 9. Середня оцінка для рівня 0 дорівнює 4.802, а для рівня 9 – 4.837.

Результати показують, що різні рівні містять приблизно схожі набори параметрів, що свідчить про те, що у даній конфігурації даний метод кластеризації не є придатним для представленого алгоритму. Проте, він може бути використаний при інших механізмах оцінки складності (або ж з іншими метриками, чи вагами для цих метрик).

Таблиця 4.3 – Приклади наборів після використання агломеративного методу

Рівень	Набір параметрів	Оцінка
0	(-3, -9, 0)	4.795
0	(-2, -6, 0)	4.813
0	(-3, -6, 0)	4.799
9	(2, 8, 0)	4.842
9	(1, 8, 0)	4.835
9	(3, 9, 9)	4.835

Третій метод кластеризації, що був використаний у експерименті – це спектральний. Він використовує спектр матриці схожості даних для виконання зниження розмірності. Подальша кластеризація виконується у цих зменшених розмірностях. Результат застосування методу представлений на рисунку 4.3.

З рисунку можна помітити, що, у порівнянні з попередніми алгоритмами, розподіл наборів за кластерами відрізняється. Рівень 3 містить найбільшу кількість наборів параметрів, а у порівнянні з рівнем 4, ця кількість більша майже у два рази. Такий розподіл є дещо схожим на нормальний розподіл даних[15].

Даний випадок є корисним у створенні реальних рівнів складності завдань для навчання. З рисунку помітно, що середні рівні складності мають найбільшу кількість наборів параметрів, оскільки найбільше завдань будуть виконуватися саме на цих рівнях. Найпростіші ж чи найскладніші рівні складності будуть використовуватися значно рідше. Учні швидше будуть переходити до середніх рівнів складності та повільніше, або ж іноді й зовсім не переходити до високих рівнів складності.

Виділяється лише рівень 9, у якому кількість завдань збільшується, порівняно з поступовим зменшенням кількості завдань у попередніх рівнях. Проте, такі випадки можуть бути виправлені із застосування ручного налаштування рівнів на початкових етапах створення наборів завдань з вчителем.

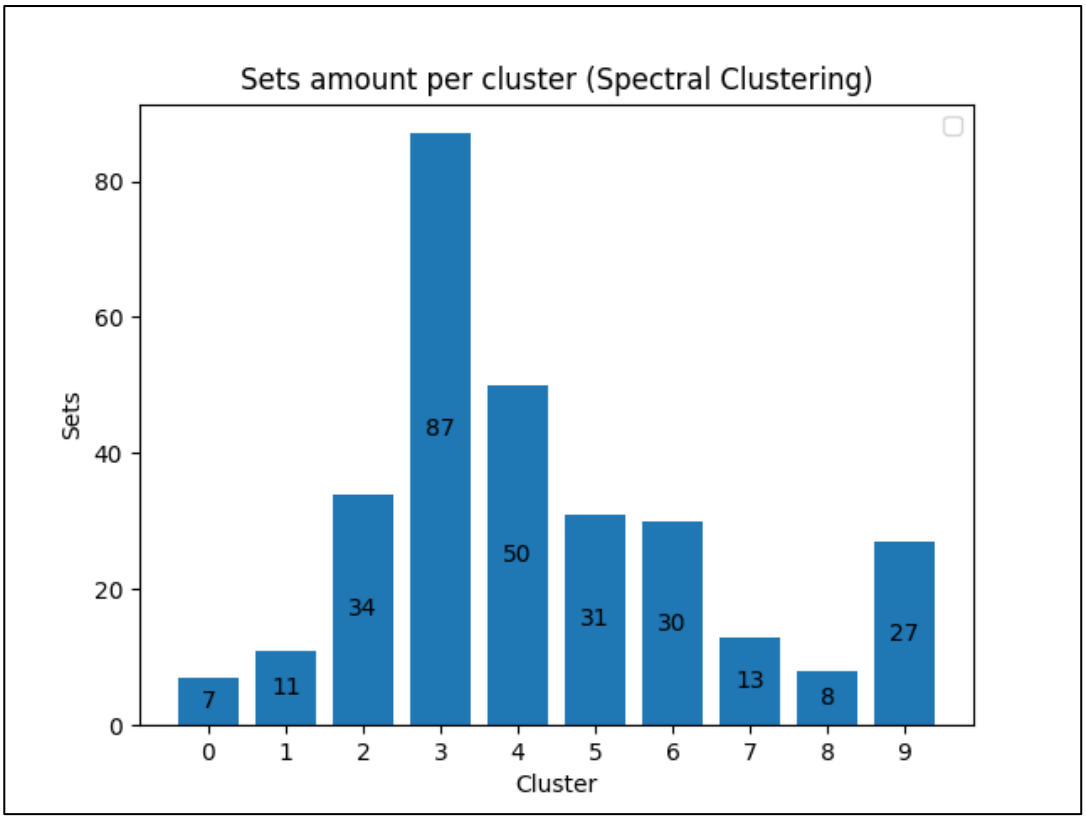

Рисунок 4.3 – Розміри кластерів після використання спектрального методу

Приклади наборів завдань після застосування спектральної кластеризації представлені у таблиці 4.4. Для кращого розуміння дана таблиця також містить набори з рівня 3. Середня оцінка рівня 0 дорівнює 4.776, рівня 9 – 4.840.

Таблиця 4.4 – Приклади наборів після використання спектрального методу

Рівень	Набір параметрів	Оцінка
1	2	3
0	(-4, 0, 0)	4.778
0	(-9, 0, 0)	4.765
0	(-10, 0, 0)	4.767

Кінець таблиці 4.4

Рівень	Набір параметрів	Оцінка
1	2	3
3	(1, -8, 0)	4.837
3	(-6, -6, 0)	4.78
3	(-1, -6, 0)	4.81
9	(-1, 6, 7)	4.802
9	(1, 6, 9)	4.864
9	(1, 6, 5)	4.854

З таблиці помітно, що рівень 0 містить набори, які мають переважно одне ненульове значення. У рівні 3 містяться набори з одним нульовим значенням. Набори ж з рівня 9 мають усі значення, відмінні від нуля. Це також відображає схожість з тим, як людина сприймає складність рівняння за кількістю значимих параметрів, та їх величиною.

5 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ ТА ПОДАЛЬШИЙ РОЗВИТОК

5.1 Аналіз результатів дослідження

У експерименті було проведено тестування запропонованого алгоритму на прикладі розв’язання квадратного рівняння з використання простої функції оцінки складності, що застосовувала метрики за використаним часом та об’ємом пам’яті при розв’язанні завдань. Три алгоритми кластеризації були використані для розподілу наборів параметрів за рівнями складності:

– метод k-середніх;
– агломеративний метод;
– спектральний метод.

Дані алгоритми були обрані, як найбільш застосовувані з тих, що отримують кількість кластерів, як вхідне значення.

Для порівняння, який з алгоритмів кластеризації показав себе краще у рамках експерименту, були використані 3 характеристики:

– кількість наборів у кожному кластері;
– чи є доцільним віднесення наборів до тих чи інших кластерів;
– недоліки алгоритму в цілому.

Методи k-середніх та агломеративний показали схожі результаті у кількісному розподілі наборів параметрів між кластерами. При цьому, агломеративний метод надав кластери, що не підходять для реального використання, оскільки набори параметрів розподілені за рівнями досить хаотично.

Спектральний метод, при цьому, показав найкращий результат у розподілі, як якісно, так і кількісно. Набори параметрів розподілені за їх актуальним змістом, таким чином, найскладніші рівні отримують значення, які відмінні від нуля. Рівняння з такими значеннями порівняно складніше розв’язувати, ніж рівняння з найлегшого рівня, де здебільшого, значущим є лише параметр *a.* При цьому, середні рівні отримали найбільшу кількість завдань. Саме завдання з цього рівні найчастіше будуть виконуватись, оскільки найлегший рівень буде проходить найшвидше. При цьому, менша кількість з учнів буде проходити до найвищого рівня складності.

Діаграма, що порівнює кількість наборів, розподілених за кластерами, відображена на рисунку 5.1.

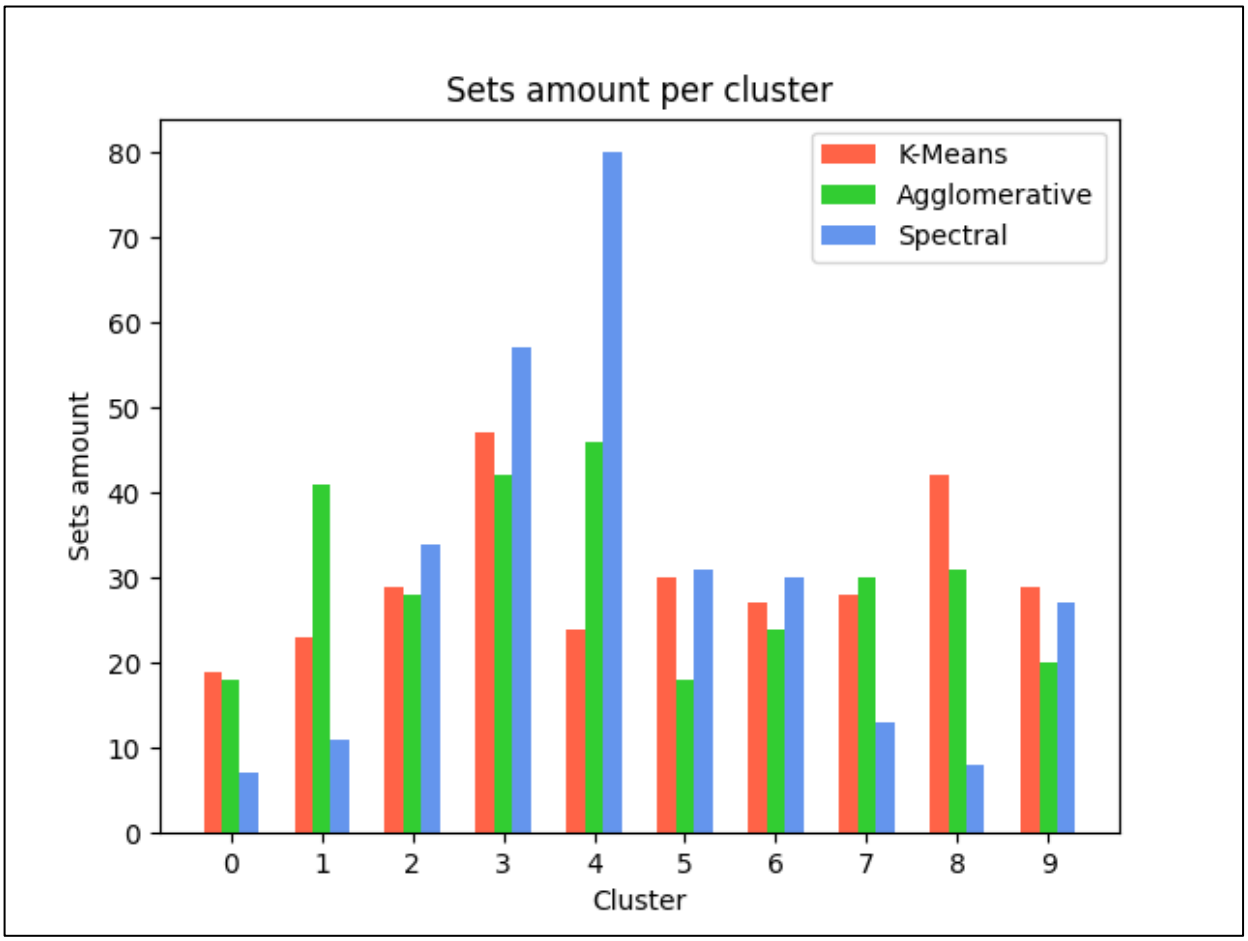

Рисунок 5.1 – Розміри кластерів після використання спектрального методу

Дана кількість також показана у чисельних значеннях у таблиці 5.1.

Таблиця 5.1 – Кількість наборів у кластерах

Рівень	K-середніх	Агломеративний	Спектральний
1	2	3	4
0	19	18	7
1	23	41	11
2	29	28	34
3	47	42	57
4	24	46	80

Кінець таблиці 5.1

Рівень	K-середніх	Агломеративний	Спектральний
1	2	3	4
5	30	18	31
6	27	24	30
7	28	30	13
8	42	31	8
9	29	20	27

Метод k-середніх має недолік, пов’язаний з проблематичною обробкою викидів та шумів. Проте, такі випадки не зустрілись під час виконання експерименту, окрім тих випадків, коли завдання з низькою оцінкою потрапляли до високих рівнів. Але це не є критично, оскільки дійсний вплив могли мати власне значення параметрів, через які і були отримані такі результати. При цьому, справжні аномалії не могли бути виявлені у даному експерименті, оскільки границі можливих значень для параметрів достатньо обмежені.

Недоліком агломеративного алгоритму є часова складність. При цьому, знову ж, у рамках експерименту, цей недолік не був виявлений, оскільки кількість згенерованих наборів була досить малою.

Такий же недолік має і спектральний алгоритм, здебільшого на великих наборах даних. Передбачається, що загалом кількість згенерованих наборів не має перевищувати декількох тисяч. А така кількість не є критичною для всіх представлених алгоритмів.

Загалом, найкраще у рамках експерименту себе показав спектральний алгоритм за кількісним та якісним розподілом згенерованих наборів параметрів. Недоліки даного алгоритму в заданих умовах використання не зустрічаються. При цьому, агломеративний алгоритм показав найгірші результати, розподіляючи набори хаотично. Метод k-середніх показав середній результат. Він може бути застосований, проте для кращого результату слід розглядати інші вхідні параметри для моделі кластеризації, або ж модифікацію методу оцінки складності.

Скорочене порівняння застосованих у експерименті алгоритмів представлене у таблиці 5.2.

Таблиця 5.2 – Порівняння методів кластеризації

Характеристика	K-середніх	Агломеративний	Спектральний
Кількість наборів у кластері	Схожа кількість	Схожа кількість	Середні рівні містять найбільшу кількість, низькі та високі – меншу
Доцільність віднесення набору до кластеру	Середній результат	Віднесення досить хаотичне	Найкращий результат
Недоліки	Проблеми з обробкою аномалій та шумів	Часова складність алгоритму	Обчислювальна складність у більшості випадків

У результатах експерименту представлені результати для одного конкретного набору вагів для заданих метрик. Порівняння різної ваги для кожної з метрик представлене у таблиці 5.3.

Таблиця 5.3 – Порівняння оцінок складності з різною вагою для метрик

W для пам’яті	W для часу	Мінімальна оцінка	Максимальна оцінка
1	2	3	4
0.1	0.9	2.078	2.722
0.2	0.8	3.425	4.021
0.3	0.7	4.765	5.320
0.4	0.6	6.106	6.620
0.5	0.5	7.440	7.919

Кінець таблиці 5.3

W для пам’яті	W для часу	Мінімальна оцінка	Максимальна оцінка
1	2	3	4
0.6	0.4	8.772	9.218
0.7	0.3	10.104	10.517
0.8	0.2	11.437	11.816
0.9	0.1	12.769	13.116

Значення оцінок, представлені у таблиці, не впливають значною мірою на результат кластеризації. Для впливу на розподіл завдань необхідно застосовувати інші метрики, і вже на базі цього аналізувати результати з різною вагою.

5.2 Подальший розвиток дослідження

Результати експерименту показують, що запропонований алгоритм потребує покращень, особливо для того, щоб зробити його таким, що може бути застосований у різних сферах навчання. Необхідно внесення зміни до функції обчислення метрик і функції оцінки. Крім того, слід обирати конкретні алгоритми кластеризації для що більше підходять до того, чи іншого прикладу використання з різними наборами параметрів.

Набори для експерименту були згенеровані шляхом ітерації всіх можливих вхідних даних для кожного параметра. Якщо встановити менший крок діапазону чисел, то кількість створених наборів значно збільшиться. Це можна значно покращити, визначивши дещо точніший спосіб. Наприклад, з усього можливого діапазону можна створити лише частину всіх наборів параметрів. Потім, після оцінки цих наборів та їх кластеризації, можна створити новий сегмент наборів для заповнення наборів середнього рівня. Таким чином, дещо покращується кількісний розподіл завдань, оскільки більшою кількістю завдань заповнюються ті рівні, які цього більше потребують. Ця генерація може бути заснована на методах машинного навчання, а нові набори будуть створюватися за властивостями прикладів з середніх рівнів складності.

Важливим моментом, який слід враховувати, є те, що час, який витрачається на виконання алгоритму розв’язання, залежить від апаратного забезпечення тієї машини, на якому він виконується. Тобто, у випадку, якщо метрика часу все ще використовується, сам алгоритм розв’язання повинен працювати на виділеній машині, яка дасть подібні результати, принаймні в рамках однієї задачі, для якої генеруються набори параметрів. У іншому випадку, результати будуть некоректними.

Методи машинного навчання також можуть покращити визначення та оцінку показників. Показники можуть бути розраховані не тільки за допомогою статичного алгоритму, але й за допомогою певної моделі, наприклад, регресійної [15], яка може передбачати значення метрики за набором параметрів, маючи інформацію про те, як раніше обчислювалась ця метрика на інших наборах.

Сама функція оцінки також може бути покращена за допомогою таких методів, як і метрики. Дана модель може оцінювати набір параметрів, не за статично отриманими метриками, а після навчання на попередніх даних та інформації про те, як оцінювались ті чи інші набори метрик. Однак це приклад навчання з учителем, оскільки початкові набори даних мають бути оцінені статично, а на перших етапах навчання моделі слід перевіряти та виправляти помилки. Отже, при такому підході буде задіяно більше людей, а мета створення алгоритму – зокрема зменшити зусилля викладачів та розробників курсів.

Існують також певні інші покращені, які можна застосувати до механізму оцінки. Наприклад, може бути використане ієрархічне оцінювання для групи різних методів, які можуть розв’язати завдання. У прикладі з квадратним рівнянням це може бути зокрема застосування теореми Вієта на додачу до способу зі знаходженням дискримінанту. Використання такого способу оцінки залежить від області застосування та різноманітності методів, які дозволяють розв’язати початкове завдання.

Найкращим алгоритмом кластеризації в експерименті був спектральний. Після його застосування були створені кластери з найбільшою кількістю наборів параметрів у середніх рівнях складності, що наближено до нормального розподілу.

Це найбільше підходить для реальних випадків застосування, оскільки низький і високий рівні складності, ймовірно, будуть задіяні рідше. Крім того, конкретні набори в кластерах були обрані правильно з точки зору студента – чим більше значущих параметрів в рівнянні та чим вони менші, тим більше часу знадобиться на його розв'язання.

Але оскільки цей конкретний алгоритм має свої недоліки, наприклад, висока вартість обчислень для великих обсягів даних, його використання для завдань із багатьма параметрами може бути проблемою. Проте, занадто велика кількість згенерованих наборів завдань у даному алгоритмі все ж не передбачається.

Методи кластеризації також мають різні вхідні параметри для моделі. Таким чином, налаштування алгоритму не тільки різними методами, але й з допомогою різних параметрів може допомогти знайти найкращу техніку кластеризації для конкретного випадку використання.

Крім того, запропонований алгоритм не описує, як саме зміниться рівень складності в процесі навчання. Це можна зробити, просто вирішивши певну кількість *n* завдань на поточному рівні складності. Однак даний алгоритм можна інтегрувати з графами знань для створення більш глибокої структури навчання. Він може включати не тільки прогрес учня на поточному рівні складності, але й у інших суміжних темах. Отже, процес пошуку нових наборів параметрів для кількох рівнів може мати більше вхідних параметрів, крім лише значення поточного рівня.

Інше припущення, яке можна зробити для подальших модифікацій в алгоритмі, полягає у використанні деревоподібних структур [16] замість кластеризації для визначення процесу навчання. Починаючи з легших рівнів, наступні параметри завдань можна позначити як дочірні гілки кореневого елемента. Рівнем складності в цьому випадку буде глибина параметрів, встановлених у структурі дерева. Таким чином також можна здійснити інтеграцію з графами знань. Таку складну структуру навчання, ймовірно, важко реалізувати, але після належного налаштування вона може стати потужним інструментом для автоматичного та динамічного створення навчальних курсів для студентів.

За результатами експерименту можна сказати, що алгоритм вже можна використовувати для створення простих наборів завдань. При різному рівні складності такі набори мали б великий вплив на мотивацію учнів, а отже, призвели б до кращих результатів від самого процесу навчання. З іншого боку, це значно полегшить роботу викладачів, коли курс передбачає велику кількість практичних завдань, особливо коли можливість сприйняття одних і тих самих завдань різними студентами повинна бути мінімізована.

Але, все-таки викладачі та творці курсів мають значний вплив на результат процесу генерації. Вони визначають багато факторів, які можуть покращити або погіршити результат роботи алгоритму, від визначення діапазонів параметрів до визначення методів, які можуть вирішити поставлене завдання. Тому, будь-які вдосконалення мають бути внесені не тільки в сам алгоритм, але й у процес навчання розробників курсу, щоб ті могли визначити стільки даних, скільки необхідно для отримання найкращого результату. Оскільки опис того, як саме алгоритм буде інтегрований у навчальних системах, не є метою дослідження, то це не є головною метою дослідження. Проте, для подальшого розвитку алгоритму, слід розглядати можливі варіанти полегшення взаємодії вчителя з системою навчання та безпосередньо з алгоритмом.

6 ВИКОРИСТАННЯ РЕЗУЛЬТАТІВ У НАУКОВІЙ ТА ПРАКТИЧНІЙ ДІЯЛЬНОСТІ

Отримані результати дослідження можуть бути застосовані у практичній діяльності, оскільки вже на даному етапі запропонований алгоритм може бути використаний для створення простих наборів завдань, зокрема у сфері вивчення математики. З подальшими модифікаціями, застосування алгоритму може бути розширене. Так, алгоритм може бути застосований у вивченні фізики, хімії, тощо.

При застосуванні динамічної генерації наборів завдань може значно покращитись мотивація учнів, що ці завдання розв’язують. Таким чином покращується ефект від навчання в цілому.

З точки зору наукової діяльності, запропоноване дослідження надає змогу з іншого боку поглянути на вже існуючі методи, такі як, наприклад, графи знань, що використовуються для надання завдань учням. Такі методи базуються на використанні таких ознак прогресу учня, як:

– пройдений матеріал;
– готовий до проходження матеріал;
– недоступний матеріал.

Проте, метод не розглядає, як саме створюються завдання, і, здебільшого, ця задача покладається на вчителя.

Автоматизація переходу учня від розв’язання завдань одного рівня складності до іншого у поєднанні з динамічним створенням цих завдань розкриває нову сферу для досліджень.

Спрямування цих досліджень на поєднання запропонованого алгоритму з уже існуючими методами у свою чергу може призвести до нових способів реалізації онлайн-курсів та, можливо, практичних завдань для освітніх систем у цілому.

Окрім інтеграції з графами знань, алгоритм може бути досліджений на можливість застосування великої кількості засобів машинного навчання, що також значно розширює коло доступних досліджень з наукової точки зору. Зокрема, значна кількість можливих досліджень може бути направлена на методи

кластеризації, або ж застосування складних структур даних. Наприклад, оцінка складності певних наборів завдань з використанням деревовидних структур даних відкриває нові можливості для дослідження застосування таких структур у цілому. Усі наведені можливості можуть бути застосовані як для подальшого розвитку сучасного стану методів навчання людини так і для розвитку у інших сферах загалом.

Дослідження не розкриває, як саме запропонований алгоритм може бути інтегрований у систему навчання. Отже, подальші дослідження на цю тему також можуть зробити вклад у сферу розробки програмного забезпечення в цілому, як механізм впровадження реалізованої системи у вже існуючі навчальні системи.

Таким чином, отримані результати дослідження є цінними як з точки зору практичної, так і наукової діяльності.

ВИСНОВКИ

У результаті проведення дослідження було виявлено, що мотивація учнів є ключовим моментом в їх самоорганізації, а отже, і в процесі навчання. Підвищити мотивацію можна, вводячи в навчання динамічні та в міру складні практичні завдання. Зміст цих завдань має формуватися залежно від успіхів учня: вже вивчені предмети або вже виконані завдання легшого рівня складності.

У ході виконання кваліфікаційної роботи було:

– проаналізовано існуючі методи та алгоритми генерації завдань динамічної складності;

– розроблено власний алгоритм для створення практичних завдань, описано його математичну модель;

– реалізовано програмно запропонований алгоритм;

– проведено експеримент з генерацією набору завдань для тестового прикладу;

– проаналізовано результати експерименту, визначені недоліки алгоритму та подальші кроки для його покращення.

У рамках дослідження були отримані результати щодо працездатності запропонованого алгоритму генерації динамічних завдань, згрупованих за різним рівнем складності. Алгоритм на даному етапі вже може бути застосований для створення наборів простих завдань. А з певними модифікаціями та покращеннями він може бути корисним для різних областей навчання. Алгоритм генерує набори параметрів для завдань, які обмежені певними умовами, заданими користувачем. Для розподілу наборів параметрів пропонується використання кластеризації.

Окрім базового алгоритму, що описаний у роботі, можна також запропонувати достатню кількість покращень у подальшому. Наприклад, визначення та оцінку метрик можна покращити за допомогою методів машинного навчання. Метрики можуть бути розраховані не тільки за допомогою статичного алгоритму, але й за допомогою моделі, яка може передбачити значення метрики з набору параметрів.

Сама функція оцінювання також може бути покращена за допомогою методів машинного навчання так само, як і метрики. Така модель повинна була б оцінювати параметри, встановлені не статично отриманими метриками, а після вивчення того, як були оцінені кілька попередніх наборів. Однак цей процес навчання має бути під наглядом, тому залучатиме більше людей, а мета створення алгоритму – зменшити зусилля викладачів та авторів курсів.

Існує також багато можливих удосконалень, які можна зробити за допомогою механізму оцінки, наприклад ієрархічного оцінювання для групи методів, які можуть вирішити початкову проблему. Це залежить від реальної сфери застосування та різноманітності методів, які дозволяють вирішити початкове завдання.

Крім того, запропонований алгоритм не описує, як саме зміниться рівень складності в процесі навчання. Це можна зробити, просто вирішивши *n* завдань поточного рівня складності. Однак його можна інтегрувати з графами знань для створення більш глибокої структури навчання. Він може включати не тільки прогрес учня на поточному рівні складності, але й інші суміжні теми. Отже, процес пошуку нових наборів параметрів для кількох рівнів може мати більше вхідних параметрів, крім значення єдиного рівня.

Інше припущення, яке можна зробити для подальших модифікацій в алгоритмі, полягає у використанні деревоподібних структур замість кластеризації для визначення процесу навчання. Починаючи з легших рівнів, наступні параметри завдань можна позначити як дочірні гілки кореневого елемента. Рівнем складності в цьому випадку буде глибина параметрів, встановлених у структурі дерева. Таким чином можна також здійснити інтеграцію з графами знань. Наявність такої складної структури навчання, ймовірно, може бути важко реалізувати, але після належного налаштування вона може стати потужним інструментом для автоматичного та динамічного створення навчального курсу для студентів.

Викладачі та автори курсів матимуть значний вплив на результат процесу генерації завдань. Вони визначають багато факторів, які можуть покращити або погіршити результат роботи алгоритму від визначення діапазонів параметрів до

визначення методів, які можуть вирішити поставлене завдання. Тому будь-які вдосконалення мають здійснюватися не тільки у самому алгоритмі, а й у тренуванні у розробників курсів навичок визначення такої кількості даних, яка необхідна для отримання найкращого результату.

Отже, можна зробити висновок, що запропонований алгоритм може бути використаний як у сфері розробки програмного забезпечення в цілому, так і для існуючих навчальних систем зокрема.

Кваліфікаційна робота пройшла апробацію на декількох наукових конференціях, а саме на:

– дванадцятій міжнародній науково-технічній конференції «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління» (додаток В);

– 6th International Conference on Computational Linguistics and Intelligent Systems (Scopus), May 12–13, 2022, Gliwice, Poland (додаток Г).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Малікін Д. Є., Кириченко І. В. Дослідження методів генерації практичних навчальних завдань на основі різних рівнів складності. Дванадцята міжнародна науково-технічна конференція «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління»: тези доп. міжнар. наук-практ. конф, 27 – 28 квітня 2022 р. Баку – Харків – Жиліна, 2022. С. 144.

2. Malikin D., Kyrychenko I. Research of methods for practical educational tasks generation based on various levels of difficulty. 6th International Conference on Computational Linguistics and Intelligent Systems (Scopus), May 12–13, 2022, Gliwice, Poland.

3. Dagiene V., Stupuriene G., Vinikiene L. Implementation of Dynamic Tasks on Informatics and Computational Thinking // Baltic Journal of Modern Computing. 2017. Т. 5. No. 3. С. 306-316.

4. Kormos J., Wilby J. Task Motivation // The Palgrave handbook of motivation for language learning. Palgrave Macmillan, Cham, 2019. С. 267-286.

5. Yilmaz E., Sahin M., Turgut M. Variables Affecting Student Motivation Based on Academic Publications // Journal of Education and Practice. 2017. Т. 8. № 12. С. 112-120.

6. Nawaz S., Srivastava N., Hyun Yu J. How Difficult is the Task for you? Modelling and Analysis of Students' Task Difficulty Sequences in a Simulation-Based POE Environment // International Journal of Artificial Intelligence in Education. 2021.

7. Porayska-Pomsta K. AI as a Methodology for Supporting Educational Praxis and Teacher Metacognition // International Journal of Artificial Intelligence in Education. 2016. Т. 26, № 2.

8. Zou X., Ma W., Ma Z., Baker R. Towards Helping Teachers Select Optimal Content for Students // 20th International Conference, AIED. 2019. С. 413-417.

9. Sharonova N., Kyrychenko I., Tereshchenko G. Application of big data methods in E-learning systems // 5th International Conference on Computational Linguistics and Intelligent Systems (COLINS-2021), 2021, – CEUR-WS, 2021, ISSN 16130073. – Т. 2870, С. 1302-1311.

10. NumPy [Електронний ресурс]. URL: https://numpy.org/ (дата звернення: 14.02.2022).

11. Pydantic [Електронний ресурс]. URL: https://pydantic-docs.helpmanual.io/ (дата звернення: 14.02.2022).

12. Scikit-learn: machine learning in Python [Електронний ресурс]. URL: https://scikit-learn.org/stable/ (дата звернення: 14.02.2022).

13. What is NoSQL? [Електронний ресурс]. URL: https://aws.amazon.com/nosql/ (дата звернення: 22.02.2022).

14. PostgreSQL: The World’s Most Advanced Open Source Relational Database [Електронний ресурс]. URL: https://www.postgresql.org/ (дата звернення: 22.02.2022).

15. Weisstein, Eric W. Normal Distribution [Електронний ресурс]. URL: https://mathworld.wolfram.com/NormalDistribution.html (дата звернення: 13.02.2022).

16. Brass P. Advanced Data Structures, 1st ed., Cambridge University Press, New York, NY, 2008.

17. G. Tereshchenko, I. Gruzdo, Overview and Analysis of Existing Decisions of Determining the Meaning of Text Documents, in: Proceedings of the International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), 2019, pp. 645–653, 8632014. doi: 10.1109/INFOCOMMST.2018.8632014.

18. Z. Dudar. I. Shubin, A. Kozyriev. Individual Training Technology in Distributed Virtual University, Current Trends in Communication and Information Technologies. IPF 2020. Lecture Notes in Networks and Systems, vol 212, pp. 379-399. Springer, Cham. doi:10.1007/978-3-030-76343-5_20.

19. Методичні вказівки до виконання кваліфікаційної роботи магістра за спеціальністю 121 – Інженерія програмного забезпечення (освітньо-наукова програма – Інженерія програмного забезпечення) для усіх форм навчання / Упор.: З. В. Дудар, В. В. Голян, В.І. Каук, В. Ю. Нечволод, І. А. Ревенчук – Харків: ХНУРЕ, 2022 – 81с.