Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження та розробка генетичних алгоритмів розв'язання ігрових задач
(тема)

Виконав:
студент 2 курсу, групи СШІм-21-1
Каверін А.М.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)

Керівник д.т.н., проф., Петров К.Е
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Каверіну Антону Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та розробка генетичних алгоритмів розв'язання ігрових задач

затверджена наказом університету від 31 березня 20 23 р. № 306Ст

2. Термін подання студентом роботи до екзаменаційної комісії 17 травня 20 23 р.

3. Вихідні дані до роботи документації до мови програмування Java, документація до фреймворку Jenetics

4. Перелік питань, що потрібно опрацювати в роботі вивчення предметної області дослідження, дослідження літератури, вивчення проблеми, що потребує рішення, формування постановки задачі дослідження, аналіз існуючих підходів, концептуальне проектування методу рішення задачі

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Рисунок 1 – Схема роботи мутатора, що заснований на моді, Рисунок 2 – Схема роботи мутатора, що заснований на тенденції, Рисунок 3 – Показники сходження алгоритмів, Рисунок 4 – Часові показники роботи алгоритму, Рисунок 5 – Графік використання обсягів пам'яті, Рисунок 6 – Графік використання процесора, Рисунок 7 – Графік кількості поколінь для вирішення задачі

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	03.04.2023	виконано
2	Вивчення предметної області дослідження	03.04.2023– 08.04.2023	виконано
3	Дослідження літератури	09.04.2023– 14.04.2023	виконано
4	Вивчення проблеми, що потребує рішення	15.04.2023– 16.04.2023	виконано
5	Формування постановки задачі дослідження	17.04.2023– 19.04.2023	виконано
6	Аналіз існуючих підходів	20.04.2023– 21.05.2023	виконано
7	Концептуальне проектування методу рішення задачі	22.04.2023– 25.04.2023	виконано
8	Програмна реалізація та тестування	26.04.2023– 05.05.2023	виконано
9	Написання пояснювальної записки	06.05.2023– 11.05.2023	виконано
10	Попередній захист	13.05.2023	виконано
11	Захист перед ЕК	17.05.2023	виконано

Дата видачі завдання 3 квітня 2023 р.

Студент _____ _____
(підпис)

Керівник роботи _____ д.т.н., проф., Петров К.Е.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 60 с., 7 рис., 4 табл., 1 дод., 28 джерел.

ГЕНЕТИЧНИЙ АЛГОРИТМ, ГЛИБИННЕ НАВЧАННЯ, ІГРОВА ЗАДАЧА, МОДА, МУТАЦІЯ, ТЕНДЕНЦІЯ, JAVA, JENETICS, JVM

Об'єкт дослідження – процес розв'язання ігрових задач з використанням еволюційних методів.

Предмет дослідження – методи розв'язання ігрових задач за допомогою генетичних алгоритмів та підходи до їх навчання.

Мета роботи – розробка методу розв'язання ігрових задач на основі використання генетичних алгоритмів.

Дослідження, що проведені в роботі, базуються на використанні методів системного аналізу ігрових задач, апарату генетичних алгоритмів, методів розв’язання ігрових задач, методів верифікації точності запропонованого підходу.

У роботі проведено теоретичний аналіз літературних джерел щодо застосування еволюційних методів, зокрема генетичних алгоритмів для безпосереднього використання при розв’язанні ігрових задач. Розроблено власний метод розв’язання ігрових задач з використанням понять моди та тренду на основі запропонованого генетичного алгоритму. Також створено пакет програмного забезпечення для проведення тестування та експериментальної перевірки розробленого методу. Розроблено усі необхідні метрики та обчислені показники ефективності запропонованого методу, що базується на використанні генетичного алгоритму для оцінки та аналізу результатів його застосування для розв’язання ігрових задач.

ABSTRACT

Explanatory note: 60 p., 7 fig., 4 tabl., 1 ann., 28 sources.

DEEP LEARNING, GAME TASK, GENETIC ALGORITHM, JAVA, JENETICS, JVM, MEAN, MUTATION, TREND

The object of research is the process of solving game problems using evolutionary methods.

The subject of research is methods of solving game problems using genetic algorithms and approaches to their learning.

The purpose of the work is to develop a method for solving game problems based on the use of genetic algorithms.

The research carried out in the work is based on the use of methods of system analysis of game problems, the apparatus of genetic algorithms, methods of solving game problems, methods of verification of the accuracy of the proposed approach.

The work provides a theoretical analysis of literary sources regarding the application of evolutionary methods, in particular genetic algorithms for direct use in solving game problems. A proprietary method of solving game problems has been developed using the concepts of fashion and trend based on the proposed genetic algorithm. A software package was also created for testing and experimental verification of the developed method. All the necessary metrics and calculated performance indicators of the proposed method, which is based on the use of a genetic algorithm for evaluating and analyzing the results of its application for solving game problems, have been developed.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 8
Вступ 9
1 Аналіз предметнох області та постановка задачі дослідження 10
1.1 Аналіз предметної області 10
1.2 Аналіз існуючих варіацій генетичного алгоритму 11
1.3 Постановка задачі 13
2 Розробка методу розв'язання ігрових задач на основі використання генетичного алгоритму 15
2.1 Аналіз проблем існуючих методів на основі генетичних алгоритмів 15
2.2 Визначення архітектури мутатора на основі понять моди та тенденції 16
2.2.1 Поняття моди 16
2.2.2 Поняття тенденції 18
2.2.3 Поняття коефіцієнту мінімально допустимої кількості значень 19
3.3 Опис комплексного мутатора на основі тенденцій та моди 20
3 Вибір та обгрунтування наботу інструментів та технологій розробки 21
3.1 Вибір мови програмування 21
3.2 Вибір фреймворку для реалізації генетичного алгоритму 24
3.3 Вибір фреймворку для тестування 28
3.4 Вибір бібліотеки для збору даних 31
4 Програмна реалізація методу розв'язання ігрових задач 36
4.1 Вибір ігрової задачі для тестування методу 36
4.2 Реалізація програмного забезпечення для вирішення ігрової задачі. 37
4.3 Оптимізація програмного забезпечення 42
4.4 Оптимізація віртуальної машини 44

5 Експериментальна перевірка запропонованого методу розв’язання ігрових задач 47
5.1 Опис експерименту та показників оцінки ефективності методу 47
5.2 Аналіз показника сходження для генетичного алгоритму 48
5.3 Аналіз показника швидкості роботи для генетичного алгоритму 49
5.4 Аналіз показників використовуваних ресурсів 50
5.5 Аналіз переваг та недоліків методу 54
5.7 Аналіз отриманих результатів 55
Висновки 56
Перелік джерел посилання 57
Додаток А Відомість кваліфікаційної роботи 60

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ГА – генетичний алгоритм;

Фреймворк – інфраструктура програмних рішень;

GA – genetics algorithm – генетичний алгоритм;

JVM – Java Virtual Machine – віртуальна машина джава.

ВСТУП

Починаючи з кінця минулого сторіччя, сфера ігрових розваг еволюціонувала та розвивалася невпинно величезними кроками і зараз займає важливу роль як у світовій економіці, де ігри займають не останнє місце з прибутків, так і у повсякденному житті звичайних людей, де ігри інтегрувалися в життя та дозвілля настільки сильно, що деякі з них не бачать існування без них.

На даний момент існують мільйони ігор різними за жанрами, за масштабами та наповненням, але всі вони об’єднані їх скінченністю. Це означає, що, досягнувши певних умов, іноді, досить різних, гра буде вважатися повністю закінченою. Таким чином поняття закінчення гри можна формалізувати як задачу багатокритеріальної оптимізації, де важко визначити остаточне ідеальне рішення, котре буде повністю задовольняти певним умовам.

В багатьох задачах багатокритеріальної оптимізації, де неможливо знайти точне рішення за прийнятний час, використовують різноманітні евристичні методи [1]. Досить часто для цих цілей використовують генетичні алгоритми, особливістю котрих є наявність операцій схрещування та мутації поміж кандидатами, які розглядаються в якості розв’язку [2], а тому удосконалення або розробка нових методів оптимізації з метою підвищення ефективності розв’язання ігрових задач є достатньо актуальною проблемою.

Окрім цього, як і будь-який інший алгоритм, генетичний алгоритм(ГА) має певний ряд недоліків, а тому в результаті роботи має бути знайдене рішення з покращення одного чи більше якісних чи кількісних показників існуючих варіантів генетичних алгоритмів.

Зважаючи на все викладене вище, мета роботи полягає у розробці власного методу розв'язання ігрових, який би мав кращі показники у порівнянні з існуючими підходами.

1 АНАЛІЗ ПРЕДМЕТНОХ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз предметної області

Протягом останніх декількох десятиріч інформаційні технології зробили величезний стрибок у розвитку. При цьому зміни були пов’язані не лише з підвищенням потужностей обчислювальних машин, а й з їх популяризацією серед звичайних громадян.

Розповсюдження потужної комп’ютерної техніки серед звичайних людей дало змогу надавати величезну кількість послуг, серед яких можна виділити власне корисні для людини як громадянина додатки і сайти, а також величезний спектр різних розважальних послуг на будь-який смак. І звичайно, що серед цього безмежного обсягу розважальних послуг є ігри.

Гра – це діяльність людини з моделювання іншого виду діяльності з розважальною чи навчальною метою. Гра відрізняється від роботи тим, що не ставить перед собою безпосередньо корисної мети (хоча сама гра може мати свою власну користь для користувача), а також тісно межує з мистецтвом, хоч зазвичай не створює художніх цінностей.

Якщо трохи звузити термін гри до комп’ютерної гри, то у такому випадку отримуємо все теж саме, але з додаванням взаємодії людини з комп’ютером, котрий емулює віртуальну реальність та надає змогу для інтерактивної діяльності.

Разом з цим починаючи з кінця двадцятого сторіччя, набули популярності і важливості розробки, що пов’язані зі штучним інтелектом. Передумовами цього стали знову ж таки популяризація комп’ютерів, здешевлення обчислювальних потужностей, носіїв інформації, а також прискорення інтернету.

Усе вище перераховане надало змогу штучному інтелекту перейти з межі фантастики до сьогоденності, де він використовуються в дуже

багатьох сферах – від персоналізованих пропозицій в інтернет-магазинах чи потокових платформах і до генерації зображень за текстом чи покращення медіа файлів будь-якого формату.

Звичайно, що штучний інтелект торкнувся і ігрової сфери, де він використовується як для генерації ігрових мап, оптимізації комп'ютерного обладнання, так і до навчання штучного інтелекту проходженню ігор чи вирішенню певних ігрових завдінь з метою подальшого покращення гри – навчання неігрових персонажів з метою їх перетворення в більш реалістичних та схожих на реальних гравців [3].

Один з достатньо ефективних варіантів навчання неігрових персонажів в іграх – це використання генетичних алгоритмів, котрі мають на меті збудувати якомога більш ефективного бота [4]. Проте важливо зазначити, що як і будь-який інший алгоритм, генетичний алгоритм має ряд серйозних недоліків.

Тому основною задачею роботи є покращення характеристик та ефективності методів, що базуються на використанні генетичного алгоритму при застосуванні їх в ігровій сфері при однакових витратах часу.

1.2 Аналіз існуючих варіацій генетичного алгоритму

Варіацій генетичних алгоритмів існує дуже велика кількість. В залежності від різних поєднань селекторів, мутаторів та/або кросоверів можна виділити різні види алгоритмів.

Окрім цього, можливо використання декількох різних варіантів мутацій та/або кросоверів, що ще сильніше збільшує варіативність генетичного алгоритму. Але серед них можна виділити ряд найбільш популярні мутатори, котрі являються базовими і дуже часто зустрічаються. До них відносять:

– мутатор перестановки – це вид мутації, котрий може бути використаний як для побітових операцій, так і для операцій з реальними

величинами. Суть мутатора полягає в том, що обираються 2 випадкові точки в геномі та обмінюються значеннями один з одним [5]. Точки для обміну обираються такі що:

$$i \leq p_1 \leq j$$
$$i \leq p_2 \leq j, \qquad (2.1)$$

де *i* – початковий індекс послідовності бітів чи реальних значень;

j – кінцевий індекс послідовності бітів чи реальних значень;

p1, *p2* – відповідні позиції для перестановки першого для другого елемента, але такі що $p1 \neq p2$.

– мутатор інверсії – це вид мутації, котрий зберігає основну частину геному та мутує краї послідовності. Він також може бути використаний як для побітових значень, так і для реальних. Алгоритм полягає в знаходженні двох точок, таких як і у рівнянні 2.1, та зберіганні послідовності даних у заданих межах [5]. В той самий час, частини, що не увійшли до результуючої послідовності попадаються на свої місця після розташування їхніх елементів з заду на перед.

– мутація з перегортанням бітів – це вид мутації, що може бути використаний лише для побітових операції. При такому виді мутації обирається індекс послідовності бітів та змінюється на протилежне значення [6, 7].

Аналізуючи описані мутатори, можна зробити висновок, що всі вони цілком базуються на ймовірнісних характеристиках, що призводить до того, що задача пошуку результату зводиться лише до певних ймовірностей. Це означає, що вирішення завдання може бути знайдене лише з певною ймовірністю, яка зазвичай є достатньо високою, але не дає жодних гарантій. Тому часто необхідно використовувати декілька паралельно запущених екземплярів генетичного алгоритму та/або перезапускати алгоритм декілька, іноді десятків, разів з метою отримання хоча б якого-небудь

результату.

Звичайно впливати на ймовірності та підвищувати їх можна багатьма способами, але результат у будь-якому випадку залежить цілком і повністю від ймовірності та кількості мутацій [8].

А тому важливим кроком є створення власного методу на основі генетичного алгоритму або додавання нової ланки у існуючий ланцюг генетичного алгоритму, який був би спроможний на зменшення впливу ймовірнісних показників під час мутації [9].

1.3 Постановка задачі

Метою даної роботи є дослідження та аналіз методів розв'язання ігрових задач з використанням генетичних алгоритмів, а також розробка на їх основі власного підходу.

Для досягнення поставленої мети необхідно вирішити такі основні завдання:

– провести аналіз предметної області;

– дослідити існуючі методи та порівняти їх;

– розробити та описати власний підхід до розв'язання ігрових задач з використанням генетичних алгоритмів;

– провести аналіз отриманих результатів.

Необхідно розробити власний алгоритм чи частину алгоритму на основі генетичного алгоритму для розв’язання ігрових задач. Головною метою роботи є покращення якісних та кількісних показників у порівнянні з іншими розглянутими алгоритмами при використанні одних і тих самих часових показників.

В ході проектування мають бути розглянуті існуючі варіанти генетичного алгоритму, повинні бути проаналізовані їх переваги та недоліки і, базуючись на отриманій інформації, повинен бути розроблений власний підхід.

Додатково, має бути проведений аналіз доступних мов програмування, бібліотек та фреймворків, а також інструментів, які можуть допомогти в реалізації, тестуванні, розробці метрик оцінювання та порівнянні отриманих рішень. Мають бути виділені їхні переваги та недоліки та основі отриманої в ході аналізу інформації повинен бути зроблений висновок стосовно найбільш доречних та ефективних мов програмування, бібліотек, фреймворків та інструментів.

Окрім цього, має бути розроблено програмне забезпечення, що реалізує запропонований підхід, а також надає можливість провести його тестування при розв’язанні різних прикладів задач одного виду. Програмне забезпечення має бути первинно протестованим задля виявлення первинних проблем та похибок.

При наявності проблем з продуктивністю роботи алгоритму, мають бути досліджені та, за можливості, реалізовані шляхи оптимізації та покращення алгоритму задля максимального покращення показників та метрик.

Маючи реалізовані елементи системи, необхідно провести тестування вже існуючого та власного методу на реальних прикладах та зібрати статистичні дані щодо ефективності роботи та порівняти їх один із одним.

На основі зібраних метрик та статистики мають бути створення таблиці метрик та візуальні зображення, які показують зміни при використання запропонованого методу у порівнянні з існуючими.

Результатом порівняння мають стати схеми, таблиці, рисунки чи графіки, які наглядно показують значення кількісних метрики кожного з методів.Значення метрик мають бути проаналізовані та їх основі мають бути зроблені висновки що ефективності роботи запропонованого методу у порівнянні з іншими існуючими реалізаціями. Мають бути проаналізовані переваги та недоліки нового методу у порівнянні з існуючими та наведені кількісні показники змін.

2 РОЗРОБКА МЕТОДУ РОЗВ'ЯЗАННЯ ІГРОВИХ ЗАДАЧ НА ОСНОВІ ВИКОРИСТАННЯ ГЕНЕТИЧНОГО АЛГОРИТМУ

2.1 Аналіз проблем існуючих методів на основі генетичних алгоритмів

Аналізуючи більшість сучасних популярних видів мутації можна помітити, що всі вони базуються на ймовірнісних показниках, що означає, що рішення може бути знайдене з певною ймовірністю, а також це значно впливає на інші показники, а саме час роботи та точність.

На перший погляд здається, що описаний вплив є не досить важливим, але у реальності це призводить до того, що збіжність генетичного алгоритму є одною з ключових його проблем. Це також означає, запустивши алгоритм для вирішення певної задачі та чекаючи протягом певного часу, можна не отримати точно результату або будь-якого результату зовсім [10].

Окрім цього іншою важливою проблемою генетичного алгоритму є виділення функції допасованості, а саме те, що для багатокритеріальної оптимізації виділення та виконання цієї функції навіть для одного індивіду може витрачатися багато часу, що у свою чергу має величезний вплив на виконання функції для всього покоління.

У такому випадку вирішення задачі може потребувати забагато часу чи комп'ютерних ресурсів, що у свою чергу означає високі грошові витрати.

Таким чином для покращення роботи методу, шо базується на використанні генетичного алгоритму можна взятись за вирішення будь-якої з описаних проблем, або за обидві одразу.

Так як найважливішою причиною проблеми збіжності генетичного алгоритму є повна залежність від ймовірнісних показників, то було вирішено додати можливість для більш детермінованої роботи алгоритму. Головною ідеєю стає збір статистики під виконання навчання поколінь та, базуючись на попередніх показниках, виконування мутації окремих генів чи окремих реальних об'єктів.

2.2 Визначення архітектури мутатора на основі понять моди та тенденції

Як було описано раніше, основою методу стає збір та аналіз даних, щодо вирішення задачі попередніми поколіннями та визначення наступного стану гену на основі проаналізованих даних.

Для того, щоб досягти такої поведінки можна визначити два поняття, що будуть описувати стан системи у попередні моменти часу, а саме поняття: моди та тенденції. Вони обидві мають на меті збір, аналіз даних та виділення найбільш популярних значень попередніх поколінь з різницею в довжини оброблюваної кількості попередників.

Окрім цього, необхідно визначити поняття мінімально допустимої кількості значень, який буде використовуватися для знаходження найбільш самодостатнього значення гену певної позиції.

2.2.1 Поняття моди

Мода в загальному розумінні – це значення випадкової величини, що трапляється найчастіше зі всієї сукупності спостережень. Тобто це значення, що зустрічається в масиві даних частіше аніж інші. Також, в сукупності даних іноді можна виділити більше ніж одну моду.

Накладаючи значення моди на генетичній алгоритм можна сказати, що це значення бітів в геномі або реальне значення в поколінні, що зустрічається частіше за інше. Мода має бути виділена протягом дуже великої вибірки значень поколінь, тому ще саме поняття моди має на меті визначити найпопулярніше значення зі всього масиву значень. Тобто, мода може визначатись починаючи з початкового нульового покоління або протягом дуже великої кількості ітерацій, наприклад: 100, 1000 або 50000, в залежності від кількості генерованих поколінь для вирішення задачі. На рисунку 3.1 зображена теоретична схема роботи мутатора основаного на

понятті моди.

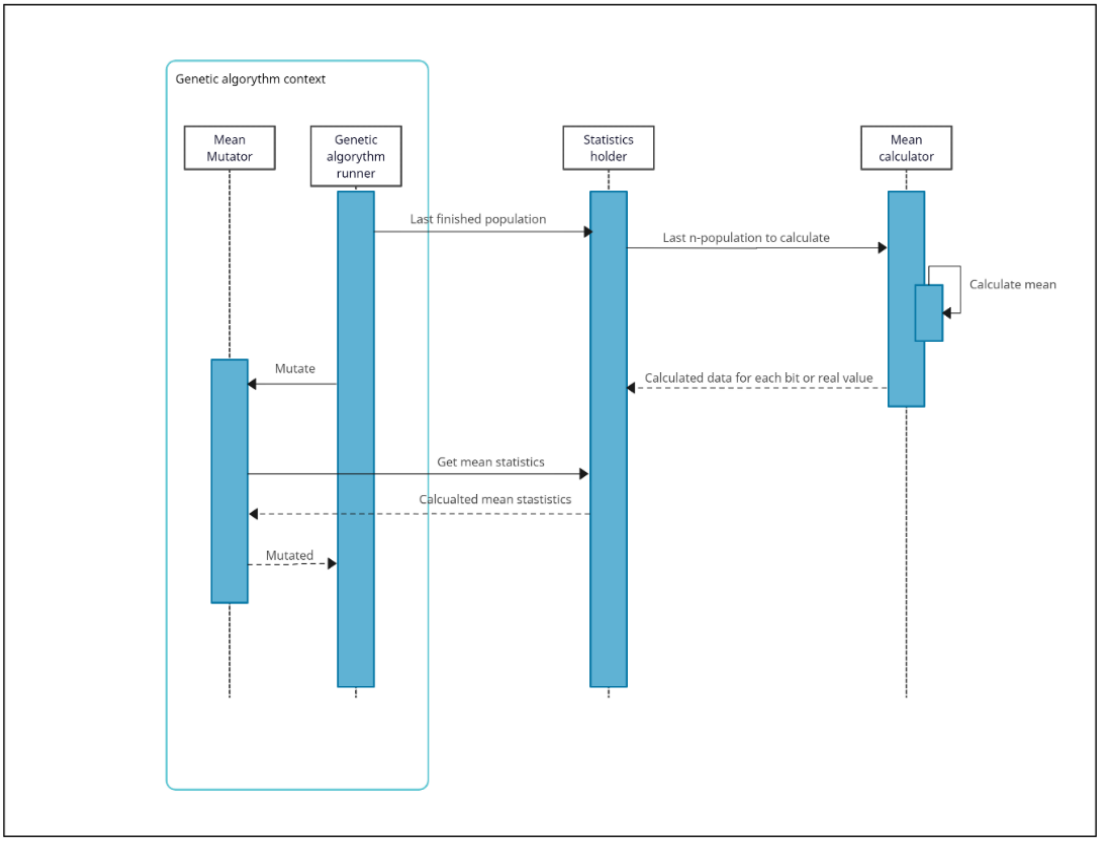

Рисунок 3.1 – Схема роботи мутатора, що заснований на моді

Виходячи з рисунку 3.1 процесор генетичного алгоритму після кожної ітерації надсилає дані про останнє покоління до зберігача статистики. Він у свою чергу надсилає дані з необхідною кількістю поколінь до калькулятора моди, котрий обчислює отримані дані та повертає найпопулярніші значення для геному задачі.

Наступний крок починається коли процесор генетичного алгоритму викликає мутатор на основі моди для мутації геному окремого індивіду. Мутатор у свою чергу запитує зберігача статистики на отримання останніх обчислених даних щодо геному. Зберігач повертає дані до мутатор на основі моди і він відповідно виконує мутації базуючись на отриманих даних.

2.2.2 Поняття тенденції

Тенденція або тренд в загальному розумінні – це значення випадкової величини, що трапляється найчастіше з дуже малої сукупності спостережень.

Використовуючи термін тенденції в генетичному алгоритмі можна сказати, що це значення, що трапляться і масиві бітів геному чи реальних значень частіше за інші, але період вибірки має бути набагато меншим за обраний у понятті моди. На відміну від моди, що по суті своїй є найпопулярнішим значення досить великого періоду навчання, тенденція означає яскраву різку і ймовірно якісну відмінність, котра виникла декілька поколінь тому і має популярність серед всіх цих поколінь.

Процес обчислення тенденції представлений на рисунку 3.2.

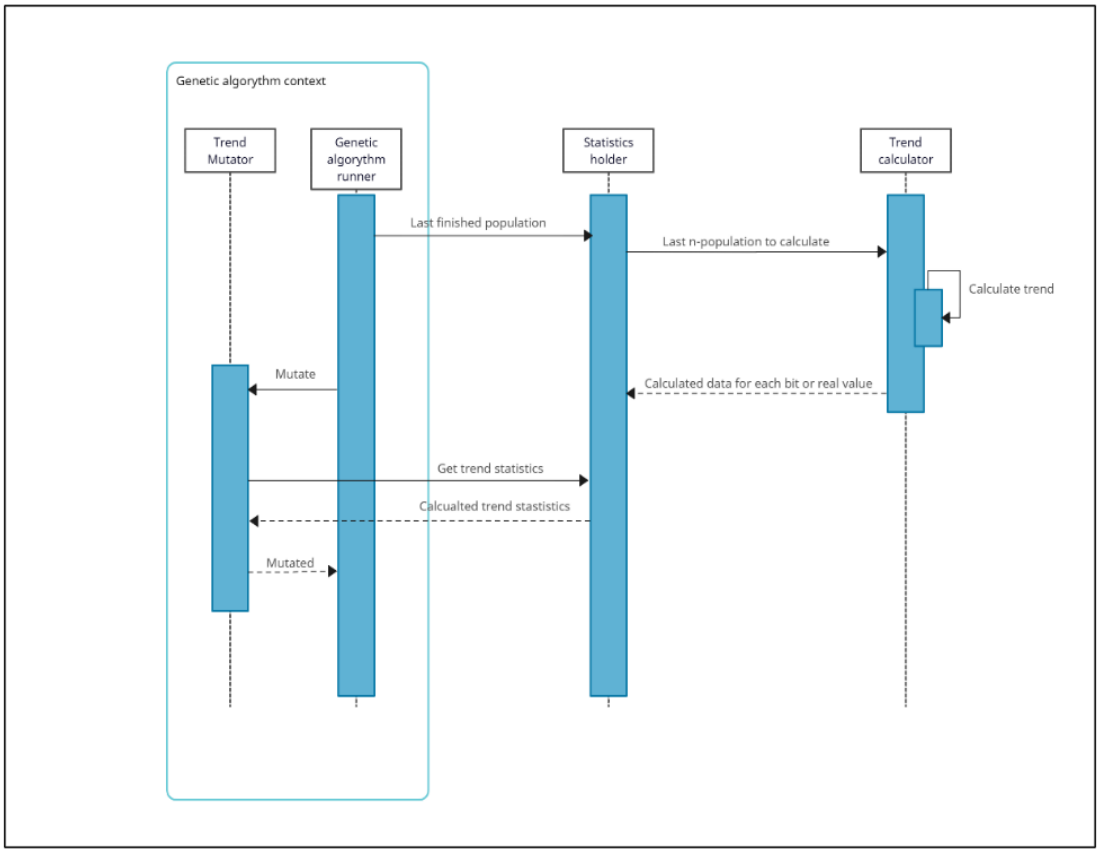

Рисунок 3.2 – Схема роботи мутатора, що заснований на тенденції

Спираючись на рисунок 3.2 можна зробити висновок, що алгоритм роботи дуже схожий на схему роботи мутатора заснованого на моді, але важливою різницею є те, що кількість поколінь, яка передається для обчислення тенденції, є значно меншою у порівняні з аналогічним для моди.

2.2.3 Поняття коефіцієнту мінімально допустимої кількості значень

Аналізуючи отримані алгоритми роботи мутаторів на основі моди та тренду можна прийти до певної проблеми, котра може сильно впливати на точність та збіжність алгоритму. Головна проблема полягає в тому, що найпопулярніше значення означає лише те, що значення зустрічається частіше аніж усі інші, але різниця між ними невідома. Тобто значення найпопулярнішого елемента може бути на одиницю більшим за наступне по популярності значення.

Окрім цього, існує ще одна важлива проблема, а саме те, що коли значення найпопулярніших елементів відомо, то при високій ймовірності та кількості мутацій індивіди наступних поколінь, отриманих за рахунок використання таких мутацій, можуть повністю збігатися один із одним.

У такому випадку ідеальним рішенням буде додавання коефіцієнту мінімально допустимої кількості значень. Цей коефіцієнт, як можна зрозуміти з назви, задає мінімальну кількість елементів у відсотках чи в умовних одиницях, котрі мають бути присутня для того, що вважати це значення найпопулярнішим.

По-перше, такий коефіцієнт надає змогу ефективно вирішувати першу проблему за рахунок того, що, наприклад, задавши значення більше ніж п’ятдесят відсотків ми отримуємо єдиний і гарантований варіант значення, якщо таке значення, звичайно, зустрічається настільки часто.

По-друге, це наддасть змогу виділите лише найбільш популярні значення серед позицій індивідів, що у свою чергу бути підвищувати важливість таких мутацій.

3.3 Опис комплексного мутатора на основі тенденцій та моди

Формалізувавши мутатори на основі тенденції та моди можна створити більш комплексний і ефективний мутатор, котрий буде поєднувати обидва підходи.

Як і у випадка мутаторів на основі моди та тенденції, існує потреба у зборі статистики для кожного покоління з метою подальшої обробки та обчислення статистики моди та тенденції. Також, як і у попередніх прикладах, все ще існує потреба в мінімально допустимої кількості значень, так як він прямо використовується в попередніх мутаторах.

Як і попередні мутатори, цей мутатор має певну проблему, пов’язану з тим, що для деяких індексів знаходження найпопулярнішого значення може бути неможливим через високу варіативність значень або через дуже великий коефіцієнт мінімально допустимої кількості значень, а тому можливий сценарій, коли мутатор не знайде жодного підходящого значення ні з боку мутатори на основі моди, ні з боку мутатора на основі тенденції, а тому у такому випадку метод має виконувати мутацію одним із доступних існуючих методів.

Базуючись на отримай проаналізованій інформації, можна описати загальний алгоритм роботи мутатора на основі моди та тенденції таким чином:

– генеруємо випадкове значення індексу послідовності геному;

– намагаємося знайти значення для згенерованого індексу серед статистики тенденцій. Якщо значення було знайдене, то виконуємо мутацію та закінчуємо загальний процес мутації, а якщо значення не було знайдено, то переходим до наступного кроку;

– намагаємося знайти значення для згенерованого індексу серед статистики моди. Повторюємо кроки як для мутації через тенденцію;

– виконуємо мутацію через мутатор за замовчуванням.

3 ВИБІР ТА ОБГРУНТУВАННЯ НАБОТУ ІНСТРУМЕНТІВ ТА ТЕХНОЛОГІЙ РОЗРОБКИ

3.1 Вибір мови програмування

Генетичні алгоритми широко використовуються для розв'язання різних завдань оптимізації та машинного навчання, і вибір правильної мови програмування може значно вплинути на ефективність їх роботи. Від вибору мови програмування залежить швидкість виконання алгоритму, розмір програмного коду, легкість розробки та підтримки проекту, а також кількість доступних бібліотек та інструментів.

У залежності від конкретної задачі, можуть бути підходящі різні мови програмування. Наприклад, для великих і складних проектів можуть бути підходящі мови з сильним типізацією та широкими можливостями об'єктно-орієнтованого програмування, такі як Java або C#. Для проектів, які потребують великої швидкості обробки даних, можуть бути більш підходящими мови з високою продуктивністю, такі як C++ або Rust.

Незалежно від обраної мови програмування, важливо мати на увазі особливості генетичного алгоритму та його реалізації. Ефективність алгоритму залежить від правильного вибору та налаштування параметрів, відповідної структури даних та алгоритмів обробки, а також від професійного розуміння основ програмування [11].

Однією з найбільш популярних мов програмування для генетичних алгоритмів є Java.

Java – це об’єктно-орієнтована мова програмування, яка має широкі можливості для розробки додатків в галузі штучного інтелекту, включаючи генетичні алгоритми. Java має велику кількість бібліотек та інструментів, що дозволяють зручно та ефективно працювати з генетичними алгоритмами [12].

Однією з переваг Java є те, що вона є кроссплатформенною мовою програмування. Це означає, що програми, написані на мові Java, можуть запускатись на будь-якій операційній системі без необхідності в розробці окремих версій для кожної платформи. Java може працювати на будь-якій операційній системі, включаючи Windows, Mac та Linux. Це дуже важливо для генетичних алгоритмів, оскільки вони можуть вимагати великих обчислювальних ресурсів. Кроссплатформенність дає можливість запускати генетичні алгоритми на будь-якому комп'ютері, що робить їх більш доступними та зручними для використання.

Крім того, Java забезпечує безпеку від несанкціонованого доступу до ресурсів системи. Завдяки вбудованим механізмам безпеки, які реалізовані в Java Virtual Machine, можливість зламати програму, написану на мові Java, дуже мала. Це дуже важливо, оскільки генетичні алгоритми можуть містити конфіденційну інформацію або працювати зі складними алгоритмами, які можуть бути вразливі до атак.

Завдяки цим перевагам, розробка генетичних алгоритмів на мові Java може бути більш ефективною та безпечною для користувачів. Таким чином, використання Java для генетичних алгоритмів може забезпечити найкращі результати та забезпечити безпеку і конфіденційність ваших програм [13].

Також, однією з головних переваг Java для розробки генетичного алгоритму є підтримка багатопотоковості, що дозволяє розробляти алгоритми, які працюють з кількома потоками одночасно.

Генетичний алгоритм є ітеративним процесом, який використовує випадковість і еволюційний принцип для розв'язання оптимізаційних задач. Такий алгоритм може займати багато часу для обчислення, особливо якщо вхідні дані великі. Використання багатопоточності може значно збільшити швидкість роботи генетичного алгоритму, розбивши обчислення на декілька потоків, які виконуються паралельно.

Java надає програмістам зручність у використанні багатопотоковості. Вона має вбудовані бібліотеки, такі як java.util.concurrent, які дозволяють

легко створювати та керувати потоками. Бібліотеки дозволяють створювати паралельні потоки, а також динамічно керувати кількістю потоків [14].

Окрім того, Java має вбудований механізм синхронізації, який дозволяє уникнути проблем, пов'язаних з доступом до спільних ресурсів з більш ніж одного потоку. Це дозволяє запобігти помилкам, таким як «гонки» (race conditions) та «блокування» (deadlocks). Крім того, Java має велику кількість стандартних бібліотек, що дозволяє програмістам працювати з різними видами даних та алгоритмами без додаткових зусиль. Наприклад, стандартна бібліотека містить класи для роботи з колекціями даних, обробки рядків, введення-виведення, множинними потоками та багато іншого.

Окрім цього, вона має розвинену систему винятків та виключень, що дозволяє зручно обробляти помилки під час виконання програми. Це може бути корисно при роботі з генетичними алгоритмами, де можуть виникати непередбачувані ситуації під час еволюції популяції.

Нарешті, Java є дуже популярною мовою програмування. Ця популярність також стала важливим фактором для вирішення задач генетичних алгоритмів.

По-перше, велика кількість програмістів, які володіють мовою програмування Java, забезпечує наявність великої спільноти, що допомагає вирішувати проблеми та розробляти нові інструменти. Це включає в себе форуми, блоги, веб-сайти та відкриті джерела, які забезпечують доступ до більшості різноманітних інструментів, бібліотек та фреймворків, які можуть бути використані для розробки генетичних алгоритмів.

По-друге, Java має багато вбудованих інструментів для розробки генетичних алгоритмів, таких як бібліотеки для математичних обчислень, роботи з файлами та мережами, що спрощує процес розробки та знижує витрати на розробку [15].

Також Java має вбудовану систему керування пам'яттю, що дозволяє уникнути багатьох помилок, пов'язаних з ручним керуванням пам'яттю. Для

задач генетичних алгоритмів, які зазвичай мають велику кількість об'єктів та структур даних, таке керування пам'яттю стає надзвичайно важливим.

Пам'ять генетичного алгоритму може стати дуже великою, залежно від розміру задачі та складності алгоритму. Використання Java дозволяє уникнути проблем, пов'язаних з неадекватним виділенням пам'яті, відповідним звільненням пам'яті та управлінням її розподілом.

У мові програмування Java існує механізм автоматичної зборки сміття, що дозволяє автоматично відслідковувати та звільняти невикористану пам'ять, що робить процес програмування значно простішим та ефективнішим. Це також знижує ризик помилок програміста, пов'язаних з неправильним використанням пам'яті, що може вплинути на результати генетичного алгоритму [16].

Отже, враховуючи перелічені властивості, можна стверджувати, що мова програмування Java є ідеальним варіантом для вирішення задач генетичного алгоритму. Вона дозволяє ефективно використовувати ресурси та забезпечує зручне та швидке програмування.

3.2 Вибір фреймворку для реалізації генетичного алгоритму

Для мови програмування Java існує низка бібліотек та фреймворків для створення генетичних алгоритмів. До них входять:

а) JGAP (Java Genetic Algorithms and Genetic Programming Framework) є одним з популярних фреймворків для створення генетичних алгоритмів з відкритим вихідним кодом. Розглянемо переваги та недоліки цього фреймворку.

До переваг JGAP можна віднести:

1) відкритий вихідний код дозволяє легко редагувати та модифікувати фреймворк під власні потреби;

2) легкість використання завдяки гнучкій архітектурі та великій кількості документації;

3) наявність готових реалізацій різних операторів для генетичних алгоритмів та генетичного програмування, таких як вибірка, мутація, кросинговер та інші;

4) надійна підтримка великої кількості типів даних та функцій.

До недоліків JGAP відносять:

1) обмежена функціональність порівняно з іншими фреймворками;

2) погана оптимізація, що може впливати на продуктивність;

3) для повноцінної роботи з фреймворком необхідно мати базові знання з генетичних алгоритмів [17].

б) Apache Commons Genetics Algorythm(GA) – це бібліотека для генетичних алгоритмів, яка надає високорівневий API для створення та запуску генетичних алгоритмів.

До переваг Apache Commons GA відносять:

1) легкість використання – надає простий інтерфейс та API, що дозволяє швидко створювати та налаштовувати генетичні алгоритми;

2) підтримку різних видів оптимізаційних задач – бібліотека може бути використана для рішення різних оптимізаційних задач, включаючи пошук максимального або мінімального значення функції, вибір найкращих параметрів моделі та інше;

3) висока швидкодія – бібліотека була оптимізована для роботи з великими об’ємами даних та дозволяє ефективно обробляти значну кількість популяцій та особин;

4) підтримка різних типів хромосом – Apache Commons GA підтримує різні типи хромосом, включаючи бінарний, дійсний та перестановочний, що дозволяє використовувати бібліотеку для різних задач.

До недоліків Apache Commons GA відносять:

1) відсутність документації – деякі користувачі відзначають, що документація фреймворку не зовсім повна, іноді незрозуміла і містить помилки;

2) обмежену функціональність – Apache Commons GA надає базовий набір функцій, які можуть бути недостатніми для більш складних завдань;

3) нестабільність – деякі користувачі зазначають, що фреймворк може бути нестабільним під час виконання деяких завдань, що може призвести до помилок або неправильних результатів;

4) відсутність підтримки: останнє оновлення було випущене у 2010 році, що може свідчити про відсутність підтримки та вдосконалення [18].

в) ECJ (Evolutionary Computation in Java) – це фреймворк для реалізації різних видів еволюційних алгоритмів, включаючи генетичні алгоритми. Основні переваги Evolutionary Computation in Java включають:

1) гнучкість та розширюваність – ECJ дозволяє легко налаштувати генетичні алгоритми для вирішення різних задач. Він має велику кількість модулів, які можуть бути легко налаштовані та використані для реалізації різних алгоритмів;

2) можливості паралелізації – ECJ може бути легко розпаралелено на багато потоків, що дозволяє збільшити швидкість обчислень;

3) відкритий код – ECJ є відкритим програмним забезпеченням, що дозволяє дослідникам та розробникам вносити свої внески у розвиток бібліотеки;

4) легкість використання – ECJ має документацію та приклади коду, що робить його легко зрозумілим та використовуваним для новачків у генетичному програмуванні.

Недоліки Evolutionary Computation in Java:

1) низька продуктивність – ECJ може мати низьку продуктивність, особливо при роботі з великими об’ємами даних, через неефективне використання пам’яті;

2) складність налаштування – ECJ має багато параметрів, які можуть бути налаштовані, що може бути складно для новачків у генетичному програмуванні;

3) обмежена функціональність – ECJ має обмежену функціональність порівняно з деякими іншими бібліотеками для генетичного програмування [19].

г) Jenetics – це фреймворк генетичних алгоритмів для мови програмування Java, який дозволяє легко створювати оптимізаційні програми, використовуючи генетичний алгоритм.

Головними перевагами Jenetics є наступні:

1) простота використання – фреймворк має простий і зрозумілий API, який дозволяє розробникам легко створювати свої власні генетичні алгоритми і зосереджуватися на самій оптимізації, а не на деталях реалізації алгоритму;

2) Jenetics надає розширені можливості для налаштування генетичних алгоритмів. Є можливість встановлювати різноманітні параметри, такі як функції відбору, методи схрещування і мутації, щоб досягти бажаного результату;

3) підтримка багатьох типів даних, включаючи числа, логічні значення, переліки, масиви і об’єкти. Це дозволяє використовувати фреймворк для різних видів оптимізаційних задач.

До недоліків Jenetics зазвичай відносять:

1) фреймворк має деякі обмеження щодо масштабування, особливо для великих об’ємів даних.;

2) важкість зрозуміння, які саме функції варто використовувати в кожній конкретній ситуації, що може призвести до використання неефективних алгоритмів [20].

Виходячи з усього описаного раніше можна зробити висновок, що серед усіх доступних варіантів для мови програмування Java, фреймворк Jenetics відзначається своєю гнучкістю, простотою в використанні та високою продуктивністю. Jenetics пропонує кращу підтримку для багатокритеріальної оптимізації та включає багато еволюційних операцій та алгоритмів. Крім того, він має широкий спектр функцій для підтримки генетичного програмування та аналізу даних. Це робить Jenetics найкращим вибором для розробки генетичних алгоритмів в Java.

3.3 Вибір фреймворку для тестування

Тестування є важливим етапом розробки програмного забезпечення, що дозволяє перевірити коректність роботи окремих фрагментів коду. Однак, якщо ви використовуєте неправильний фреймворк для тестування, то результати тестування можуть бути некоректними або недостовірними. Вибір правильного фреймворку для тестування є важливим для забезпечення якості програмного забезпечення та забезпечення його надійності. Правильний вибір фреймворку дозволяє ефективно тестувати код, отримувати швидкі результати та забезпечувати простоту відлагодження. У цьому контексті, необхідно ретельно аналізувати можливості фреймворків та бібліотек для тестування, враховуючи їх особливості та переваги.

Деякі з доступних фреймворків та бібліотек для unit тестування для мови програмування Java включають такі:

а) JUnit – це один з найпопулярніших тестових фреймворків для Java. Він був створений з метою забезпечити автоматизоване тестування коду в проектах Java і дозволяє розробникам перевіряти правильність функціонування окремих частин програмного забезпечення, класів, методів або навіть всього проекту в цілому.

JUnit має безліч переваг, серед яких:

1) простота використання – він дуже простий у використанні і легко впроваджується в будь-який проект Java;

2) підтримка багатьох основних функцій тестування, таких як анотації @Before та @After для виконання певних дій перед та після тестування, методи assertEquals для порівняння очікуваних та фактичних результатів;

3) підтримка параметризованих тестів.

Незважаючи на свої переваги, JUnit також має деякі недоліки:

1) відсутність паралельного виконання – він не забезпечує підтримку паралельного виконання тестів, що може призвести до затримок в часі виконання великих тестових наборів;

2) JUnit не має вбудованої можливості тестування GUI, що може створювати проблеми при тестуванні деяких типів додатків [21].

В цілому, JUnit є потужним та легко використовуваним тестовим фреймворком, який може допомогти розробникам забезпечити якість свого програмного забезпечення. Його переваги значно переважають недоліки, і він є одним з найкращих виборів для тестування Java-проектів.

б) TestNG є одним з найпопулярніших тестових фреймворків для мови програмування Java, що використовується для автоматизованого тестування програмного забезпечення. Фреймворк має широкий набір функцій, що дозволяють розширити функціональність тестів та забезпечити високу якість коду.

До переваг TestNG зазвичай відносять:

1) гнучкість – фреймворк дозволяє легко створювати тести та змінювати їх при потребі;

2) TestNG підтримує багатопоточність, що дозволяє прискорити виконання тестів;

3) групування – фреймворк також має вбудовану можливість створення груп тестів, що дозволяє виконувати певну групу тестів замість усіх;

4) підтримка різних типів тестів, таких як функціональні, параметризовані, інтеграційні тощо. Фреймворк дозволяє легко налаштувати ці тести та виконувати їх у будь-якому порядку.

Серед недоліків TestNG можна виділити:

1) відносно складну конфігурацію тестів, що може бути важкою для початківців;

2) підтримка фреймворком тестування додатків на різних мовах програмування є обмеженою, що може стати проблемою для команд з розробників з різних фахових груп [22].

Загалом, TestNG є потужним та гнучким тестовим фреймворком, який дозволяє розширити функціональність тестів та забезпечити високу якість програмного забезпечення.

в) AssertJ є одним з найпопулярніших тестових фреймворків для мови програмування Java. Він надає зручний та зрозумілий інтерфейс для написання тестів, що дозволяє більш ефективно використовувати час розробки та знизити кількість багів у коді. AssertJ дозволяє зробити тестовий код більш читабельним та зрозумілим, що допомагає збільшити продуктивність команди розробників.

Одними з головних переваг AssertJ є:

1) його багатий набір методів для перевірки різних типів об'єктів, таких як строки, колекції, мапи, примітивні типи та інші;

2) AssertJ дозволяє використовувати лямбда-вирази та метод-специфікації для встановлення умов, що спрощує написання складних тестів.

Серед недоліків AssertJ виділяють:

1) складність – можуть бути складність вивчення та використання у порівнянні з іншими тестовими фреймворками, такими як JUnit і TestNG;

2) погана кросплатформенність – можуть виникати проблеми з виконанням тестів на деяких платформах або в середовищах, що не підтримують Java 8 та вище [23].

Порівнюючи JUnit з TestNG та AssertJ, можна сказати, що JUnit є одним з найбільш популярних тестових фреймворків у світі Java-розробки. Він має досить простий та зрозумілий інтерфейс, що робить його доступним для початківців у тестуванні. JUnit має широку підтримку в різних інтегрованих середовищах розробки та забезпечує простоту використання тестових механізмів для розробників.

Хоча TestNG також має свої переваги, такі як підтримка групування тестів та виконання тестів в декількох потоках, JUnit все ще залишається більш популярним тестовим фреймворком.

Щодо AssertJ, то він забезпечує багатий набір методів для перевірки різних типів даних, але він не є повноцінним тестовим фреймворком, а лише бібліотекою для написання більш зрозумілих тестів.

Отже, JUnit є найкращим вибором серед TestNG та AssertJ для тестування Java-програм з його зрозумілим інтерфейсом, широкою підтримкою та простотою використання.

3.4 Вибір бібліотеки для збору даних

В сучасному світі стало важливо не тільки розробляти високоякісне програмне забезпечення, а й забезпечити ефективність його роботи. Одним із інструментів для забезпечення ефективності є збір метрик ефективності в додатках.

Метрики дозволяють отримувати дані про роботу програми, її продуктивність та надійність. У цьому контексті виникає потреба в бібліотеках та фреймворках для збору метрик ефективності, що дозволяють відстежувати різноманітні показники додатка та аналізувати отримані дані.

У мові програмування Java існують багато бібліотек та фреймворків для збору метрик ефективності. Розглянемо найпопулярніші з них:

а) Micrometer – це фреймворк для збору метрик ефективності, який підтримує різні системи моніторингу. Він дозволяє легко і швидко зібрати метрики з різних джерел у вашому додатку, таких як системні ресурси, виконані запити, використання пам'яті та інші.

Перевагами Micrometer є:

1) простота використання. Він має чіткий API, який дозволяє легко додавати метрики в коді вашого додатку;

2) підтримка багатьох різних видів метрик, включаючи лічильники, гістограми та таймери;

3) забезпечення надійного механізму збору метрик у великих та складних додатках. Він включає механізм зберігання буферу та механізм відправки даних в моніторингову систему з пакетною обробкою, що дозволяє зменшити навантаження на мережу та збільшити швидкість збору метрик.

Однак, Micrometer має свої недоліки, такі як:

1) відсутність можливості моніторити стан системи, такий як навантаження процесора або кількість активних з'єднань;

2) складність – може бути важким для налаштування, якщо ви не знаєте, які метрики потрібно зібрати і які параметри вказувати [24].

б) Prometheus – це потужний фреймворк для збору метрик, який дозволяє моніторити стан системи в реальному часі. Він забезпечує збір даних про різні метрики, такі як лічильники, гістограми та розподілених метрик, що дозволяє аналізувати роботу системи з різних точок зору.

Серед переваг Prometheus виділяють:

1) можливість збирати дані з різних джерел, що дозволяє збирати дані з різних сервісів та систем, що працюють в різних середовищах;

2) підтримка динамічної конфігурації, що дозволяє додавати та видаляти цілі моніторингу під час роботи системи.

Однак, на додаток до переваг, є деякі недоліки у фреймворку Prometheus:

1) складність – він може бути складним у налаштуванні та налагодженні, зокрема, якщо використовується для збору метрик з декількох джерел;

2) збір метрик з великої кількості джерел може призвести до надмірної величини обсягу даних, що потребує додаткового масштабування та налаштування [25].

в) Java Microbenchmark Harness (JMH) – це фреймворк для тестування ефективності коду в Java. Він дозволяє проводити точні вимірювання швидкості виконання функцій та методів, а також допомагає уникнути помилок, пов'язаних з внесенням штучних оптимізацій для зменшення часу виконання.

Серед переваг JMH є наступні:

1) JMH надає велику кількість налаштувань, включаючи зміну числа потоків, часу очікування між операціями та налаштування параметрів Java Virtual Machine(JVM);

2) JMH має добре документовану API, яка дозволяє користувачам створювати власні тестові сценарії.

А серед недоліків називають такі:

1) складність налаштування та використання. Він вимагає деяких знань зі статистики та налаштування JVM, щоб досягти точності тестів та отримати надійні результати;

2) JMH може бути повільним виконуватися на великих проектах з великою кількістю коду, що може викликати проблеми з продуктивністю [26].

У цілому, JMH – потужний та точний фреймворк для тестування ефективності коду в Java. Хоча він може бути складним у використанні та

вимагає певних знань для налаштування, результати, отримані з його допомогою, можуть бути дуже цінними для покращення продуктивності вашого додатку.

г) Java Mission Control (JMC) – це фреймворк для збору метрик ефективності, розроблений компанією Oracle. JMC надає можливість збирати та аналізувати дані про виконання Java-додатків, що дозволяє знайти проблемні місця та покращити продуктивність додатку.

JMC має багато переваг серед яких:

1) Flight Recorder, який забезпечує збір даних під час виконання додатку. Flight Recorder збирає різноманітні дані про виконання додатку, такі як використання пам'яті, час виконання методів та транзакцій баз даних, та зберігає їх у форматі JFR (Java Flight Recorder);

2) JMC також містить інші інструменти для аналізу та візуалізації даних, такі як Mission Control Console та Memory Leak Detector. Mission Control Console надає користувачам можливість переглядати дані, зібрані Flight Recorder, та аналізувати їх у режимі реального часу. Memory Leak Detector допомагає знайти проблеми з витік пам'яті в додатку;

3) JMC поставляється разом з JDK та не вимагає жодних додаткових залежностей;

4) JMC дозволяє відстежувати профілювання в реальному часі, що дозволяє оперативно реагувати на проблеми з продуктивністю.

Одна JMC має також недоліки:

1) використання JMC може бути складним для користувачів, які не мають досвіду в роботі з фреймворками для збору метрик ефективності;

2) збір великої кількості даних під час виконання додатку може призвести до збільшення навантаження на систему [27].

ґ) Java VisualVM є безкоштовним інструментом для моніторингу та

профілювання додатків, написаних на мові програмування Java. Він дозволяє збирати різноманітні метрики про додаток, такі як CPU, пам'ять, введення-виведення, потоки, та багато інших.

Використовуючи Java VisualVM, розробники можуть визначити проблемні місця в своєму коді, виявити утечки пам'яті, аналізувати використання ресурсів та оптимізувати додаток для підвищення його продуктивності.

Java VisualVM має наступні переваги:

1) Java VisualVM має безліч корисних функцій, таких як створення знімків пам'яті, профілювання коду, моніторинг GC, аналіз стеку викликів та інші;

2) інструмент дозволяє підключати різноманітні плагіни для додаткового аналізу метрик та може бути інтегрований з іншими інструментами моніторингу та профілювання;

3) простий у використанні та має зручний інтерфейс, що дозволяє швидко отримувати необхідну інформацію

До недоліків Java VisualVM зазвичай відносять:

1) Java VisualVM може бути те, що він має обмежені можливості збору метрик в розподілених середовищах, таких як облака;

2) можуть виникнути певні складнощі при налагодженні інструменту на деяких операційних системах, але це залежить від конкретної конфігурації [28].

Загалом, Java VisualVM є корисним інструментом для моніторингу та профілювання додатків на мові програмування Java. Він дозволяє розробникам ефективно аналізувати продуктивність свого коду та вчасно виявляти та вирішувати проблеми, що дозволяє підвищити якість та швидкодію додатка.

Отже, використання Java VisualVM є розумним вибором для збору метрик в Java-програмах і допоможе покращити продуктивність та ефективність програмного забезпечення.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДУ РОЗВ’ЯЗАННЯ ІГРОВИХ ЗАДАЧ

4.1 Вибір ігрової задачі для тестування методу

Методи, що базуються на застосуванні генетичних алгоритмів – є потужний інструмент для розв'язання ігрових задач. Вони може бути застосовані для вирішення різноманітних головоломок, головокружливих завдань та ігрових складнощів За допомогою генетичних алгоритмів можна швидко та ефективно шукати рішення гри з великою кількістю варіантів. Головною перевагою використання генетичних алгоритмів є їхній потужний пошуковий алгоритм, який забезпечує високу швидкість і точність розв'язання завдань.

Гра «Судоку» – це логічна головоломка, яка полягає в заповненні 9x9 сітки цифрами від 1 до 9 таким чином, щоб кожне число зустрічалось тільки один раз в кожному рядку, стовпці та кожному з дев'яти 3x3 квадратів, які складають сітку. Ця гра зазвичай розглядається як гарний вибір для тестування генетичних алгоритмів завдяки своїй складності та кількості можливих комбінацій розв'язку гри, тому що наявність навіть двох дублікатів у рядках чи колонках може свідчити про неправильне розташування десяти чи навіть більше клітинок на ігровій поверхні.

Генетичний алгоритм може бути використаний для створення програми, яка розв'язує головоломку, шляхом генерації випадкових комбінацій та їх подальшої еволюції, що дозволяє знайти найкращий можливий розв'язок. Це дозволяє використовувати ігру «Судоку» як тестовий випадок для перевірки ефективності та точності роботи генетичного алгоритму. Більш того, успішне вирішення гри «Судоку» може бути використано як метрика ефективності для оцінки роботи генетичного алгоритму.

4.2 Реалізація програмного забезпечення для вирішення ігрової задачі

Базуючись на обраній мові програмування та фреймворку для генетичного алгоритму, було створено код для реалізації методу вирішення ігрової задачі «Судоку».

Як і для будь-якої задачі для вирішення генетичним алгоритмом, вона має реалізовувати фітнес-функцію, мета котрої є обчислення допасованості покоління після виконання їх модифікацій або на початку генерації для подальшого порівняння та вибору найбільш пристосованих поколінь.

Реалізація функції допасованості для вирішення задачі «Судоку» може мати декілька алгоритмів обчислення, але найбільш ефективним серед них є обчислення кількості дублікатів в рядках та колонках ігрової поверхні та подальша мінімізація їх. Ідеальним рішенням є кількість дублікатів, котра в кінцевому результаті буде дорівнювати нулю.

Сформулювати задачу пошуку функції допасованості можна наступним чином. Для *i*-тої рядки і *j*-тої колонки Судоку, що містить число *x*, визначаємо кількість його повторень на даному рядку і колонці. Це можна зробити, проходячи по всіх елементах в рядку і колонці, та перевіряючи, чи вони дорівнюють x. Якщо кількість повторень більше 1, тоді ми знаходимо дублікат. Формальний запис функції виглядає наступним чином:

$$(x_{i,j} \in R_i \cap \sum_{k=1}^{9} [x_{i,k} = x_{i,j}] > 1) \cup (x_{i,j} \in C_j \cap \sum_{k=1}^{9} [x_{k,j} = x_{i,j}] > 1) \quad (4.1)$$

де $x_{i,j}$ – число на кожному *i*-тому рядку та *j*-тій колонці;

R_i – множина чисел на *i*-тому рядку;

C_j – множина чисел на *j*-тій колонці;

$[x_{i,k}=x_{i,j}]$ – булева функція, шо повертає 1, якщо значення однакові та 0, якщо значення різні.

Реалізувавши дану функцію мовою програмування Java отримуємо код програми наведений у лістингу 4.1.

Лістинг 4.1 – Програмний код реалізації пошуку функції допасованості для ігрової задачі «Судоку»

```
override fun fitness(): Function<Riddle, Int> {
        return Function {
            val lists = it.array
            val rowDuplicates = lists.sumOf { it.size - it.distinct().size }
            val columnDuplicates = lists.rotate().sumOf { it.size - it.distinct().size }
            rowDuplicates + columnDuplicates + cellsDuplicates
        }
    }
    private fun Array<ByteArray>.rotate(): List<List<Byte>> {
        val lists = mutableListOf<List<Byte>>()
        for (x in indices) {
            val list = mutableListOf<Byte>()
            for (y in 0 until this[x].size) {
                list += this[y][x]
            }
            lists += list
        }
        return lists
    }
```

Наступним є крок реалізації механізму пошуку найбільш часто популярних значень для кожної клітинки ігрового поля з-поміж всіх поколінь. Така функція буде використання для виконання мутацій модою чи тенденцію.

Виражаючи це у вигляді математичної формули отримуємо:

$$mode(x) = argmax_{i=1}^{n} count(x_i) \quad (4.2)$$

де *x* – масив значень для певної клітинки ігрового поля;

n – кількість індивідів в поколінні.

Окрім цього, необхідно реалізувати функцію, яка буде вибирати значення, якщо воно підходить за мінімально допустимої кількості значень за допомогою коефіцієнта мінімальної допустимої кількості значень. Виразити цю функцію можливо за допомогою математичної формули 4.3

$$f(x) = mode(x) \geq k * n * minAllow \quad (4.3)$$

де *mode(x)* – значення самого популярного значення для клітинки ігрового поля;

k – значення кількості індивідів в поколінні;

n – значення кількості поколінь для обчислення моди або тренду;

minAllow – коефіцієнт мінімально допустимої кількості значень який лежить у межах в 0 до 1 включно.

Реалізовуючи (4.2) та (4.3) програмно отримуємо програмний код, що наведений у лістингу 4.2.

Лістинг 4.2 – Програмний код для обчислення моди та тренду для кожної клітинки ігрового поля.

```
    private fun calculateStatistics(size: Int, percentage: Double): Map<Pair<Int, Int>, Byte?> {
            if (history.size < size) {
                return emptyMap()
            }
            val entries = history.entries
```

Продовження лістингу 4.2

```
            val trendRiddles = (
                    entries
                        .takeIf { it.size == size }
                        ?:    entries.sortedBy    {    it.key
}.takeLast(size)
                    )
                .flatMap { it.value }
                .map { it.array.toCells() }
            val statistics: MutableMap<Pair<Int, Int>, Byte?>
= mutableMapOf()
            for (row in 0..8) {
                for (column in 0..8) {
                    if (!writableMatrix[row][column]) {
                        continue
                    }
                    statistics[row to column] = trendRiddles
                        .map { it[row][column] }
                        .fold(mutableMapOf<Byte,   Int>())   {
acc, it ->
                            acc.merge(it, 1, Int::plus)
                            acc
                        }.entries
                        .firstOrNull {
                            it.value >= totalAmountToSelect *
size * percentage
                        }?.key
                }
            }
            return statistics
        }
```

І останнім кроком для реалізації методу є власне створення мутатора для генетичного алгоритму на основі моди та тренду. Цей мутатор виконує

мутацію будь-якої клітинки, що не має попередньо встановленого значення під час генерації ігрової плитки. Код для мутатора наведений у лістингу 4.3

Лістинг 4.3 – Програмний код мутатора на основі моди та тренду

```
override fun mutate(
        chromosome: Chromosome<AnyGene<Riddle>>,
        p: Double,
        random: RandomGenerator
    ): MutatorResult<Chromosome<AnyGene<Riddle>>> {
        val riddle = chromosome.gene().allele()

        val arrays = riddle.array.toCells()

        val mutation = random.nextInt(2, 5)
        var mutationsLeft = mutation

        while (mutationsLeft != 0) {
            val cellIndex = random.nextInt(arrays.size)
            if (writableMatrix[cellIndex].count { it } < 2)
{
                continue
            }
            var                 swap1                 =
random.nextInt(arrays[cellIndex].size)
            while (!writableMatrix[cellIndex][swap1]) {
                swap1                                     =
random.nextInt(arrays[cellIndex].size)
            }
            val                 value                 =
populationEvolutionStatistics.getTrendValueByCellCoords(cellIn
dex, swap1)
                ?:
populationEvolutionStatistics.getMeanValueByCellCoords(cellInd
ex, swap1)
            value?.let { arrays[cellIndex].indexOf(it) }
```

Продовження лістингу 4.3

```
                    ?.let {
                        val buffer = arrays[cellIndex][swap1]
                        arrays[cellIndex][swap1]            =
arrays[cellIndex][it]
                        arrays[cellIndex][it] = buffer
                    }
                mutationsLeft--
            }
            return MutatorResult(
                chromosome.newInstance(ISeq.of(AnyGene.of {
                    DefaultRiddle(
                        riddle,
                        arrays.toCells()
                    )
                })),
                mutation
            )
        }
```

4.3 Оптимізація програмного забезпечення

Функціональне програмування є одним із стиль програмування, де великий акцент робиться на функціях. Це дозволяє писати більш декларативний та експресивний код, а також робити його більш безпечним та стійким до помилок. Однак, при використанні функціонального програмування в генетичних алгоритмах можуть виникнути проблеми зі створенням великої кількості об'єктів.

Генетичні алгоритми зазвичай потребують створення великої кількості об'єктів, щоб представити кожен потенційний розв'язок проблеми. При використанні функціонального програмування ця проблема може бути ще більшою, оскільки функції зазвичай повертають нові значення замість змінювати старі.

Це може призвести до збільшення використання пам'яті, зменшення продуктивності та затримки в часі виконання програми. Також можуть виникнути проблеми зі збиранням сміття та вирішенням проблем з пам'яттю.

Отже, при використанні функціонального програмування в генетичних алгоритмах потрібно бути уважним та розглядати можливі наслідки великої кількості об'єктів, що створюються. Треба розглянути різні варіанти оптимізації коду та вибрати той, який дозволить зменшити кількість створюваних об'єктів та збільшити продуктивність програми.

Виходячи з цього є необхідність у оптимізації коду таким чином, щоб зменшити кількість об’єктів, що будуть створенні, що у свою чергу зменшує кількість невикористовуваних об’єктів у пам’яті при кожній ітерації, що у свою чергу означає зменшення навантаження на пам’ять робочої станції та процесор через зменшення навантаження на аналіз та очистку пам’яті.

Для оптимізації роботи алгоритму були виконані наступні кроки:

– використання імперативний методу написання коду замість функціонального – основною проблемою функціонального програмування є створення на кожному кроку обробки цільової функції нових об’єктів замість їхньої модифікації при імперативному. Тому різниці між кількістю об’єктів при функціональному програмування у порівняні з імперативним може сягати декількох разів у кращому випадку або в n разів більше за кожний окремий функціональний крок, що виконується.

Таким чином позбавлення коду функціональних частин призводить до зменшення кількості занятої пам’яті, що у свою чергу призводить до зменшення навантаження процесора.

Звичайно, що такий підхід може призвести до більшої кількості похибок та помилок під час реалізації, але у випадку коли, код був написаний функціонально та покритий тестами після цього, то шанс на створення помилки зменшується, так як виконується створення не нової логіки, а відбувається простий рефакторинг коду.

– переписування коду з використанням простих та примітивних типів даних замість використання «розумних», гнучких та зрозумілих структур даних – під час написання звичайного коду використання складних структура даних є більш оптимальним кроком, тому що зменшується загальна кількість коду, що у свою чергу несе за собою спрощення логікі та зменшення шансу на створення помилок.

Окрім цього код стає зрозумілішим, а тому вносити правки та додатки стає простіше, але у випадку, коли код вже написаний та протестований, є можливість для його переписування без побоювань зламати вже готову функціональність.

4.4 Оптимізація віртуальної машини

Java Virtual Machine (JVM) – це віртуальна машина, що відповідає за виконання Java-програм. Вона забезпечує переносимість коду, безпеку та управління пам'яттю. Однак, в деяких випадках робота JVM може бути неоптимальною та призводити до зниження продуктивності.

Оптимізація роботи JVM є важливою задачею для розробників Java-програм. Використання різних параметрів JVM дозволяє підібрати оптимальні налаштування для певного додатку та забезпечити його ефективну роботу.

Найбільш важливими параметрами JVM є: -Xmx та -Xms. Параметр -Xmx визначає максимальний обсяг пам'яті, який може використовувати програма, а параметр -Xms – початковий обсяг пам'яті, який виділяється для програми при запуску. Підбір оптимальних значень цих параметрів дозволяє забезпечити ефективне використання пам'яті та запобігти переповненню кеша.

Ще один важливий параметр JVM -XX:+AggressiveOpts. Включення цього параметра дозволяє використовувати різні оптимізації на рівні коду,

такі як заміна виклику методу на пряме виконання коду та заміна виклику віртуального методу на пряме виконання коду.

Параметр -XX:CompileThreshold – визначає кількість викликів методу, після яких він буде скомпільований в машинний код. Зменшення цього параметра дозволяє прискорити запуск програми, але може призвести до збільшення обсягу пам'яті, що використовується для компіляції коду.

Параметри -XX:+UseStringDeduplication дозволяє JVM автоматично знаходити та видаляти дублюючіся рядки, що допомагає зменшити використання пам'яті.

Опціональний параметр -XX:MaxInlineLevel – цей параметр дозволяє визначити максимальну глибину вкладеності методів, які можуть бути вбудовані в інші методи. Зменшення цього параметра може допомогти зменшити використання пам'яті та покращити продуктивність.

При розробці програмного забезпечення, особливо тих, які мають високі вимоги до продуктивності, важливо розглядати різні аспекти оптимізації роботи JVM. Один з таких аспектів – вибір збиральника сміття.

У старіших версіях JVM використовувалися прості збірники сміття, які мали деякі обмеження та недоліки. Одними з найпоширеніших видів були збірники сміття на основі поколінь (Serial Collector, Parallel Collector та Concurrent Mark-Sweep (CMS) Collector).

Проте ці старі збірники сміття мали свої недоліки. Вони потребували додаткового часу для копіювання об'єктів та виконання маркування та видалення, що може вплинути на продуктивність додатка. Також, їх обмежена адаптивність та недостатня оптимізація можуть призводити до проблем зі збіркою сміття на великих обсягах пам'яті.

Збиральник сміття відповідає за автоматичне видалення об'єктів, які більше не використовуються програмою, з пам'яті. В JVM існує декілька збиральників сміття, кожен з яких має свої переваги та недоліки. Вибір збиральника залежить від вимог до продуктивності та інших факторів.

Один з найпоширеніших збиральників сміття – G1 (Garbage First). G1 є компромісом між двома іншими збиральниками сміття: CMS (Concurrent Mark Sweep) та Parallel Scavenge. Він працює в два етапи: він розбиває пам'ять на різні регіони та виконує збірку сміття на тих регіонах, які мають найбільшу кількість незбережених об'єктів. Це дозволяє зменшити час зборки сміття та зменшити вплив зборки на продуктивність програми.

Інший збиральник сміття – Shenandoah – працює на основі принципу операційного переміщення об'єктів. За допомогою цього збиральника можна зменшити час паузи в програмі, але це може призвести до додаткового навантаження на процесор.

Ще один збиральник сміття – ZGC (Z Garbage Collector) був розроблений для роботи з дуже великими обсягами пам'яті. Він дозволяє використовувати до 16 терабайтів пам'яті та зменшує час зборки сміття до 10 мілісекунд.

Таким чином, оптимізація роботи JVM може забезпечити покращення продуктивності, зниження часу виконання та економію ресурсів. Однак, необхідно пам'ятати, що оптимізація параметрів JVM є складним інженерним процесом, який вимагає вміння розуміти взаємозв'язки між різними параметрами та їх вплив на роботу JVM.

У цьому контексті існує безліч інструментів та бібліотек, що допомагають здійснити оптимізацію роботи JVM. Для прикладу, інструменти профілювання, такі як Java Flight Recorder і Java Mission Control, можуть допомогти знайти головні точки генерації сміття та збільшення часу виконання.

У кінцевому підсумку, оптимізація роботи JVM є важливою задачею, яка може мати значний вплив на продуктивність додатків. Відповідне налаштування параметрів JVM може допомогти забезпечити оптимальну роботу програмного забезпечення, що має значення в галузі високонавантажених додатків, таких як серверні додатки, веб-додатки та інші.

5 ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА ЗАПРОПОНОВАНОГО МЕТОДУ РОЗВ'ЯЗАННЯ ІГРОВИХ ЗАДАЧ

5.1 Опис експерименту та показників оцінки ефективності методу

Для тестування роботи методу, що базується на генетичному алгоритмі на основі тренду та моди та його порівняння з роботою звичайного алгоритму заміною були запущено два випробування з певним списком умов:

– генерується повністю заповнена ігрова поверхня 9 на 9;

– генерується ігрова поверхня з певною кількістю пустих комірок в залежності від рівня складності;

– ігрова поверхня довільно заповнюється унікальними значеннями для кожної підповерхні 3 на 3;

– запускається 100 тестів окремо для звичайного алгоритму та для алгоритму на основі моди та тренду;

– одночасно працюючими можуть бути 5 тестів для кожного з видів алгоритму через обмежені ресурси комп'ютера;

– кожен тест є обмеженим 48 годинами та завершується невдало у разі перевищення часу роботи;

– кожні 5 хвилин кожен працюючий тест надає дані про кількість дублікатів, час роботи, кількість згенерованих поколінь та вигляд ігрової поверхні;

– окремо збираються метрики використаних ресурсів пам'яті та процесорного навантаження.

В результаті виконання тестування були зібрані базові метрики роботи алгоритмів, а саме час, що для тестування звичайного алгоритму заміною потребувало близько 23 діб, в той час як для алгоритму на основі моди та тенденції зайняв близько 18 діб.

5.2 Аналіз показника сходження для генетичного алгоритму

Результатом тестування обох алгоритмів є набір метрик, які дають яскраве представлення стосовно якості їхньої роботи.

Метрики сходження обох алгоритмів відображені на рисунку 5.1.

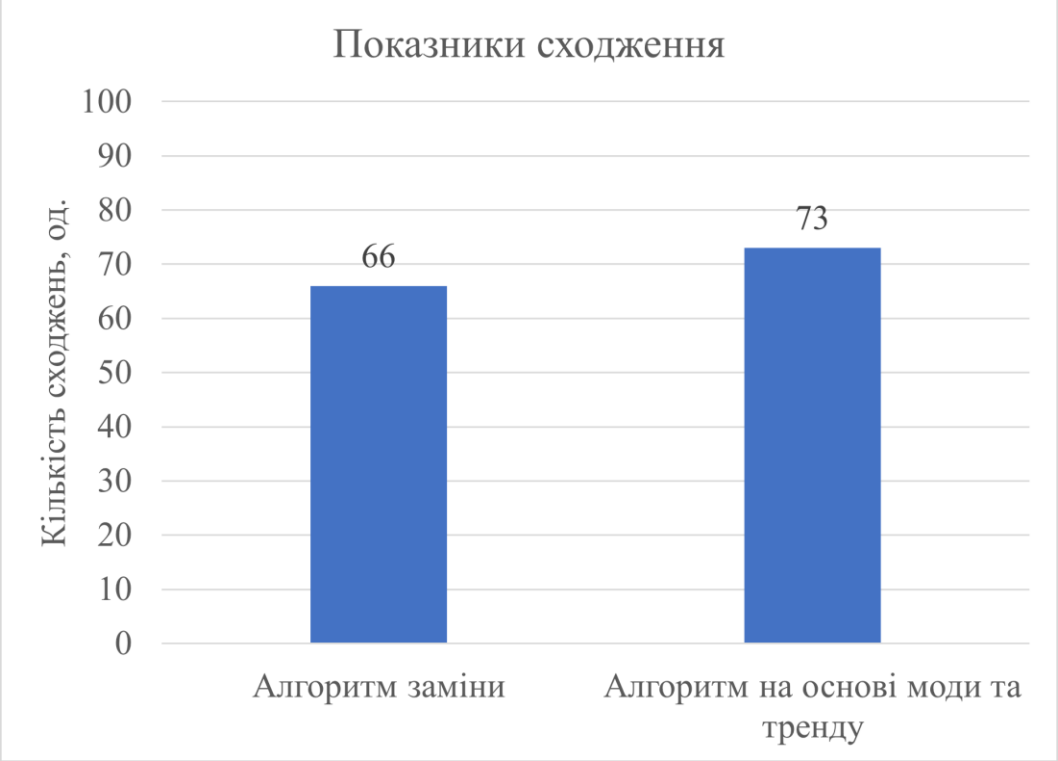

Рисунок 5.1 – Показники сходження алгоритмів

Виходячи з отриманих результатів можна зробити висновок, що алгоритм на основі моди та тренду має більш високий показник сходження, аніж звичайний алгоритм заміни, за рахунок того, що мутація відбувається не лише за рахунок випадкових процесів заміни будь-яких комірок, а й з використанням попередніх обчислених величин.

Такими чином вирахувавши значення покращення роботи алгоритмом на основі моди та тренду, можна зробити висновок, що цей алгоритм показує приріст значення в 10.1%.

5.3 Аналіз показника швидкості роботи для генетичного алгоритму

В результаті тестування та аналізу метрик швидкості роботи для обох алгоритмів були отримані наступні результати позначені в табл. 5.2.

Таблиця 5.1 – Метрики швидкості роботи алгоритмів

	Часові показники, хв.		
	Min	Avg	Max
Заміна	0.05	1457	2877
Мода та тренд	0.13	1158	2838

Візуально часові показники обох алгоритмів для мінімального, середнього та максимального значення зображені на рисунку 5.1.

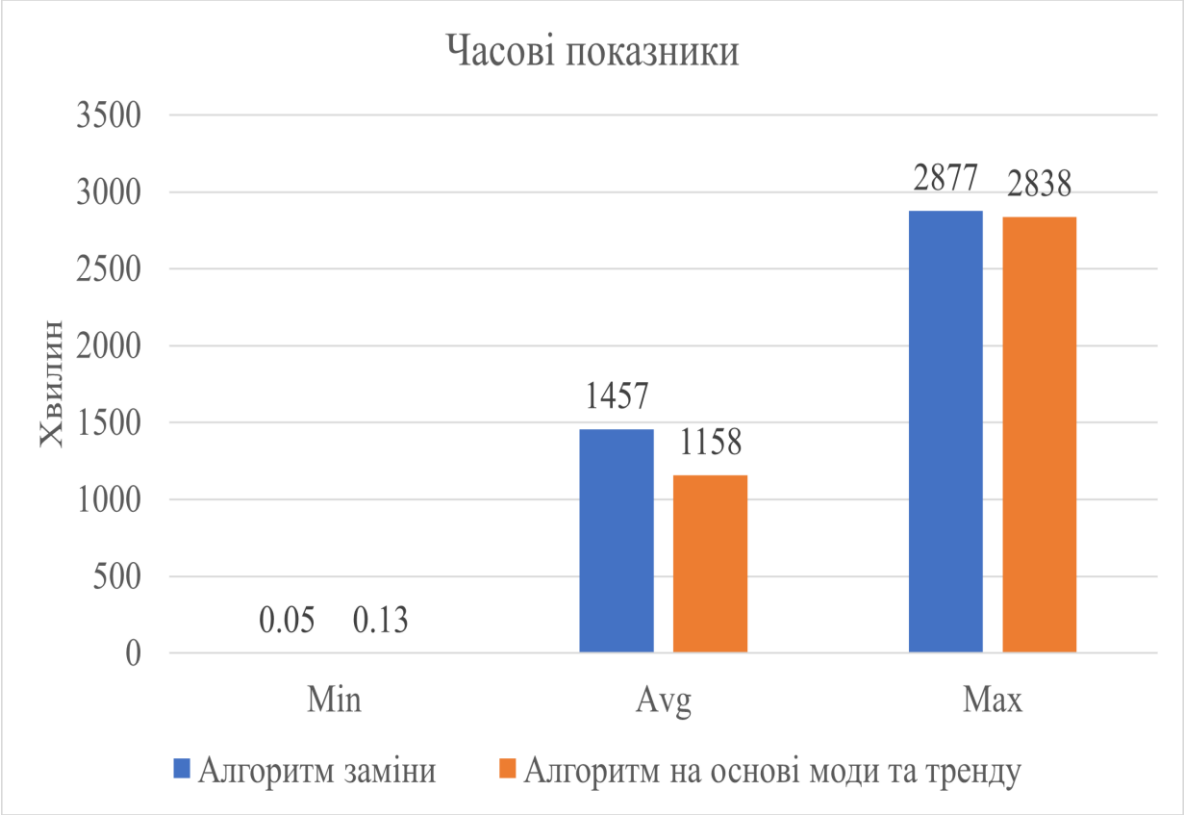

Рисунок 5.2 – Часові показники роботи алгоритму

На основі отриманого результату виходить, що часові показники для алгоритму заміни є дещо кращими з мінімальної точки зору, але ця різниця є досить малою і не відображає реальну різницю роботи.

В той самий час показник максимального часу роботи для алгоритму на основі моди та тренду є дещо кращим, а саме на 1.2%.

Середнє ж значення для алгоритму на основі моди та тренду є набагато кращим і складає аж 25.8%.

5.4 Аналіз показників використовуваних ресурсів

Під час проведення тестування обох алгоритмів були зібрані метрики споживання ресурсів комп’ютера, а саме:

– обсяг пам’яті, шо використовується;

– відсоток використання процесора;

– метрика кількості сгенерованих поколінь, що може бути використаний як відносний показник кількості ресурсів, що затрачаються для кожного покоління.

Окрім цього, необхідно зауважити, що під час тестування виконувалась робота одразу 5 екземплярів генетичного алгоритму, а тому при аналізі значень, відображених нижче, необхідно брати до уваги, що реальна кількість ресурсів необхідних для виконання одного генетичного алгоритму будуть значно нижчими.

Також, важливо зауважити, що тестування відбувалося, на комп’ютері, що мають такі характеристики:

– процесор має 6 реальних ядер та 12 віртуальних при частості 2.6 гігагерц;

– комп’ютер має 32 гігабайти вбудованої пам’яті, але віртуальна машина JVM була обмежена 16 гігабайтмами.

Результат використаної пам’яті відображений у табл. 5.2.

Таблиця 5.2 – Використання пам’яті алгоритмами

	Обсяг пам’яті що використовується, Мб		
	Min	Avg	Max
Заміна	3628	6988	13816
Мода та тренд	4787	8496	17179

Також графік пам’яті зображений на рисунку 5.3.

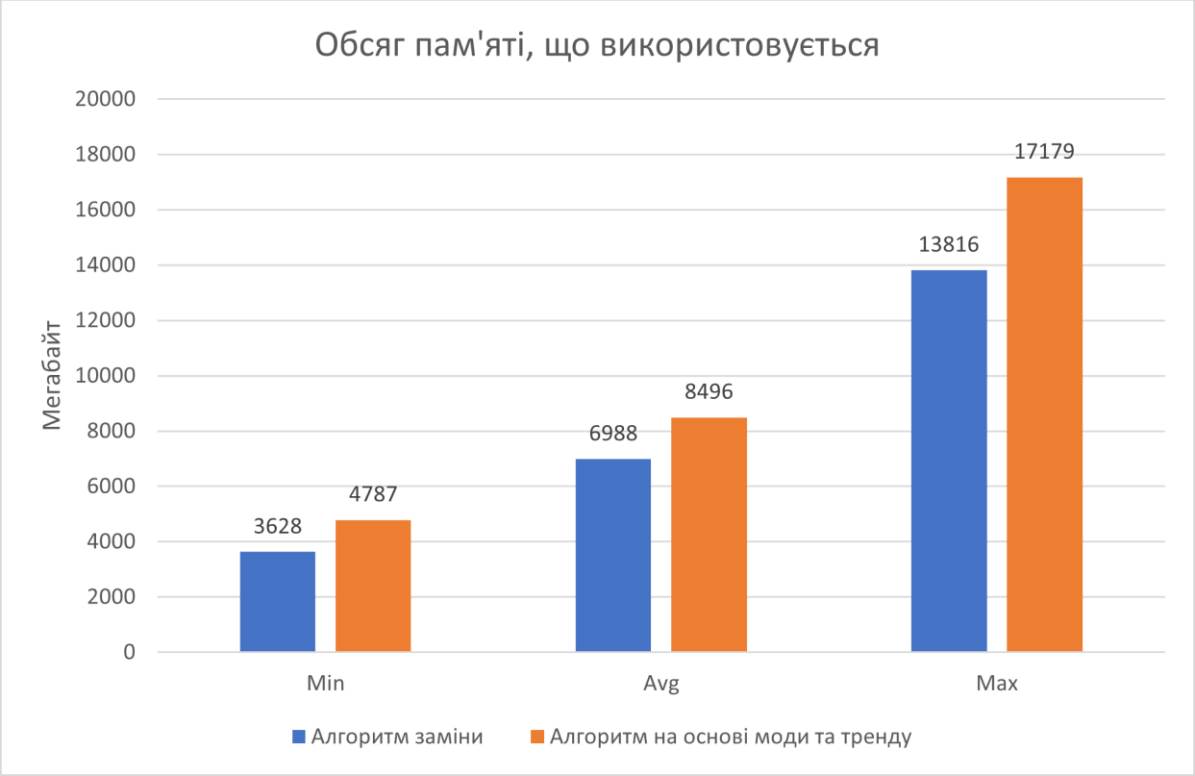

Рисунок 5.3 – Графік використання обсягів пам’яті

Аналізуючи дані по обсягу пам’яті стає очевидним, що алгоритм на основі моди та тренду використовує значно більше ресурсів комп’ютера по всім показникам, як за мінімальним – на 32%, за середнім – на 22% та так і за максимальним – на 24%.

Окрім цього, дані, щодо відсотку процесора, що був використаний зображені у табл. 5.3.

Таблиця 5.3 – Використання процесора алгоритмами

	Відсоток процесора що використовується, %		
	Min	Avg	Max
Заміна	21.4	35.75	82.6
Мода та тренд	22.3	44.9	98.9

Також, ці значення відображені графіком на рисунку 5.4.

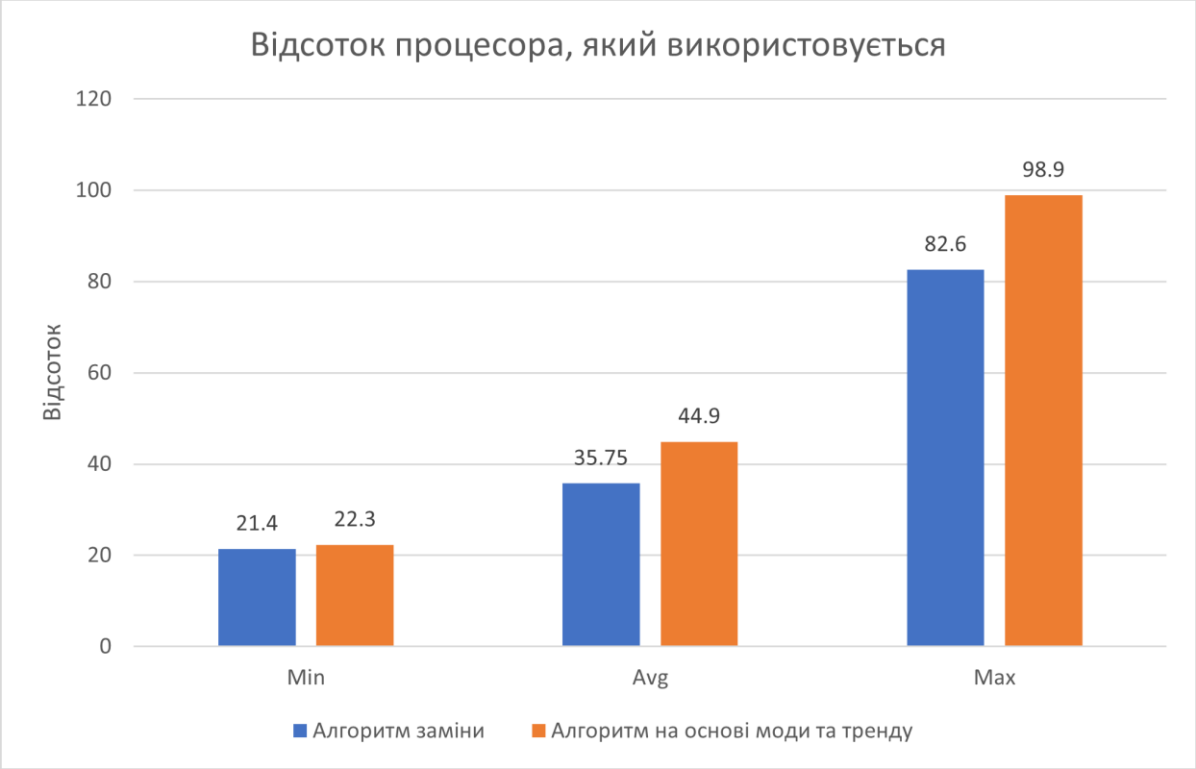

Рисунок 5.4 – Графік використання процесора

Аналізуючи обсяги використання процесора, можна, знову ж, зробити висновок, що ресурси процесора використовуються більше алгоритмом на основі моди та тенденції, а саме мінімальне значення зросло на 4%, середнє значення на 25%, а максимальне значення на 20%.

Також, були зібрані дані, щодо кількості поколінь необхідних для вирішення ігрових задач для обох алгоритмів.

Дані, щодо кількості поколінь приведені у табл. 5.4.

Таблиця 5.4 – Кількість поколінь для вирішення задачі

	Кількість поколінь, од.		
	Min	Avg	Max
Заміна	17	2296402	9637638
Мода та тренд	43	722120	2258288

Значення відображені також на рисунку 5.5.

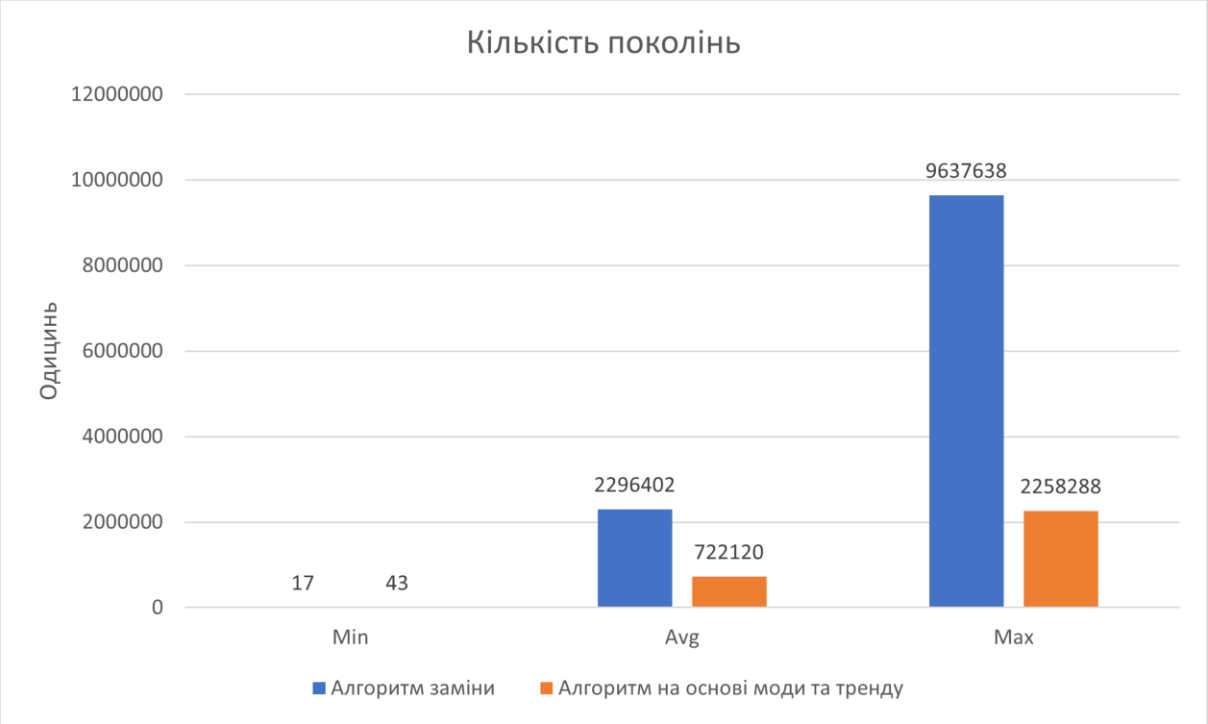

Рисунок 5.5 – Графік кількості поколінь для вирішення задачі

Базуючись на цих значення можна зробити висновок, що необхідна кількість поколінь для вирішення задачі «Судоку» відрізняється значно, а саме мінімальна кількість використана більше алгоритмом на основі моди та тенденції більше на 152%, у той час як середнє значення більше для алгоритму заміни на 218%, так само як і максимальне значення значно більше для того ж алгоритму заміни 327%.

5.5 Аналіз переваг та недоліків методу

Базуючись на отриманих результатах можна зробити висновок, що використання алгоритму на основі моди та тенденції дозволяє досягнути значно кращих значень сходження при меншій кількості поколінь.

Це важливо для вирішення різноманітних задач. Для складних задач, використання генетичних алгоритмів може бути дуже часо- та ресурсоємким процесом, тому покращення за рахунок створення власного алгоритму на основі моди та тенденції дає наступне:

– непряме покращення швидкодії через меншу потребу в використанні більшої кількості одночасно запущених екземплярів генетичного алгоритму;

– значно підвищує шанс на знаходження правильного значення;

– пряме покращення швидкості роботи в середньому і в максимумі, що відображено метриками швидкодії.

Будь-який алгоритм, що використовується має певний набір недоліків з якими необхідно миритися, якщо переваги алгоритму є більш значущими для користувача.

Алгоритм на основі моди та тенденції має наступні недоліки:

– використання більшого обсягу пам’яті обчислювальної машини, що у свою чергу значить, що мається потреба в загальних більших обсягах пам’яті у порівняні з алгоритмом заміни;

– використання більшого обсягу процесорного часу через наявність необхідності до виконання більшої кількості обчислень, так через використання більшого числа об’єктів системи, а то наявний збиральник сміття певної мови програмування повинен працювати частіше;

– загальна низька швидкість генерації поколінь через додаткові необхідні обчислення статистики та через потребу процесора виконувати інші задачі пов’язані зі збиранням сміття.

Важливо відзначити, що реалізація алгоритму може бути реалізована більш оптимально за рахунок більшої оптимізації основних слабких частин. Може бути вирішена проблема великих обсягів пам’яті необхідних для алгоритму за рахунок використання пулів об’єктів, що має в середньому підвищити обсяги пам’яті через резервацію, але у той самий час максимальний обсяг пам’яті значно зменшиться, що у свою чергу значить, що процесор буде мати більше часу на виконання обчислень, а не на виконання очищення пам’яті від сміття.

5.7 Аналіз отриманих результатів

Підводячи підсумки на основі отриманих результатів роботи можна зробити висновок, що метод генетичного алгоритму на основі моди та тенденції має значні покращення з точки зору сходження результатів у порівняні зі звичайним алгоритмом заміни.

Також, запропонований метод на основі моди та тенденції показує значно більшу ефективність використання нових поколінь у порівнянні зі звичайними методами.

Це означає, що алгоритм може бути впроваджений у повсякденну експлуатацію при реалізації генетичних алгоритмів та використаний при необхідності знаходження результату, коли можлива кількість одночасно запущених екземплярів алгоритму є обмеженою, а більше споживання ресурсів обчислювальної машини для одно екземпляру не є критичним.

Окрім цього, даний алгоритм може бути додатково оптимізований та покращений за рахунок:

– використання пулів об’єктів;

– підбору більш точних параметрів;

– використовуючи додаткові особливості певних мов програмування, фреймворків, а також інших інструментів.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було розроблено власний метод на основі генетичного алгоритму для розв’язання ігрових задач. Головною метою роботи є покращення якісних та кількісних показників у порівнянні з іншими розглянутими алгоритмами при використанні одних і тих самих часових показників.

Проаналізована предметна область та виділені основні проблеми, з якими зіткаються розробники при реалізації генетичних алгоритмів.

Також були розглянуті існуючі методи, що базуються на використанні генетичних алгоритмів та був виділений список переваг та недоліків задля подальшого вирішення проблем у вже існуючих рішеннях.

Був спроектований мутатор на основі моди та тенденції, а також був доданий коефіцієнт мінімально допустимої кількості значень.

Перед виконанням реалізації, був проведений аналіз мов програмування, бібліотек та фреймворків, а також інших інструментів, що могли допомогти у досягненні рішення.

Було реалізовано власний метод на основі моди та тенденції на мові програмування Java з використання фреймворку Jenetics.

Додатково, було виконано оптимізацію генетичного алгоритму з метою покращення показників ефективності роботи та задля зменшення кількості ресурсів необхідних для роботи алгоритму.

Обчислені показники сходження, швидкодії та використання ресурсів обчислювальної машини в ході тестування роботи методу на прикладі гри «Судоку».

На основі аналізу метрик, були виділені переваги та недоліки розробленого методу та був зроблений висновок щодо його ефективності та працездатності, який підтвердив ефективність його використання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Muthee A. The Basics of Genetic Algorithms in Machine Learning. URL: https://www.section.io/engineering-education/the-basics-of-genetic-algorithms-in-ml/ (Last accessed: 05.04.2023)
2. Генетичні алгоритми. Ключові поняття і методи реалізації. URL: http://www.znannya.org/?view=ga_general (дата звернення: 05.04.2023)
3. Каверін А. М. Дослідження та розробка генетичних алгоритмів розв'язання ігрових задач. *27-й Міжнародний молодіжний форум «Радіоелектроника та молодь XXI століття»*: Зб. матеріалів форуму. Т.6 Конференція «Інтелектуальні інформаційні системи», м. Харків, 10-12 трав. 2023р. Харків: ХНУРЕ. 2023. С. 218–219.
4. Sheppard C. Genetic Algorithms with Python. Clinton Sheppard, 2018. 312 p.
5. Introduction to Evolutionary Computing. URL: https://link.springer.com/book/10.1007/978-3-662-44874-8 (дата звернення: 06.04.2023)
6. Morgan B. Unit 3) Genetic algorithms (Part2) Advanced topics. URL: https://towardsdatascience.com/unit-3-genetic-algorithms-part-2-advanced-topics-a24f5be287d5 (Last accessed: 06.04.2023)
7. E G. D. Genetic algorithms in search, optimization, and machine learning. Reading, Mass : Addison-Wesley Pub. Co., 1989. 412 p.
8. Deterministic system. URL: https://www.daviddarling.info/encyclopedia/D/deterministic_system.html (Last accessed: 07.04.2023)
9. Ross S. M. Stochastic processes. 2nd ed. New York : Wiley, 1996. 510 p.
10. Kramer O. Genetic Algorithm Essentials. Springer, 2017. 92 p.

11. Мови програмування: для чого потрібні. URL: https://lviv.logos-academy.com/movy-programuvannya-yaku-obraty-i-z-chogo-rozpochaty-vyvchennya (дата звернення: 08.04.2023)

12. Fusco M., Mycroft A., Urma R.-G. Modern Java in Action: Lambdas, streams, functional and reactive programming. Manning Publications, 2018. 592 p.

13. Переваги мови програмування Java. URL: https://np.pl.ua/2021/03/perevahy-movy-prohramuvannia-java/ (дата звернення: 11.04.2023)

14. Zobenica D. Introduction to Genetic Algorithms in Java. URL: https://stackabuse.com/introduction-to-genetic-algorithms-in-java/ (Last accessed: 12.04.2023)

15. Sierra K., Bates B. Head First Java, 2nd Edition. O'Reilly Media, Inc., 2005. 656 p.

16. What is Java Garbage Collection? How It Works, Best Practices, Tutorials, and More. URL: https://stackify.com/what-is-java-garbage-collection/ (дата звернення: 14.04.2023)

17. Page S. E., Miller J. H. Complex Adaptive Systems: An Introduction to Computational Models of Social Life (Princeton Studies in Complexity). Princeton University Press, 2007. 284 p.

18. Genetic algorithms. URL: https://commons.apache.org/proper/commons-math/userguide/genetics.html (Last accessed: 15.04.2023)

19. ECJ. URL: https://cs.gmu.edu/~eclab/projects/ecj/ (Last accessed: 15.04.2023)

20. Jenetics. URL: https://jenetics.io/ (Last accessed: 15.04.2023)

21. Hunt A., Thomas D., Langr J. Pragmatic Unit Testing in Java 8 with JUnit. Pragmatic Bookshelf, 2015. 236 p.

22. TestNG. URL: https://testng.org/doc/ (Last accessed: 17.04.2023)

23. AssertJ - fluent assertions java library. URL: https://assertj.github.io/doc/ (Last accessed: 17.04.2023)

24. Micrometer. URL: https://micrometer.io/docs (Last accessed: 19.04.2023)

25. Brazil B. Prometheus: Up & Running: Infrastructure and Application Performance Monitoring. O'Reilly Media, 2018. 386 p.

26. Microbenchmarking with Java. URL: https://www.baeldung.com/java-microbenchmark-harness (Last accessed: 19.04.2023)

27. Shcherbakov A. Monitoring Java Applications with Flight Recorder. URL: https://www.baeldung.com/java-flight-recorder-monitoring (Last accessed: 19.04.2023)

28. Java VisualVM. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/visualvm/ (Last accessed: 20.04.2023)

ДОДАТОК А

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	60 с.
	Інші документи	
2.	Презентаційні матеріали	18 плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Каверін А. М.			Дослідження та розробка генетичних алгоритмів розв’язання ігрових задач
Перевірив		Петров К. Е.			
Н.контр.		Малєєва І.А.			
Затв.		Філатов В.О.			

Шифр групи	Код напр./спец.
СШІм-21-1	122
ХНУРЕ кафедра ШІ	