ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Метод оптимізації програмного коду на основі генетичного алгоритму»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Владислав ШОРОБУРА
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-62
Владислав ШОРОБУРА

Керівник: Богдан ХУДІК
доктор філософії (PhD)

Рецензент:

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Шоробурі Владиславу Вікторовичу

1. Тема кваліфікаційної роботи: «Розробка методу оптимізації програмного коду на основі генетичного алгоритму»

керівник кваліфікаційної роботи Богдан ХУДІК, доктор філософії (PhD), старший вилкадач кафедри ІПЗ
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024 р. № 320.

2. Строк подання кваліфікаційної роботи «17» грудня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, методи оптимізації коду, генетичний алгоритм.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідження методів оптимізації програмного коду на основі генетичного алгоритму
2. Аналіз технологій еволюційних алгоритмів для оптимізації програмного коду

3. Розробка та тестування методу оптимізації коду на основі генетичного алгоритму

5. Перелік ілюстративного матеріалу: *презентація*
1. Мета, об’єкт та предмет дослідження
2. Порівняльний аналіз методів оптимізації програмного коду
3. Модель оптимізації програмного коду
4. Модифікований метод оптимізації коду (діаграма діяльності)
5. Алгоритм оптимізації (блок-схема)
6. Переваги інноваційного підходу
7. Практичний результат
8. Інтерфейс користувача (початковий екран)
9. Візуалізація процесу оптимізації
10. Висновки
11. Публікації та апробація роботи

6. Дата видачі завдання«16» жовтня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.10-23.10.2024	
2	Вивчення матеріалів для аналізу методів оптимізації коду	24.10-03.11.2024	
3	Дослідження основних принципів генетичних алгоритмів	04.11-13.11.2024	
4	Аналіз використання генетичних алгоритмів для оптимізації коду	14.11-20.11.2024	
5	Дослідження архітектури оптимізації коду	21.11-28.11.2024	
6	Створення методу оптимізації коду на основі генетичного алгоритму	28.11-02.12.2024	
7	Оформлення роботи: вступ, висновки, реферат	03.12-04.12.2024	
8	Розробка демонстраційних матеріалів	05.12-06.12.2024	
9	Попередній захист роботи	07.12-17.12.2024	

Здобувач вищої освіти _____ Владислав ШОРОБУРА
(підпис)

Керівник
кваліфікаційної роботи _____ Богдан ХУДІК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 103 стор., 28 рис., 17 табл., 23 джерел.

Мета роботи – оптимізувати параметри програмного коду за рахунок використання генетичного алгоритму.

Об'єкт дослідження – процес створення програмного коду на етапі рефакторингу

Предмет дослідження – методи оптимізації параметрів програмного коду з застосуванням генетичних алгоритмів

Короткий зміст роботи:

У роботі досліджено основні принципи генетичних алгоритмів та їх застосування для автоматизованої оптимізації параметрів програмного коду. Показано переваги використання генетичних алгоритмів для задач оптимізації, а також проведено аналіз сучасних підходів і досліджень у цій галузі. Окреслено завдання оптимізації коду та обґрунтовано вибір відповідного датасету для тестування.

Запропоновано архітектуру методу оптимізації, який передбачає використання генетичних алгоритмів для пошуку ефективних рішень. Реалізовано генетичний алгоритм із залученням обраного програмного інструментарію та протестовано його ефективність на даних. Встановлено, що запропонований метод оптимізації покращує продуктивність коду порівняно з традиційними методами. Виконано аналіз продуктивності оптимізованого коду та проведено порівняння з іншими методами оптимізації.

КЛЮЧОВІ СЛОВА: ГЕНЕТИЧНИЙ АЛГОРИТМ, ОПТИМІЗАЦІЯ ПРОГРАМНОГО КОДУ, ЕВОЛЮЦІЙНИЙ АЛГОРИТМ, ПРОДУКТИВНІСТЬ, ЕФЕКТИВНІСТЬ, МЕТОД ОПТИМІЗАЦІЇ, АВТОМАТИЗАЦІЯ.

ABSTRACT

The textual part of the qualification work for obtaining a master's degree: 103 pages, 28 pictures, 17 tables, 23 sources.

Objective of the work – to optimize software code parameters using a genetic algorithm.

Object of study – the process of creating software code at the refactoring stage.

Subject of study – methods of optimizing software code parameters using genetic algorithms.

Summary of the work:

The study explores the fundamental principles of genetic algorithms and their application for the automated optimization of software code parameters. It demonstrates the advantages of using genetic algorithms for optimization tasks and analyzes current approaches and research in this field. The tasks of code optimization are outlined, and the choice of an appropriate dataset for testing is justified.

An architecture for an optimization method is proposed, incorporating genetic algorithms to seek efficient solutions. The genetic algorithm was implemented using selected software tools and tested for effectiveness on sample data. The findings indicate that the proposed optimization method improves code performance compared to traditional methods. An analysis of the optimized code's performance was performed, along with a comparison with other optimization techniques.

KEYWORDS: GENETIC ALGORITHM, PROGRAM CODE OPTIMIZATION, EVOLUTIONARY ALGORITHM, PERFORMANCE, EFFICIENCY, OPTIMIZATION METHOD, AUTOMATION.

ЗМІСТ

ВСТУП ... 9
1 ОСОБЛИВОСТІ ВИКОРИСТАННЯ ГЕНЕТИЧНОГО АЛГОРИТМУ ... 12
1.1 Теоретичні основи генетичних алгоритмів ... 12
1.2 Використання генетичних алгоритмів для оптимізації програмного коду ... 15
1.3 Огляд попередніх досліджень та рішень ... 19
2 ПРОЕКТУВАННЯ МЕТОДУ ОПТИМІЗАЦІЇ ... 22
2.1 Постановка завдання оптимізації коду ... 22
2.2 Вибір і обґрунтування датасету ... 25
2.3 Архітектура методу оптимізації ... 27
3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО КОДУ ... 37
3.1 Програмні засоби та інструменти ... 37
3.2 Реалізація генетичного алгоритму ... 39
3.3 Використання датасету для тестування ... 42
4 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ ... 45
4.1 Особливості реалізації та функціонування програми ... 45
4.2 Аналіз продуктивності оптимізованого коду ... 55
4.3 Порівняння з іншими методами оптимізації ... 58
ВИСНОВКИ ... 63
ПЕРЕЛІК ПОСИЛАНЬ ... 66
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ ... 68
ДОДАТОК Б. АЛГОРИТМ ГЕНЕТИЧНОЇ ОПТИМІЗАЦІЇ ... 75
ДОДАТОК В. ДІАГРАМА ПОСЛІДОВНОСТЕЙ ... 76
ДОДАТОК Г. ДІАГРАМА ПОСЛІДНОВНОСТЕЙ ... 77
ДОДАТОК Ґ. ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ ... 78
ДОДАТОК Д. ДІАГРАМА ДІЯЛЬНОСТІ ... 79
ДОДАТОК Е. ТЕСТУВАННЯ ПРОГРАМИ ... 80
ДОДАТОК Є. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ ... 84

ВСТУП

Оптимізація програмного коду є однією з важливих проблем сучасного програмування, оскільки ефективність програмного забезпечення (ПЗ) безпосередньо впливає на його швидкодію, ресурсоємність та масштабованість. В умовах зростання обсягів даних, підвищення складності програмних систем та необхідності ефективного використання обчислювальних ресурсів зростає значення методів оптимізації, що здатні покращити продуктивність без втрати функціональності. Серед таких методів важливе місце займають генетичні алгоритми (ГА), які завдяки своїй здатності знаходити глобальні екстремуми та працювати з великою кількістю варіантів рішень, демонструють значний потенціал для оптимізації параметрів коду.

Актуальність теми. Генетичні алгоритми активно застосовуються у задачах оптимізації завдяки своїй здатності імітувати природний процес еволюції, що дозволяє генерувати й комбінувати різні рішення, поступово вдосконалюючи їх на основі критерію придатності. В галузі оптимізації параметрів коду це дає змогу створювати та обирати варіанти коду з найвищою продуктивністю. Проте, незважаючи на попередні дослідження, питання ефективного застосування ГА для автоматичної оптимізації параметрів коду залишається відкритим. Існуючі підходи зазвичай сфокусовані на оптимізації вузьких аспектів (наприклад, продуктивності обраного фрагменту коду), але не охоплюють комплексну оптимізацію на рівні всієї системи, що є актуальним завданням для України з огляду на зростаючі вимоги до ефективного ПЗ у різних галузях.

Аналіз останніх досліджень і публікацій. У світовій практиці генетичні алгоритми використовуються для вирішення різних задач оптимізації, включаючи компіляцію та налаштування ПЗ. Зокрема, автори, як-от Мітчелл та Голдберг, дослідили теоретичні основи ГА, тоді як інші, такі як Люкас і Тернер, розглядали їхнє застосування в конкретних галузях ПЗ. Проте більшість досліджень зосереджені на прикладних аспектах, а не на універсальних методах оптимізації

параметрів коду. В цьому аспекті залишається нерозв’язаним завдання комплексної оптимізації, яка б враховувала як продуктивність, так і надійність коду.

Мета і завдання дослідження. Оптимізувати параметри програмного коду за рахунок використання генетичного алгоритму. Для досягнення цієї мети були поставлені такі завдання:

- Дослідити теоретичні основи генетичних алгоритмів та їх застосування в оптимізації коду.
- Вибрати та обґрунтувати датасет для тестування розробленого методу.
- Спроектувати архітектуру методу оптимізації, що базується на генетичному алгоритмі.
- Реалізувати розроблений метод та провести тестування на обраному датасеті.
- Провести порівняння продуктивності оптимізованого коду з традиційними методами.

Об'єкт дослідження – процес створення програмного коду на етапі рефакторингу.

Предмет дослідження – застосування генетичних алгоритмів для оптимізації програмного коду.

Методи дослідження. Методи оптимізації параметрів програмного коду з застосуванням генетичних алгоритмів.

Наукова новизна та практична значущість отриманих результатів полягає в розробці універсального підходу до оптимізації параметрів коду, який дозволяє автоматизувати процес підбору продуктивних рішень за допомогою генетичних алгоритмів. Наукова новизна полягає у вдосконаленні методики оптимізації та її адаптації до змінних умов. Практична значущість роботи полягає в можливості її використання для оптимізації параметрів ПЗ в умовах обмежених обчислювальних ресурсів на підприємствах та в організаціях.

Апробація результатів та публікації. Основні результати роботи були представлені на групових зборах, засіданнях кафедри, а також наукових конференціях і семінарах, зокрема на конференціях з інноваційних інформаційних

технологій, що підтверджує актуальність та практичну значущість виконаного дослідження.

Теоретична, методична та практична значущість отриманих результатів. Розроблений метод має як прикладне, так і теоретико-методичне значення. Прикладна значущість полягає в практичному використанні отриманих результатів для оптимізації параметрів ПЗ, а теоретичні аспекти можуть бути використані для подальших досліджень у галузі штучного інтелекту та еволюційних алгоритмів.

Структура роботи. Робота складається з вступу, чотирьох розділів, висновків, списку використаних джерел та додатків

1 ОСОБЛИВОСТІ ВИКОРИСТАННЯ ГЕНЕТИЧНОГО АЛГОРИТМУ

1.1 Теоретичні основи генетичних алгоритмів

Генетичні алгоритми (ГА) є потужним інструментом оптимізації, який використовує принципи природного добору для пошуку найкращого розв’язку задачі [1]. Вони імітують процес еволюції, розвиваючи покоління розв'язків, які поступово покращуються через застосування біологічно натхненних операцій, таких як селекція, кросовер та мутація [2]. Генетичні алгоритми зарекомендували себе як ефективний засіб для розв’язання багатьох складних завдань оптимізації, зокрема в задачах з великим простором розв’язків або в задачах, де традиційні методи оптимізації демонструють обмежену ефективність [3].

Основні принципи роботи генетичних алгоритмів подані в таблиці 1.1 [4].

Таблиця 1.1

Основні принципи роботи генетичних алгоритмів

№	Принципи	Пояснення
1	2	3
1	Імітація біологічної еволюції	ГА базується на концепції еволюції Чарльза Дарвіна, яка передбачає природний добір найбільш пристосованих особин для виживання. У ГА цей принцип застосовується для пошуку найкращих рішень шляхом «еволюції» набору можливих рішень або «популяції». Популяція складається з численних «особин», кожна з яких представляє можливий розв’язок задачі. Метою ГА є поступове вдосконалення популяції шляхом повторного застосування алгоритмічних кроків
2	Кодування розв’язків у вигляді генетичних представлень	У генетичних алгоритмах розв’язки задачі кодуються зазвичай у вигляді рядків з чисел, бітів або символів, які називають «хромосомами». Кожен елемент хромосоми, що відповідає певній характеристиці розв’язку, називається «геном». Вибір схеми кодування є ключовим для успішного функціонування ГА, оскільки вона визначає можливість і точність оцінки кожного рішення

Продовження таблиці 1.1

Основні принципи роботи генетичних алгоритмів

1	2	3
3	Оцінка пристосованості рішень	У ГА кожен розв’язок оцінюється за допомогою спеціальної функції — "функції пристосованості", яка визначає, наскільки добре цей розв'язок відповідає критеріям задачі. Функція пристосованості відіграє вирішальну роль у відборі рішень для подальшої оптимізації. Ті рішення, які мають вищі значення функції пристосованості, отримують більшу ймовірність бути обраними для наступного покоління, що забезпечує поступове покращення популяції

Основні структурні елементи генетичного алгоритму показані на рисунку 1.1 та описані в таблиці 1.2 [5].

Рис. 1.1 Основні структурні елементи генетичного алгоритму

Таблиця 1.2

Основні структурні елементи

№	Елементи	Пояснення
1	2	3
1	Ініціалізація популяції	Процес роботи ГА починається з генерації початкової популяції випадкових рішень. Це дозволяє алгоритму здійснювати пошук в широкому просторі можливих рішень. Початкову популяцію зазвичай формують таким чином, щоб вона охоплювала різноманітні можливості, що підвищує ймовірність успішної оптимізації

Продовження таблиці 1.2

Основні структурні елементи

1	2	3
2	Відбір (селекція)	Після оцінки функції пристосованості кожного рішення здійснюється відбір тих рішень, які будуть брати участь у подальшому процесі. Найчастіше застосовують методи, такі як "турнірний відбір" або "відбір за рулеткою", які сприяють тому, що рішення з вищим рівнем пристосованості мають більшу ймовірність бути обраними для створення наступного покоління
3	Кросовер (рекомбінація)	Кросовер полягає у комбінуванні частин двох або більше рішень для створення нових. Це імітує природний процес обміну генетичною інформацією і сприяє появі нових комбінацій генів, які можуть мати кращі характеристики. Кросовер є ключовим для генетичних алгоритмів, оскільки дозволяє ефективно досліджувати нові області простору рішень
4	Мутація	Мутація вносить невеликі випадкові зміни у вибрані хромосоми. Це зменшує ризик передчасного збіжності алгоритму і дозволяє уникати потрапляння в локальні мінімуми, відкриваючи нові можливості для покращення популяції
5	Заміщення	Після застосування кросовера і мутації створюється нове покоління рішень, яке замінює попереднє. Зазвичай, нове покоління формується з певною часткою особин, обраних із старого покоління, що зберігає "пам’ять" про кращі попередні рішення
6	Завершення алгоритму	Процес повторюється доти, доки не буде досягнуто заданого критерію зупинки — зазвичай, коли кількість поколінь досягне заданого значення або ж коли значення функції пристосованості перестають суттєво покращуватись

Генетичні алгоритми є надзвичайно гнучким інструментом, здатним розв’язувати складні задачі оптимізації завдяки імітації біологічних процесів еволюції [6]. Теоретичні основи ГА включають ключові поняття, такі як кодування рішень, оцінка пристосованості та базові операції (відбір, кросовер, мутація), які сприяють ефективному пошуку оптимального розв'язку. Завдяки здатності ГА адаптуватись та генерувати нові можливі варіанти, вони демонструють високу ефективність у задачах, де інші методи можуть виявитися неефективними. Такий підхід дозволяє підвищити продуктивність та точність розв’язків і є перспективним напрямком для вдосконалення програмного забезпечення та оптимізації коду.

1.2 Використання генетичних алгоритмів для оптимізації програмного коду

Генетичні алгоритми (ГА) є потужним інструментом для вирішення задач оптимізації, і їхній потенціал активно використовується в різних галузях, зокрема для оптимізації програмного коду. Оптимізація коду має важливе значення в умовах сучасного програмного забезпечення, адже висока продуктивність і ефективність ресурсів є ключовими критеріями якості. Використання ГА дозволяє автоматизувати процес пошуку оптимальних рішень у складних багатоваріантних середовищах [7]. Цей підхід імітує еволюційні механізми природного відбору, сприяючи генерації та відбору найбільш ефективних варіантів коду.

Принцип дії генетичних алгоритмів у задачах оптимізації коду.

Генетичний алгоритм є метаевристичним методом, що використовує механізми, подібні до біологічної еволюції, включаючи операції селекції, схрещування та мутації [8]. У контексті оптимізації коду, він працює за алгоритмом, етапи якого представлені на рисунку 1.2 та описані в таблиці 1.3 [9] [10].

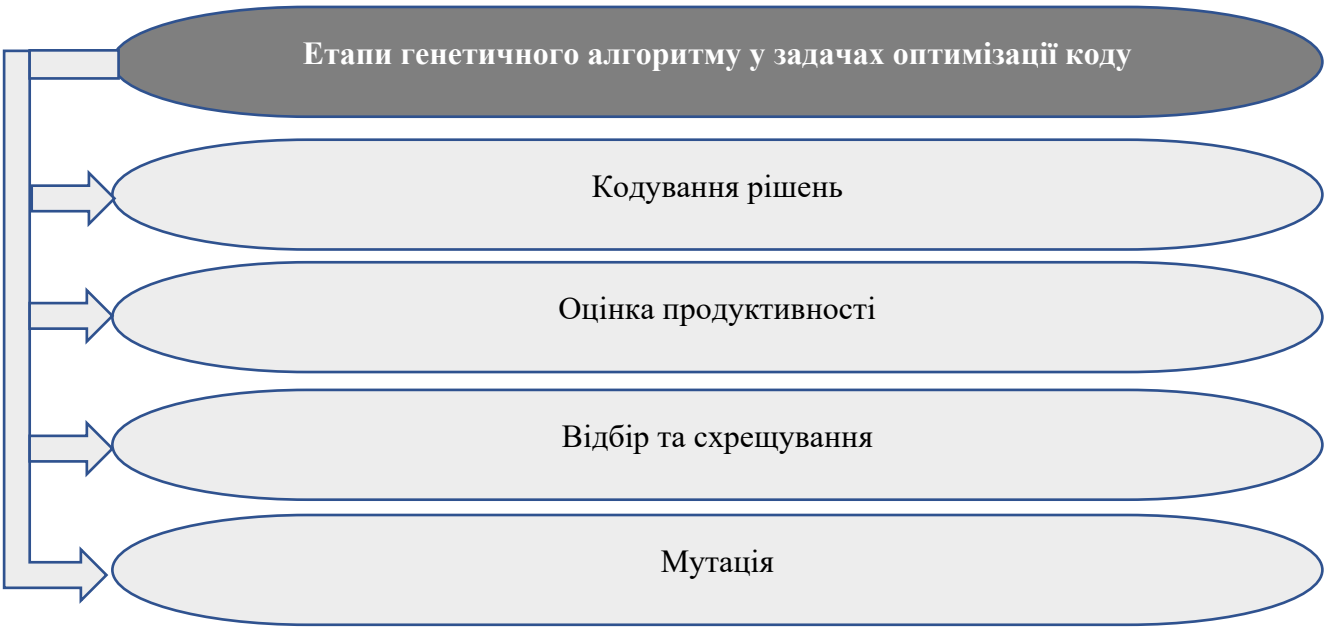


Рис. 1.2 Етапи генетичного алгоритму у задачах оптимізації коду

Таблиця 1.3

Етапи генетичного алгоритму у задачах оптимізації коду

№	Етапи	Пояснення
1	Кодування рішень	Кожен варіант програми (або її частини) представляється у вигляді особини — набору параметрів, які кодуються в структурованій формі (наприклад, бінарним або рядковим кодуванням)
2	Оцінка продуктивності	Кожній особині присвоюється функція пристосованості, що оцінює її відповідність заданим критеріям (продуктивність, енергоефективність, швидкість виконання).
3	Відбір та схрещування	Обираються найбільш придатні особини, які потім схрещуються для генерації нових рішень. Механізм схрещування дозволяє поєднувати найбільш успішні частини коду різних особин, створюючи потенційно більш ефективні варіанти
4	Мутація	Для підтримання генетичного різноманіття до отриманих рішень застосовуються невеликі випадкові зміни, які дозволяють уникати локальних екстремумів і сприяють глобальній оптимізації

Етапи оптимізації коду за допомогою генетичних алгоритмів

Оптимізація програмного коду за допомогою ГА включає етапи представлені на рисунку 1.3 [11].

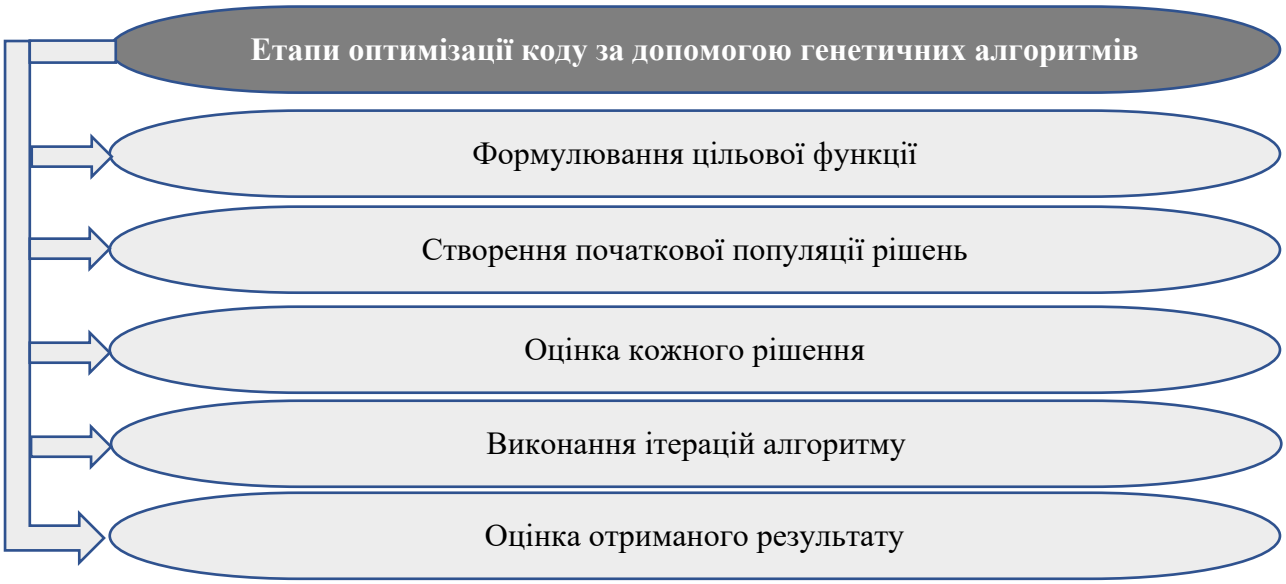


Рис. 1.3 – Етапи оптимізації коду за допомогою генетичних алгоритмів

На початковому етапі визначаються ключові метрики, які потрібно оптимізувати (наприклад, час виконання функції, обсяг пам’яті, використаний на обчислення).

На основі оригінального коду генерується початкова множина варіантів, кожен з яких представляє можливу модифікацію програми.

Виконується тестування кожного варіанту коду, щоб визначити рівень відповідності встановленим критеріям оптимізації.

Процес відбору, схрещування та мутації повторюється доти, поки не буде досягнуто необхідного рівня оптимізації або не буде вичерпано ресурсів (часу або обчислювальної потужності).

Завершальним кроком є тестування та аналіз знайденого рішення на відповідність поставленим вимогам. Якщо рішення не відповідає очікуванням, можуть бути внесені коригування в налаштування алгоритму або цільову функцію.

Застосування генетичних алгоритмів до різних аспектів оптимізації програмного коду

Генетичні алгоритми дозволяють оптимізувати код за критеріями, які подані в табл. 1.4 [12].

Таблиця 1.4

Критерії оптимізації коду за допомогою генетичних алгоритмів

№	Етапи	Пояснення
1	Оптимізація швидкодії	Завдяки автоматичному пошуку більш продуктивних варіантів можна зменшити час виконання окремих функцій чи програмного модуля
2	Зниження використання ресурсів	Зокрема, оптимізація обчислювальних ресурсів і пам’яті досягається шляхом виявлення та видалення зайвих фрагментів або переписування певних структур коду
3	Поліпшення структури коду	ГА також можна використовувати для автоматичного пошуку більш раціональної організації коду, включно з рефакторингом або модульною оптимізацією, що сприяє поліпшенню читабельності та підтримки коду
4	Енергозбереження	В умовах розробки програм для мобільних і вбудованих пристроїв генетичні алгоритми допомагають скоротити енергоспоживання за рахунок зменшення навантаження на центральний процесор

Переваги та обмеження використання генетичних алгоритмів в оптимізації коду (рисунок 1.4) [13]

Рис. 1.4. Переваги та обмеження використання генетичних алгоритмів в оптимізації коду

Генетичні алгоритми мають низку переваг у задачах оптимізації програмного коду:

Гнучкість – ГА легко адаптуються до різних типів завдань і цільових функцій, що дозволяє налаштовувати їх для різних критеріїв оптимізації.

Пошук глобальних рішень – Можливість уникати локальних мінімумів і досягати глобальних оптимумів особливо важлива для складних задач оптимізації.

Автоматизація процесу – Використання ГА значно скорочує обсяг ручної праці та дозволяє досягти результатів навіть у випадках, коли точні аналітичні методи є недоступними.

Разом з тим, існують певні обмеження:

Високі вимоги до обчислювальних ресурсів – Запуск ГА може потребувати значної кількості часу та ресурсів, що є критичним чинником при обробці великих обсягів коду.

Чутливість до налаштувань – Вибір параметрів, таких як розмір популяції, коефіцієнти мутації та селекції, суттєво впливає на якість і швидкість знаходження оптимального рішення.

Застосування генетичних алгоритмів для оптимізації програмного коду є перспективним напрямом, що дозволяє автоматизувати пошук ефективних і продуктивних рішень у процесі розробки програмного забезпечення. ГА сприяють підвищенню продуктивності коду, поліпшенню його структури та зниженню ресурсозатратності, що є критичним для сучасних додатків з високими вимогами до ефективності. Хоча використання цього методу має певні обмеження, вдосконалення алгоритмів та збільшення обчислювальних можливостей роблять його все більш доступним і застосовним у реальних умовах програмування.

1.3 Огляд попередніх досліджень та рішень

Генетичні алгоритми (ГА) вже тривалий час привертають увагу дослідників як один із ефективних інструментів для розв’язання складних задач оптимізації. Особливо значущим є застосування генетичних алгоритмів у галузі оптимізації програмного коду, де вони дають змогу автоматизувати процес пошуку найбільш ефективних рішень. Досвід використання ГА в оптимізації програмного коду є різноманітним і включає численні підходи та методики, що були розроблені для вирішення різних аспектів проблеми.

Основні дослідження подані в таблиці 1.5.

Таблиця 1.5

Основні дослідження генетичних алгоритмів

№	Дослідження	Пояснення
1	**2**	**3**
1	Дослідження Джеймса Коулза (1994)	Коулз запропонував перший генетичний алгоритм для оптимізації програмного коду, що стало основою для подальших розробок [14]

Продовження таблиці 1.5

Основні дослідження генетичних алгоритмів

1	2	3
2	Дослідження Смітсона та Лі (2001)	Вони внесли значний вклад у розвиток методів генетичних алгоритмів, зокрема у випадку оптимізації паралельних систем [15]
3	Дослідження Ченга та Лі (2010)	Ченг та Лі розробили новий підхід до оптимізації коду за допомогою генетичних алгоритмів, що значно підвищило ефективність процесу [16]

Огляд підходів та рішень

Оптимізація продуктивності коду. Генетичні алгоритми широко застосовуються для підвищення швидкодії програмного забезпечення. Зокрема, дослідження в цьому напрямі зосереджені на автоматичному виборі оптимальних параметрів для коду або алгоритму, що дозволяє зменшити час виконання програми [17]. Підхід включає в себе використання ГА для генерування і тестування різних варіантів конфігурацій коду, визначення менш ресурсоємних структур даних та вибору алгоритмів з найменшими витратами обчислювальної потужності [18]. Приклади таких досліджень демонструють, що ГА здатні значно скоротити час виконання коду без значної зміни його структури, забезпечуючи при цьому стабільну роботу програмного продукту.

Мінімізація обсягу коду та використання пам’яті. Ще однією важливою областю дослідження є оптимізація коду щодо його обсягу [19]. Надмірний обсяг коду може призводити до підвищеного використання пам’яті та зниження загальної продуктивності системи. В попередніх дослідженнях ГА використовувалися для автоматичного видалення надлишкових елементів коду або заміни складних структур на більш компактні. Деякі рішення дозволяють проводити рефакторинг коду, орієнтуючись на мінімізацію зайвих операцій і обсягів пам’яті, що виділяється, що є особливо актуальним для систем з обмеженими ресурсами, наприклад, вбудованих систем.

Оптимізація під конкретну архітектуру або платформу. Генетичні алгоритми також успішно застосовуються для адаптації коду під специфічні апаратні платформи. Дослідження в цьому напрямі показали, що за допомогою ГА можна налаштовувати параметри коду або алгоритмів відповідно до особливостей апаратного забезпечення, таких як архітектура процесора чи обсяг кеш-пам’яті

[20]. Наприклад, оптимізація для процесорів ARM та x86 може мати суттєві відмінності, і ГА дозволяють автоматично знаходити такі комбінації параметрів, які найбільш ефективно працюють на конкретній архітектурі.

Автоматичний пошук оптимальних алгоритмів. В окремих дослідженнях генетичні алгоритми використовуються для пошуку найкращих алгоритмів для виконання певного завдання в програмному коді [21]. У цьому випадку генетичний алгоритм може випробовувати різні варіанти алгоритмів, аналізуючи їх продуктивність і обираючи ті, що забезпечують найкращі результати. Наприклад, для обробки великих даних ГА можуть допомогти автоматично вибрати найшвидший алгоритм сортування або обробки інформації, що дозволяє зменшити час виконання програми.

Оптимізація за критеріями енергозбереження. Дослідження з оптимізації коду за критеріями енергозбереження є особливо актуальними для мобільних пристроїв та портативних систем [22]. Генетичні алгоритми використовуються для адаптації програмних компонентів, орієнтованих на зниження енергоспоживання без шкоди продуктивності. Попередні дослідження показують, що ГА можуть бути ефективними для вибору енергоефективних варіантів виконання коду, особливо для задач, де важливо підтримувати тривалий час автономної роботи пристроїв.

Системи автоматизованої оптимізації та застосування в інструментах розробки. Сучасні дослідження в галузі автоматизованої оптимізації коду на основі генетичних алгоритмів все частіше орієнтовані на створення інструментів, які могли б інтегруватися в середовища розробки [23]. Такі системи дозволяють автоматично застосовувати ГА під час компіляції або при профілюванні коду, забезпечуючи безперервну оптимізацію продукту.

Огляд попередніх досліджень та рішень показує, що генетичні алгоритми є універсальним та потужним інструментом, здатним вирішувати різноманітні задачі оптимізації програмного коду. Їх застосування дозволяє зменшити час виконання програм, оптимізувати використання ресурсів, адаптувати код до конкретної архітектури або платформи, а також підвищувати енергоефективність. Попередній досвід показує значний потенціал генетичних алгоритмів для автоматизації процесів оптимізації та впровадження інструментів, які інтегруються у робочий процес розробника. Перспективи розвитку цих технологій полягають у подальшому вдосконаленні методів оптимізації, а також у розробці нових інструментів, що спрощують застосування ГА в реальних умовах.

2 ПРОЕКТУВАННЯ МЕТОДУ ОПТИМІЗАЦІЇ

2.1 Постановка завдання оптимізації коду

Оптимізація коду є одним із ключових етапів у розробці ефективного програмного забезпечення, особливо в умовах, де продуктивність і адаптивність алгоритмів мають вирішальне значення для кінцевих користувачів. Оптимізація параметрів фокусується на налаштуванні вхідних параметрів програми, не змінюючи основного коду, щоб досягти кращих результатів у роботі програми. Сучасні технології надають розробникам широкий спектр інструментів для покращення продуктивності їхніх програм, зокрема завдяки методам автоматизованої оптимізації. Серед таких методів найбільш цікавими і перспективними є генетичні алгоритми, які наслідують принципи природної еволюції для пошуку оптимальних параметрів.

В умовах постійно зростаючої складності сучасних програмних систем завдання оптимізації параметрів стає не просто засобом підвищення швидкодії, а критично важливим аспектом. Зокрема, для застосунків, що обробляють великі обсяги даних або потребують складних обчислень (наприклад, у сфері штучного інтелекту або обробки зображень), ефективна оптимізація здатна значно скоротити час обробки, зменшити споживання ресурсів та забезпечити стабільність виконання. З огляду на це, основною метою є створення методології, яка дозволить автоматично знаходити оптимальні значення для параметрів алгоритмів, що використовуються в коді, і в такий спосіб покращувати продуктивність програмного забезпечення.

Метод оптимізації параметрів: використання генетичного алгоритму

Метод оптимізації параметрів базується на генетичному алгоритмі, який налаштовується для пошуку оптимальних параметрів, що покращують ефективність виконання алгоритмів. Основна ідея генетичного алгоритму полягає в ітеративному відборі, комбінації та мутації наборів параметрів, що імітує процес

природного добору. У цій методології визначено ключові завдання оптимізації, які показані на рисунку 2.1.

Рис. 2.1 Ключові параметри завдання оптимізації

Визначення параметрів для оптимізації. Основний крок у проектуванні методу оптимізації полягає у визначенні параметрів, що підлягають налаштуванню. Це можуть бути значення розмірів популяцій, кількість ітерацій, значення ймовірності мутації тощо. Визначено такі параметри: кількість поколінь, розмір популяції, коефіцієнт мутації та розмір турніру. Завдання полягає у виборі таких значень для цих параметрів, що забезпечать найкращий баланс між швидкістю виконання та точністю роботи алгоритму.

Налаштування початкових значень. Для запуску оптимізаційного процесу важливо задати початкові значення для параметрів, що будуть налаштовуватись. Генетичний алгоритм починає роботу зі створення початкової популяції, кожен член якої є набором параметрів. Потім обчислюється продуктивність (фітнес-функція) для кожного набору параметрів. Фітнес-функція — це критерій, за яким оцінюється, наскільки добре конкретний набір параметрів підходить для задачі.

Використання фітнес-функції. Фітнес-функція оцінює продуктивність для кожного набору параметрів. Ця функція вимірює показники, такі як швидкодія або обчислювальна ефективність. Мета фітнес-функції — ідентифікація наборів

параметрів, які забезпечують найкращу продуктивність для конкретної задачі, а також подальший добір найбільш перспективних параметрів для подальших ітерацій алгоритму.

Виконання турнірного відбору. Турнірний відбір — це метод відбору, що використовується для вибору батьків, з яких у наступній генерації будуть отримані нові набори параметрів. Для забезпечення розмаїття в параметрах обрані набори параметрів піддаються мутації, що дозволяє уникнути стагнації в оптимізаційному процесі та знайти нові оптимальні значення. Це гарантує, що з кожною ітерацією алгоритму відбувається адаптація параметрів.

Мутація та кросинговер. На кожному кроці алгоритму з’являються нові комбінації параметрів, які піддаються мутації — випадковій зміні параметрів з певною ймовірністю. Кросинговер, тобто комбінація параметрів із двох обраних батьківських наборів, забезпечує появу нового покоління з оптимізованими значеннями параметрів.

Оцінка стабільності та адаптивності параметрів. Важливою частиною методу є також можливість забезпечення стабільної роботи програми, що досягається через перевірку значень на адаптивність до різних вхідних даних. З кожною ітерацією генетичний алгоритм зменшує похибку у параметрах, наближаючись до оптимальних значень, що забезпечують найкращий баланс між швидкістю обробки та точністю.

Результати виконання оптимізації та збереження параметрів

Після завершення роботи алгоритму параметри зберігаються у зовнішньому файлі settings.json, що дає змогу програмі автоматично завантажувати оптимальні значення при наступних запусків. Це гарантує, що програма постійно працюватиме з найкращими налаштуваннями, не потребуючи додаткових налаштувань вручну. Процес збереження дозволяє зберегти налаштування щодо мови, розміру популяції, кількості поколінь, коефіцієнту мутації та інших параметрів, що забезпечують стабільну роботу.

Оптимізація параметрів на основі генетичного алгоритму дозволяє суттєво підвищити ефективність виконання алгоритмів за рахунок автоматичного

налаштування важливих параметрів, таких як кількість поколінь, розмір популяції, коефіцієнт мутації тощо. Метод забезпечує адаптивність і стабільність програмного забезпечення, надаючи можливість програмі автоматично підлаштовуватися під різні умови виконання.

2.2 Вибір і обґрунтування датасету

У рамках розробки методу оптимізації параметрів на основі генетичного алгоритму критично важливо забезпечити наявність належного набору даних (датасету), на якому буде проводитися навчання і тестування алгоритму. Датасет є базовим елементом, що визначає успіх чи невдачу будь-якого оптимізаційного процесу, оскільки від його якості, обсягу і відповідності поставленим цілям залежить точність, ефективність та релевантність отриманих результатів.

Процес оптимізації параметрів програмного коду є складним і залежить від багатьох факторів, зокрема від типу алгоритму, умов його виконання та необхідної точності. Для розробки методу оптимізації на основі генетичного алгоритму важливо забезпечити відповідний датасет, який буде використаний для тестування і перевірки ефективності цього методу. Вибір і обґрунтування датасету є критичним етапом, оскільки від цього залежить точність і надійність результатів оптимізації.

Вибір датасету

Для налаштування і тестування запропонованого методу оптимізації параметрів програмного коду важливо мати багатий і різноманітний набір даних, що відображає різні типи обчислювальних задач. Вибір датасету, який містить реалістичні приклади, дозволить найбільш ефективно перевірити можливості методу, а також наочно продемонструвати його практичну значущість. Доцільно обрати датасети, що вирішують проблеми з різними типами алгоритмів і задач, таких як сортування великих масивів, обробка зображень, аналіз фінансових даних або алгоритми компресії (таблиця 2.1).

Таблиця 2.1

Алгоритми та задачі датасетів

№	Алгоритми та задачі	Пояснення
1	Сортування великих масивів	Оскільки процес сортування є одним із основних в багатьох програмних додатках, оптимізація параметрів алгоритму сортування (наприклад, порогу переключення між швидким сортуванням і сортуванням вставками) є важливою задачею. Такий датасет дозволяє перевірити, наскільки ефективно генетичний алгоритм може налаштувати ключові параметри для досягнення найкращої швидкості сортування
2	Обробка зображень	Для задач обробки зображень важливими є параметри, що відповідають за фільтрацію або стискання зображень, зокрема розміри ядра фільтра або коефіцієнт розмиття. Вибір датасету, що містить різноманітні зображення, дозволяє оптимізувати ці параметри для досягнення найкращого балансу між якістю зображення та швидкістю обробки
3	Аналіз фінансових даних	Для алгоритмів, що аналізують великі обсяги фінансових даних, важливими є параметри вибірки, порогових значень для оцінки значущості результатів або налаштування для побудови фінансових моделей. Тестування методу оптимізації параметрів на таких наборах даних дозволяє перевірити ефективність у сфері обробки великих даних
4	Алгоритми компресії даних	Оскільки вибір оптимальних параметрів може значно впливати на ефективність алгоритмів компресії, датасет, що включає текстові файли, зображення та інші типи даних, дозволяє тестувати налаштування для досягнення найкращої ефективності компресії, залежно від виду даних

Обґрунтування вибору

Для тестування запропонованого методу оптимізації обрано різноманітні типи задач, оскільки вони відображають широкий спектр реальних сценаріїв, з якими стикаються розробники. Завдяки використанню таких різноманітних датасетів можна забезпечити більш точне налаштування параметрів і оптимізацію у багатьох сферах програмної інженерії. Використання реалістичних задач дозволяє також оцінити ефективність методу в умовах, наближених до реальних. Оскільки метод генетичної оптимізації потребує перевірки на великих обсягах

даних, ці набори допомагають забезпечити перевірку на різноманітних типах вхідних даних і умовах виконання.

Вибір і обґрунтування датасету важливі не лише для перевірки методів оптимізації, але й для забезпечення практичної значущості програмного забезпечення. Методи оптимізації повинні бути здатні покращити ефективність роботи програмних систем, не обмежуючи їх функціональність і не потребуючи значних ресурсів. Таким чином, важливою метою є вибір таких наборів даних, що дозволяють провести всебічну оцінку можливостей запропонованого методу.

Вибір і обґрунтування датасету є важливим етапом розробки методів оптимізації параметрів. Вибрані датасети для задач сортування, обробки зображень, аналізу фінансових даних та компресії є представниками широкого спектру реальних проблем, що дозволяє перевірити універсальність і ефективність методу. Завдяки такому підходу можна досягти оптимізації параметрів програмного коду, яка не лише покращить продуктивність, але й зробить алгоритми більш адаптивними до різних умов і сценаріїв.

2.3 Архітектура методу оптимізації

Метод оптимізації, що описаний в попередніх частинах коду, є складним програмним рішенням, яке дозволяє ефективно налаштовувати параметри алгоритмів для підвищення їх продуктивності. Він поєднує в собі використання генетичних алгоритмів та інтерфейсу для взаємодії з користувачем, що дозволяє здійснювати налаштування та моніторинг оптимізації в реальному часі. Основною метою цього методу є оптимізація параметрів програмного коду без необхідності переписування його логіки. Завдяки використанню еволюційних алгоритмів, метод дозволяє знайти найкращі значення для параметрів, що відповідають за ефективність роботи програмного коду.

Основні компоненти архітектури методу оптимізації

Архітектура методу оптимізації є багаторівневою і включає в себе кілька важливих компонентів, які працюють разом для досягнення оптимальних

результатів. Кожен з компонентів має свою специфічну задачу, але в цілому вони утворюють єдину, злагоджену систему. Основні елементи архітектури подані на рисунку 2.2.

Рис. 2.2 Основні елементи архітектури методу оптимізації

Генетичний алгоритм – основний механізм, що використовується для оптимізації параметрів. Генетичний алгоритм працює за принципом природного відбору, де популяція потенційних розв'язків еволюціонує шляхом схрещування та мутацій. Він дозволяє знаходити оптимальні значення параметрів для різних функцій чи алгоритмів. В коді генетичний алгоритм реалізовано через використання параметрів, таких як розмір популяції, кількість поколінь, коефіцієнт мутації та розмір турніру, що визначають, як швидко і ефективно алгоритм досягає оптимуму.

Генетичний алгоритм представлений на рисунку 2.3.

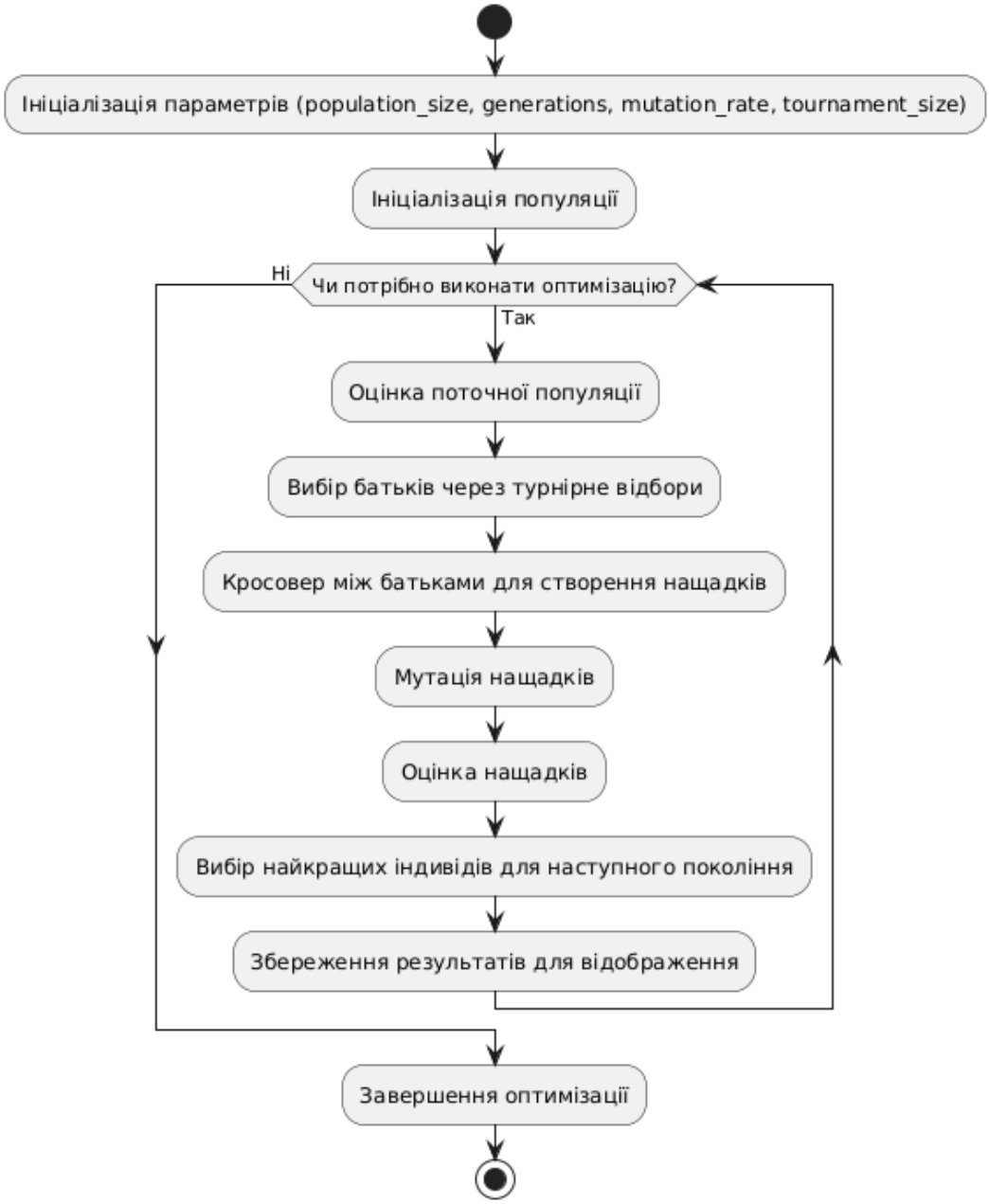


Рис. 2.3 Блок-схема генетичного алгоритму

Користувацький інтерфейс – для зручності взаємодії з програмою розроблено інтерфейс на основі Tkinter, який дозволяє користувачу налаштовувати параметри та отримувати зворотний зв'язок. Інтерфейс включає в себе текстове поле для виведення логів і статусів роботи програми, а також поля для введення значень параметрів (рисунок 2.4). Це забезпечує інтуїтивно зрозумілу роботу програми навіть для користувачів з обмеженим досвідом.

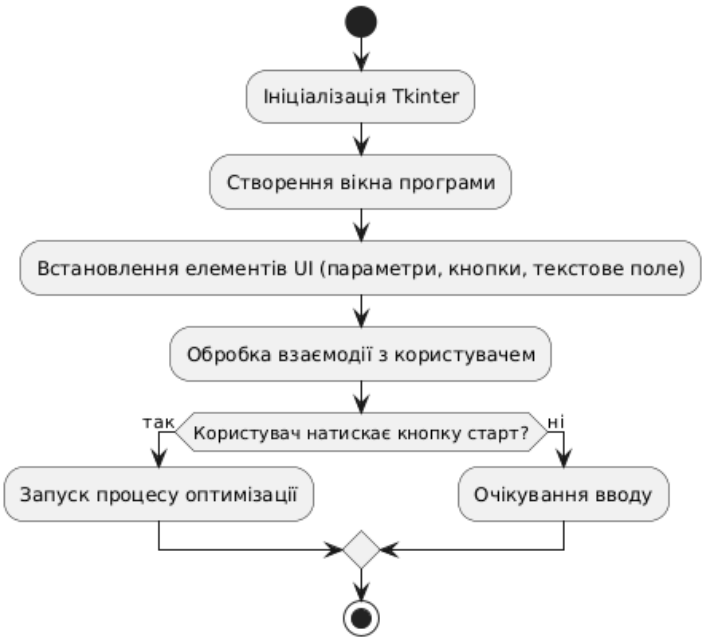


Рис 2.4 Блок-схема користувацького інтерфейсу

Модуль збереження та завантаження налаштувань – для забезпечення безперервності роботи програми в процесі оптимізації передбачено механізм збереження налаштувань у файл (у форматі JSON) (рисунок 2.5). Це дозволяє зберігати поточний стан програми і відновлювати його при наступному запуску, що є важливим для тривалих процесів оптимізації. Завдяки цьому, користувач може без перешкод продовжити роботу в будь-який момент, не втрачаючи важливих даних.

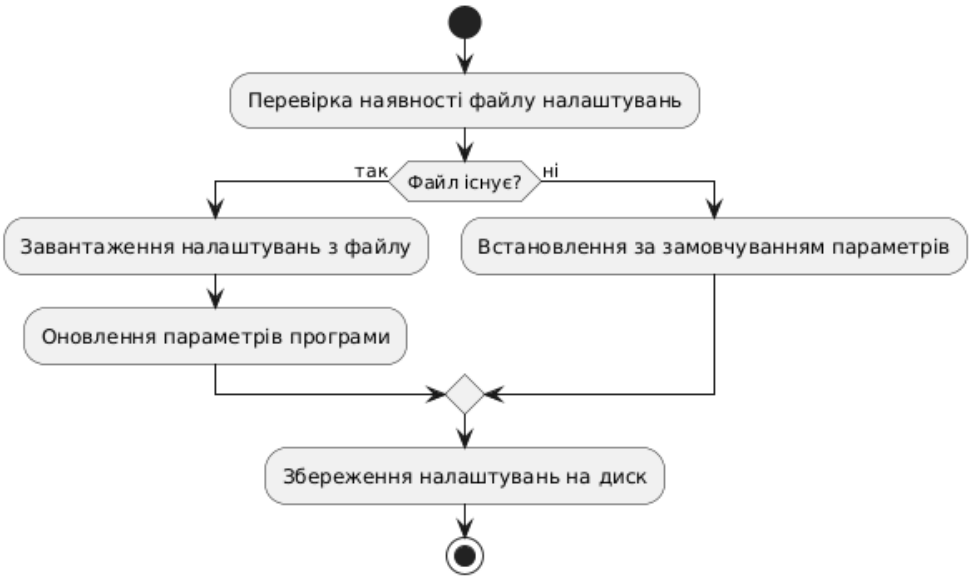


Рис. 2.5 Блок-схема модулю збереження та завантаження налаштувань

Модуль для ведення логів – для ефективного моніторингу роботи програми та виявлення помилок реалізовано систему логування (рисунок 2.6). Логи фіксують всі важливі події, що відбуваються під час роботи програми, такі як початок та завершення оптимізації, зміни параметрів, помилки при виконанні тощо. Це дозволяє користувачу та розробникам отримувати детальну інформацію про виконання процесу оптимізації і швидко реагувати на можливі проблеми.

Рис. 2.6 Блок-схема модулю для введення логів

Механізм очищення ресурсів – після завершення процесу оптимізації або при закритті програми передбачено очищення ресурсів, що включає в себе зупинку всіх активних потоків, звільнення пам'яті та збереження налаштувань (рисунок 2.7). Це дозволяє мінімізувати ймовірність виникнення помилок при наступному запуску програми та забезпечує ефективне використання системних ресурсів.

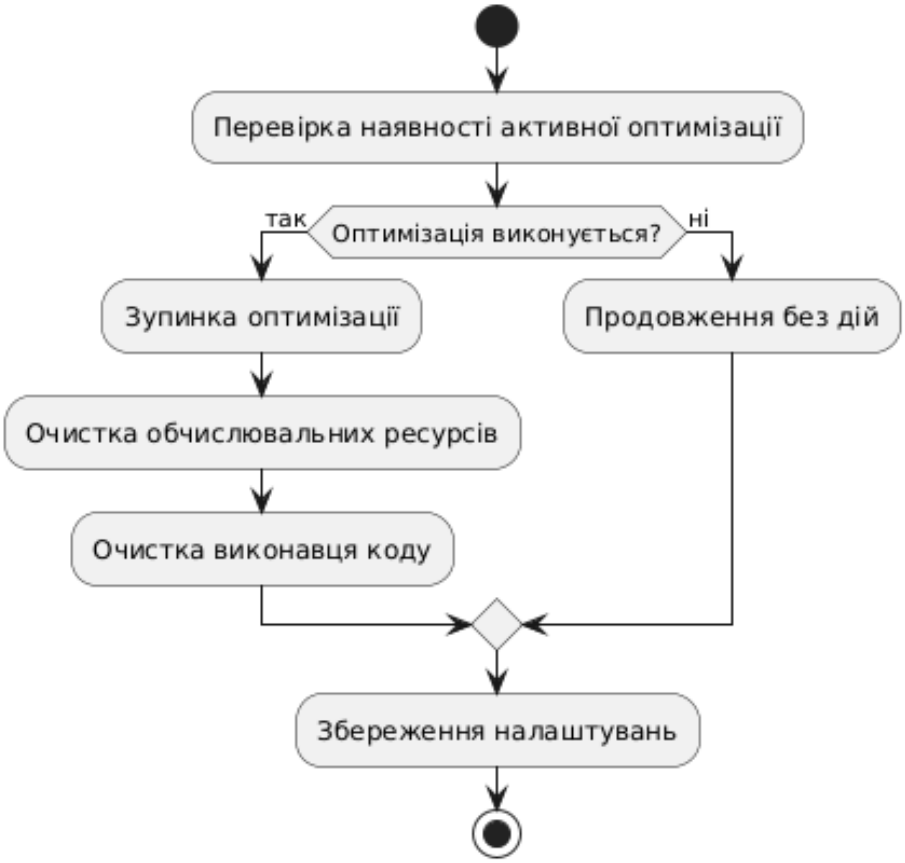


Рис. 2.7 Блок-схема механізму очищення ресурсів

Принципи роботи методу оптимізації

Основна ідея методу полягає в тому, щоб автоматично знаходити найбільш ефективні параметри для різних типів алгоритмів, забезпечуючи при цьому баланс між швидкістю виконання та якістю результату. Метод оптимізації працює за етапами представлені на рисунку 2.8 та описаними в таблиці 2.2.

Рис. 2.8 Етапи методу оптимізації

Таблиця 2.2

Етапи методу оптимізації

№	Етапи	Пояснення
1	Ініціалізація параметрів	Спочатку користувач задає початкові значення параметрів, які будуть використовуватися в процесі оптимізації. Це включає такі параметри, як розмір популяції, кількість поколінь, коефіцієнт мутації тощо (рис. 2.9)
2	Оптимізація	Генетичний алгоритм виконує етапи відбору, схрещування та мутації для створення нових поколінь розв'язків, поступово покращуючи їх до досягнення оптимального значення (рис.2.10)
3	Оцінка ефективності	Кожен з потенційних розв'язків оцінюється за допомогою відповідної функції ефективності, яка дозволяє визначити, наскільки добре параметри справляються з поставленою задачею (рис. 2.11)
4	Підбір оптимальних параметрів	На основі результатів оцінки генетичний алгоритм вибирає найбільш ефективні значення параметрів і передає їх користувачу (рис. 2.12)
5	Збереження результатів	Після завершення процесу оптимізації всі налаштування зберігаються в конфігураційному файлі, що дозволяє відновити їх на наступних етапах роботи (рис. 2.13)

Рис. 2.9 Блок-схема ініціалізації параметрів

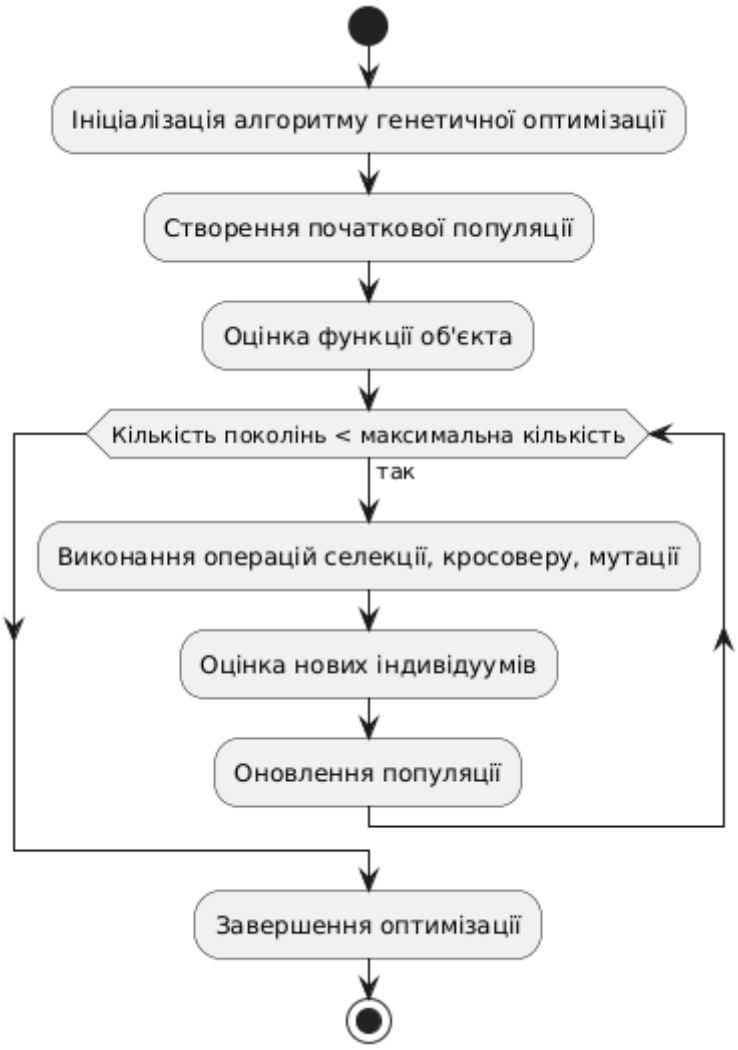


Рис. 2.10 Блок-схема оптимізації

Рис. 2.11 Блок-схема оцінки ефективності

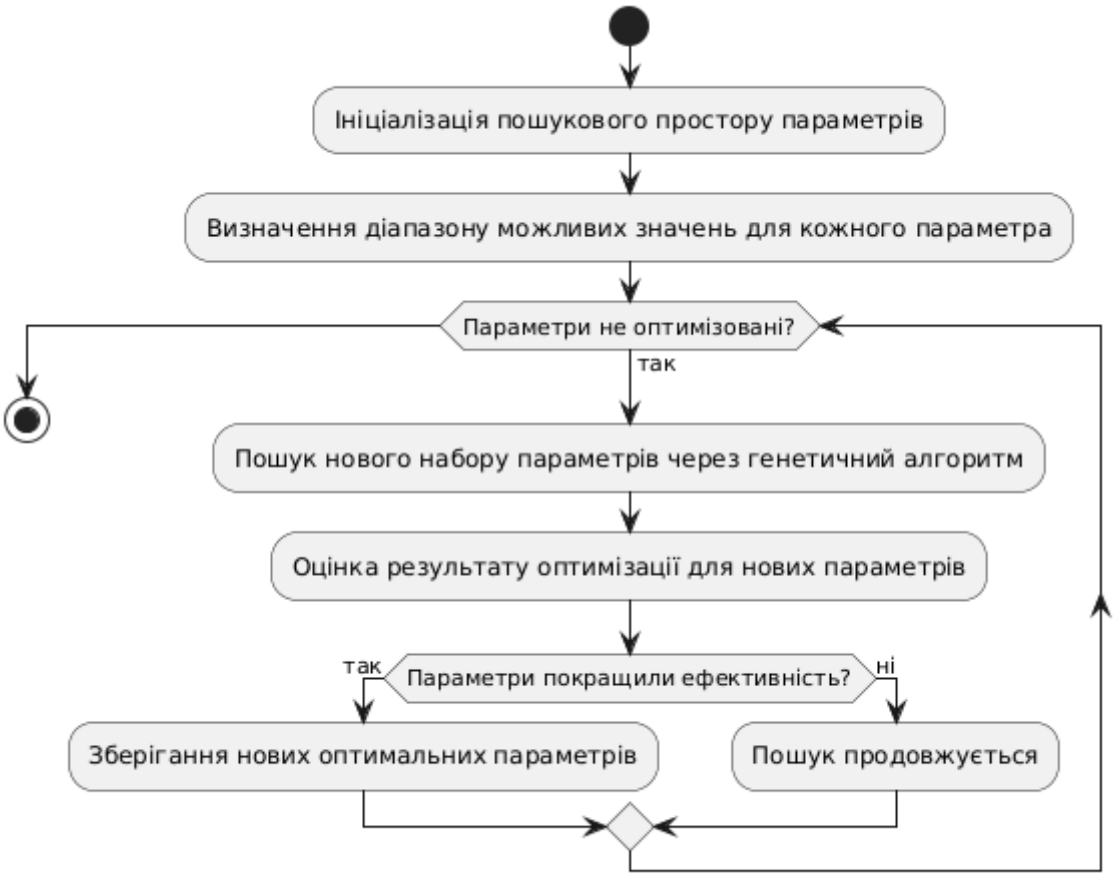


Рис. 2.12 Блок-схема підбору оптимальних параметрів

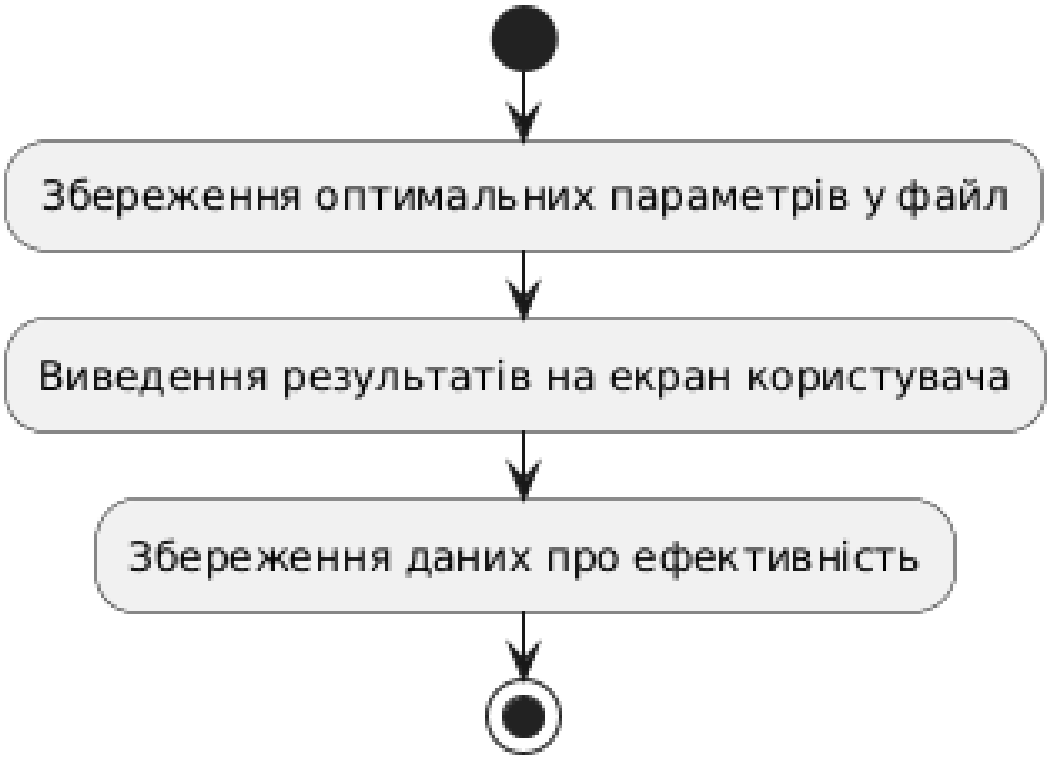


Рис. 2.13 Блок-схема збереження результатів

Архітектура методу оптимізації забезпечує високу гнучкість, ефективність та простоту у використанні. Завдяки використанню генетичних алгоритмів, метод дозволяє автоматично налаштовувати параметри програмного коду для досягнення максимальної ефективності, що є особливо важливим для складних та ресурсоємних обчислень. Користувацький інтерфейс та система логування надають зручні інструменти для моніторингу та контролю процесу оптимізації. Система збереження налаштувань дозволяє зберегти поточний стан програми, що робить роботу з програмою ще більш зручною. Окрім цього, метод оптимізації дозволяє покращити продуктивність без необхідності переписування існуючого коду, що значно економить час і ресурси розробників.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО КОДУ

3.1 Програмні засоби та інструменти

Розробка програмного продукту для оптимізації параметрів на основі генетичного алгоритму передбачає використання низки програмних засобів та інструментів, які забезпечують ефективне створення, тестування та інтеграцію компоненів системи. Вибір технологій та інструментів, застосованих під час розробки цього методу, обґрунтований необхідністю оптимізації параметрів з мінімальними затратами ресурсів та часу, що забезпечить максимальну ефективність програми для широкого кола користувачів. Основні програмні засоби та інструменти, які були використані для реалізації цієї системи, представлені на рисунку 3.1.

Рис. 3.1 Основні програмі засоби та інструменти реалізації системи

Вибір мови програмування

Основною мовою програмування для розробки системи було обрано Python. Це універсальна мова програмування, яка має безліч переваг, таких як простота у використанні, високий рівень абстракції, а також велика кількість бібліотек для обробки даних, розв'язування оптимізаційних задач, роботи з графічними інтерфейсами, що забезпечує зручність розробки. Мова Python також широко використовується в наукових розробках та інженерії, що робить її ідеальним вибором для реалізації генетичних алгоритмів та оптимізаційних задач.

Використання графічного інтерфейсу

Для реалізації графічного інтерфейсу користувача (GUI) був використаний бібліотека Tkinter. Tkinter є стандартним інструментом для створення GUI в Python, надаючи зручні можливості для швидкого створення вікон, кнопок, текстових полів та інших елементів управління. Вибір Tkinter обумовлений його інтеграцією з Python, простотою використання та достатньою функціональністю для створення інтерфейсу, який дозволяє користувачам налаштовувати параметри генетичного алгоритму, відслідковувати процес оптимізації та переглядати результати.

Логування та обробка помилок

Для системи логування використовувався стандартний модуль logging Python, що дозволяє ефективно управляти повідомленнями про події, помилки та іншу важливу інформацію в процесі виконання програми. Модуль logging надає гнучкі механізми для налаштування рівнів важливості повідомлень (наприклад, ERROR, INFO), що дозволяє детально відслідковувати виконання коду, а також здійснювати ефективну діагностику помилок. Впровадження власного обробника логів LogTextHandler, який виводить повідомлення безпосередньо в текстове поле програми, дозволяє користувачам оперативно відслідковувати хід оптимізації та вчасно виявляти можливі проблеми.

Функції збереження та завантаження налаштувань

Для збереження налаштувань користувача та конфігураційних даних використовується формат JSON. Цей формат є простим для обробки та підтримується безпосередньо мовою Python через стандартну бібліотеку json.

Збереження налаштувань у форматі JSON дозволяє забезпечити збереження таких параметрів, як розмір популяції, кількість поколінь, рівень мутації та інші, що дозволяє користувачеві повторно використовувати налаштування при наступному запуску програми. Важливо, що JSON є людським форматом, що дозволяє легко переглядати та редагувати файли вручну, якщо це необхідно.

Потоки та багатозадачність

Для реалізації паралельного виконання задач та забезпечення плавного інтерфейсу користувача було використано механізм багатозадачності в Python. Використання окремого потоку для виконання генетичного алгоритму дозволяє не блокувати інтерфейс програми під час виконання складних обчислень, зокрема під час оптимізації параметрів. Для цього в програмі застосовується стандартна бібліотека threading, що дозволяє створювати потоки, контролювати їх виконання та забезпечувати коректне завершення роботи при закритті програми.

Таким чином, для розробки програми оптимізації параметрів на основі генетичного алгоритму були обрані ефективні та перевірені програмні засоби, зокрема мова Python, бібліотеки Tkinter для створення графічного інтерфейсу, модуль logging для обробки помилок та логування, формат JSON для збереження налаштувань користувача, а також механізми багатозадачності для паралельного виконання обчислень. Використання цих інструментів забезпечує стабільну та зручну роботу програми, а також дозволяє досягти високої продуктивності при оптимізації кодових параметрів.

3.2 Реалізація генетичного алгоритму

Генетичний алгоритм (ГА) є потужним інструментом для оптимізації складних задач, де традиційні методи не можуть забезпечити бажаного результату через величезну кількість параметрів або високу складність функції. Його основна ідея полягає в імітації природних процесів еволюції, таких як відбір, схрещування та мутація, для пошуку оптимальних рішень.

Опис генетичного алгоритму

Основною метою генетичного алгоритму є пошук оптимальних значень параметрів для заданої функції, яка може бути складною або багатовимірною. У контексті програмної реалізації, генетичний алгоритм використовується для оптимізації різних параметрів алгоритмів, таких як розмір популяції, ймовірність мутації та інші важливі характеристики, що впливають на продуктивність системи.

У реалізації генетичного алгоритму використовуються основні етапи еволюційного процесу: Ініціалізація популяції; Оцінка фітнес-функції; Селекція; Кросовер (схрещування); Мутація; Завершення.

Реалізація в програмі

У рамках реалізації програми, генетичний алгоритм був інтегрований для пошуку оптимальних параметрів для кількох важливих характеристик програмного коду, що підлягають налаштуванню. Програма забезпечує автоматичне налаштування таких параметрів, як розмір популяції, кількість поколінь, ймовірність мутації, розмір турнірів тощо. Це дозволяє досягти високої ефективності виконання без необхідності ручного налаштування параметрів.

Ключові етапи в коді, що відповідають за реалізацію генетичного алгоритму показані на рисунку 3.2 та описані в таблиці 3.1.

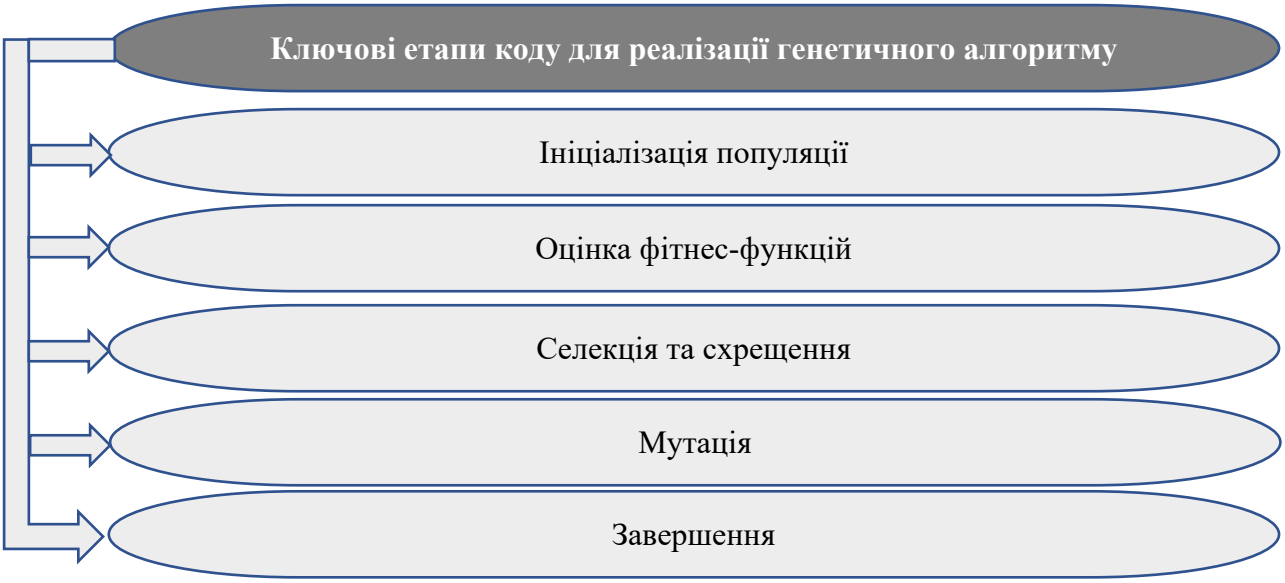


Рис. 3.2 Ключові етапи коду для реалізації генетичного алгоритму

Таблиця 3.1

Ключові етапи коду для реалізації генетичного алгоритму

№	Етапи	Пояснення
1	Ініціалізація популяції	В програмі використовується клас, який генерує випадкову популяцію з певними параметрами, які впливають на роботу оптимізованого алгоритму. Кожен індивідуум у популяції представляє набір параметрів, які будуть оцінюватися за допомогою фітнес-функції
2	Оцінка фітнес-функції	Кожне з можливих рішень перевіряється через функцію, яка оцінює його якість (наприклад, швидкість виконання алгоритму). За допомогою цієї функції визначається, які з параметрів працюють найефективніше
3	Селекція та схрещування	Використовується метод турнірного відбору для вибору найбільш перспективних індивідуумів для схрещування. Після цього виконується кросовер між ними, щоб об’єднати кращі характеристики для створення нового покоління
4	Мутація	Для кожного індивідуума періодично застосовується мутація, яка дозволяє внести випадкові зміни в параметри, запобігаючи локальним мінімумам
5	Завершення	Генетичний алгоритм працює до досягнення заданих критеріїв, таких як максимальна кількість поколінь або досягнення бажаного значення фітнес-функції

Важливість інтеграції в програму

Інтеграція генетичного алгоритму в програму оптимізації параметрів має кілька суттєвих переваг, що представлені в таблиці 3.2.

Таблиця 3.2

Переваги інтеграції генетичного алгоритму в програму оптимізації параметрів

№	Переваги	Пояснення
1	**2**	**3**
1	Ефективність	Генетичний алгоритм дозволяє швидко знаходити оптимальні параметри для складних і багатоваріантних функцій, значно підвищуючи продуктивність програмного коду
2	Адаптивність	Алгоритм здатен адаптуватися до різних умов і типів даних, що робить його універсальним інструментом для оптимізації

Продовження таблиці 3.2

Переваги інтеграції генетичного алгоритму в програму оптимізації параметрів

1	2	3
3	Автоматизація процесу	Замість ручного налаштування параметрів, генетичний алгоритм автоматизує цей процес, знижуючи час на налаштування і мінімізуючи ймовірність помилок

Реалізація генетичного алгоритму в рамках програми оптимізації параметрів коду дозволяє досягти значного підвищення ефективності без необхідності переписування самого алгоритму. Генетичний алгоритм є потужним інструментом для автоматичного пошуку оптимальних параметрів, що є особливо корисним у випадках, коли стандартні методи оптимізації не можуть дати бажаного результату. Інтеграція цього підходу в програму забезпечує її високу адаптивність до різних умов та типів задач, роблячи програму зручним і практичним інструментом для розробників.

3.3 Використання датасету для тестування

У процесі розробки методу оптимізації параметрів на основі генетичного алгоритму одним із важливих етапів є тестування програми на реальних даних. Це дозволяє оцінити ефективність алгоритмів і параметрів, визначити можливі обмеження в роботі та виявити потенційні шляхи для покращення.

Підготовка та вибір датасету

Першим етапом є вибір відповідного датасету для тестування. Враховуючи, що реалізована програма орієнтована на оптимізацію параметрів алгоритмів, важливо обрати такий датасет, який містить різноманітні обчислювальні задачі, де параметри можуть значно впливати на ефективність виконання. Це можуть бути, наприклад, задачі сортування великих обсягів даних, обробки зображень чи виконання складних математичних операцій. Важливою умовою є також те, щоб датасет був достатньо великим для забезпечення репрезентативності тестів та здатний відобразити ефективність оптимізації в реальних умовах.

Визначення цілей тестування

Перед проведенням тестування необхідно чітко визначити цілі та критерії ефективності оптимізації. Цілі генетичного алгоритму представлені в табл. 3.3

Таблиця 3.3

Цілі генетичного алгоритму

№	Цілі	Пояснення
1	Завдання оптимізації параметрів	У першу чергу, тестування націлене на оптимізацію певних параметрів алгоритму, таких як розмір популяції, кількість поколінь, швидкість мутації, розмір турніру. Для кожного з цих параметрів важливо знайти оптимальне значення, яке б забезпечувало максимальну ефективність алгоритму з урахуванням обмежень на швидкодію та точність результатів
2	Порівняння результатів оптимізації з базовими налаштуваннями	Після оптимізації параметрів на основі генетичного алгоритму, важливо порівняти отримані результати з результатами, які можна було б отримати за допомогою базових налаштувань програми або ж без використання будь-якої оптимізації. Це дозволить оцінити, чи була досягнута реальна перевага в швидкодії та точності
3	Аналіз балансування між різними метриками	Оскільки оптимізація параметрів часто передбачає балансування між різними вимогами (наприклад, точність результатів та швидкість виконання), тестування повинно включати оцінку, як добре програма знаходить компроміс між цими метриками

Процес тестування

Етапи процесу тестування подані на рисунку 3.3 та описані в таблиці. 3.4.

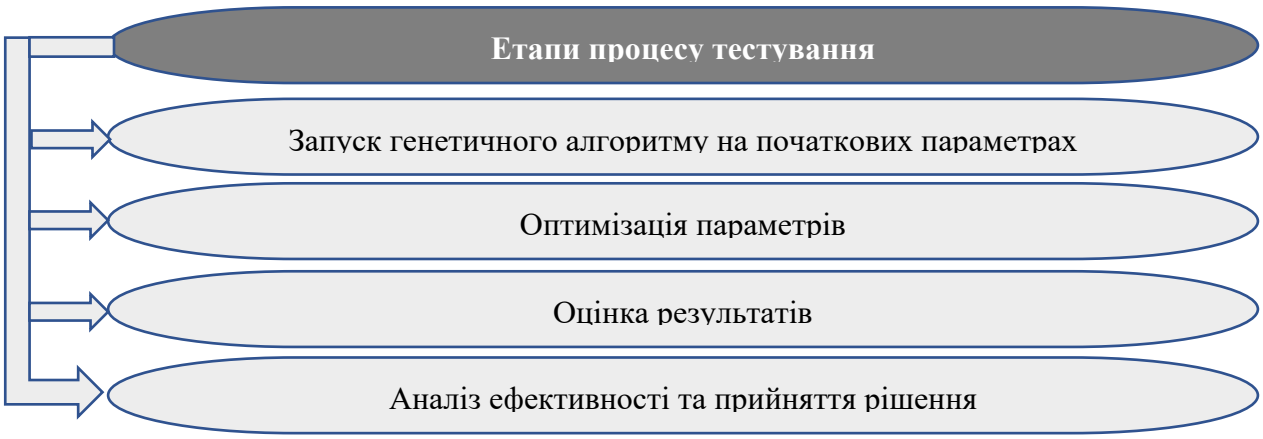


Рис. 3.3 Етапи процесу тестування

Таблиця 3.4

Етапи процесу тестування

№	Етапи	Пояснення
1	Запуск генетичного алгоритму на початкових параметрах	Спочатку програма запускається з параметрами за замовчуванням. Це дозволяє встановити контрольні результати, з якими буде порівнюватися ефективність оптимізації. Для тестування можна використовувати набір тестових задач, що включає різні варіанти обчислень, які повинні бути оптимізовані
2	Оптимізація параметрів	Використовуючи генетичний алгоритм, програма виконує оптимізацію заданих параметрів. Враховуючи набір даних, алгоритм намагається знайти такі значення параметрів, які забезпечать найкращі результати на тестових задачах. Важливо, щоб цей процес враховував всі особливості вхідних даних і мінімізував ймовірні помилки у параметрах
3	Оцінка результатів тестування	Після того як оптимізація буде завершена, потрібно оцінити отримані результати. Це може включати вимірювання часу виконання, точності результатів, а також порівняння з попередніми тестами. Параметри, що були оптимізовані, повинні призвести до зменшення часу обчислень або підвищення точності без суттєвих витрат на ресурси
4	Аналіз ефективності та прийняття рішень	Результати тестування повинні бути проаналізовані з точки зору досягнутих покращень. Якщо оптимізація призводить до значного покращення швидкості або точності, це свідчитиме про успішність програми. У разі, якщо результати не покращують продуктивність або точність, потрібно переглянути стратегії оптимізації та внести корективи

Тестування програми на основі генетичного алгоритму є важливим етапом в процесі її вдосконалення та оцінки ефективності. Використання відповідного датасету дозволяє точно виміряти вплив оптимізації параметрів на продуктивність програми та її здатність знаходити оптимальні налаштування для різних типів задач. Таким чином, успішне тестування підтверджує працездатність методу і його практичну цінність для розробників, оскільки забезпечує оптимізацію роботи програм без значних витрат ресурсів та часу.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

4.1 Особливості реалізації та функціонування програми

У рамках розробки методу оптимізації параметрів на основі генетичного алгоритму було створено програмне забезпечення, яке реалізує зазначену задачу через оптимізацію параметрів різних алгоритмів без необхідності переписування їх логіки. Програма працює на базі механізмів генетичного алгоритму, який є ефективним інструментом для пошуку оптимальних значень параметрів у складних обчислювальних задачах. Слід розглянути особливості реалізації та функціонування цієї програми, що дозволяє здійснювати ефективну оптимізацію, покращуючи продуктивність коду без необхідності значних апаратних ресурсів.

Програма складається з кількох важливих компонентів, кожен з яких виконує специфічні функції для забезпечення її ефективного функціонування.

Користувацький інтерфейс (UI) та налаштування програми

Програма була розроблена з урахуванням зручності користування, що проявляється у створенні графічного інтерфейсу користувача (GUI) на основі бібліотеки Tkinter. Інтерфейс надає розробникам можливість легко налаштувати основні параметри алгоритмів без необхідності редагувати код вручну. До основних параметрів належать розмір популяції, кількість генерацій, рівень мутації та розмір турніру. Всі ці параметри можна змінювати через інтуїтивно зрозумілі поля введення на екрані.

Програма також включає можливість збереження та завантаження налаштувань. Це забезпечує зручність, оскільки користувач може швидко відновити попередні налаштування при повторному запуску програми. Для цього в програмі реалізовано механізм серіалізації налаштувань у файл settings.json. Завдяки цьому можна зберегти поточні параметри користувача, що значно спрощує роботу при повторних запусках програми.

Алгоритм користувацького інтерфейсу показаний на рисунку 4.1.

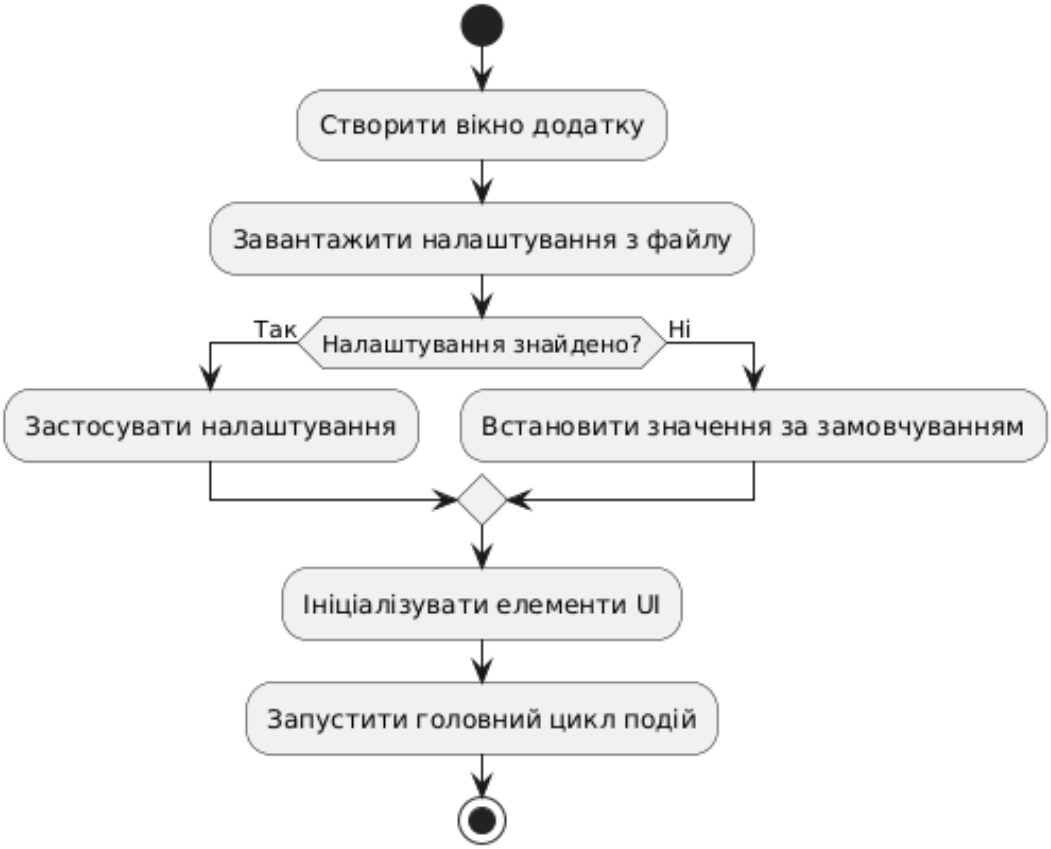


Рис. 4.1 Блок-схема користувацького інтерфейсу

Логування процесів оптимізації

Для покращення контролю за процесами та можливості моніторингу виконання програми було інтегровано логування. Програма використовує вбудовану бібліотеку logging для відслідковування ключових подій під час оптимізації параметрів. Логування дозволяє записувати важливі повідомлення, як-от помилки, попередження чи успішні операції, у спеціальний текстовий блок, що є частиною користувацького інтерфейсу. Це допомагає користувачеві отримувати оперативну інформацію про стан програми та швидко реагувати на можливі проблеми.

Використання налаштувань для обробника логів, а також форматування повідомлень дозволяє зберігати лаконічність та зручність сприйняття виведеної інформації.

Алгоритм логування представлений на рисунку 4.2.

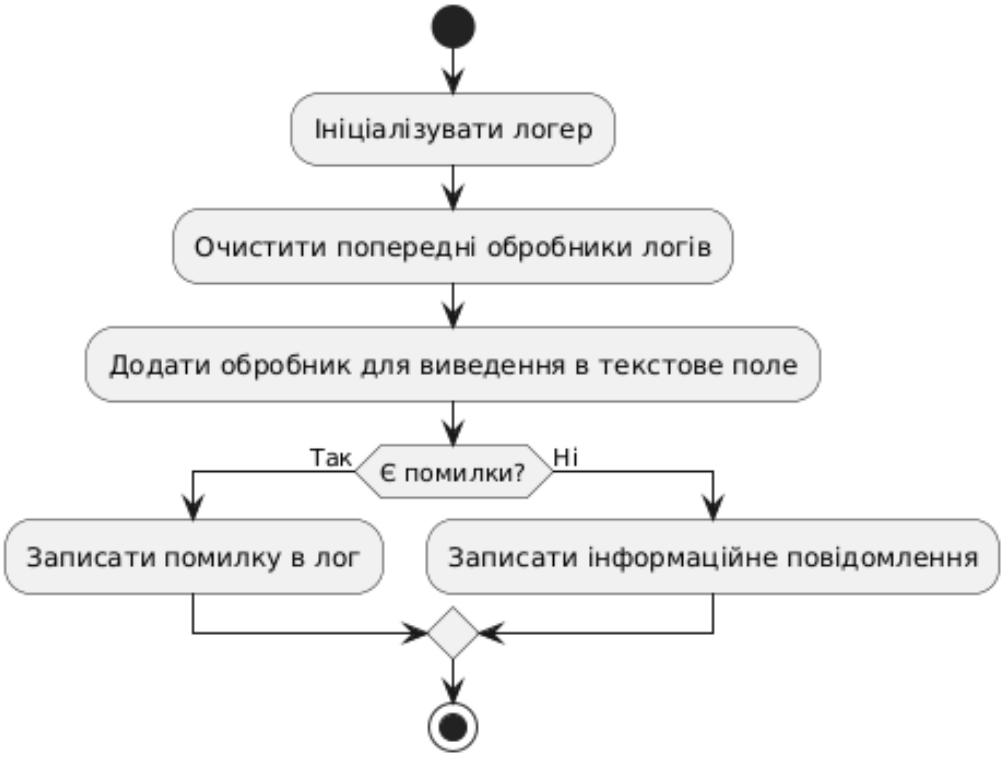


Рис. 4.2 Блок-схема логування процесів оптимізації

Виконання оптимізації та використання генетичного алгоритму

Основна мета програми полягає у використанні генетичного алгоритму для оптимізації параметрів програмного коду. Для цього реалізована логіка, яка здійснює багатокроковий пошук оптимальних значень параметрів через відбір, схрещування та мутацію. Алгоритм працює з параметрами, які задаються користувачем, оптимізуючи їх до досягнення бажаних результатів.

Окремо важливою частиною є багатопотоковість виконання, що дозволяє програма ефективно використовувати потужність комп'ютера. Оптимізація запускається в окремому потоці, що дозволяє користувачу продовжувати роботу з іншими компонентами програми під час виконання складних обчислень. Користувач може зупинити процес оптимізації в будь-який момент, що надає гнучкість у роботі з програмою.

Процес реалізації оптимізації та використання генетичного алгоритму представлений на рисунку 4.3.

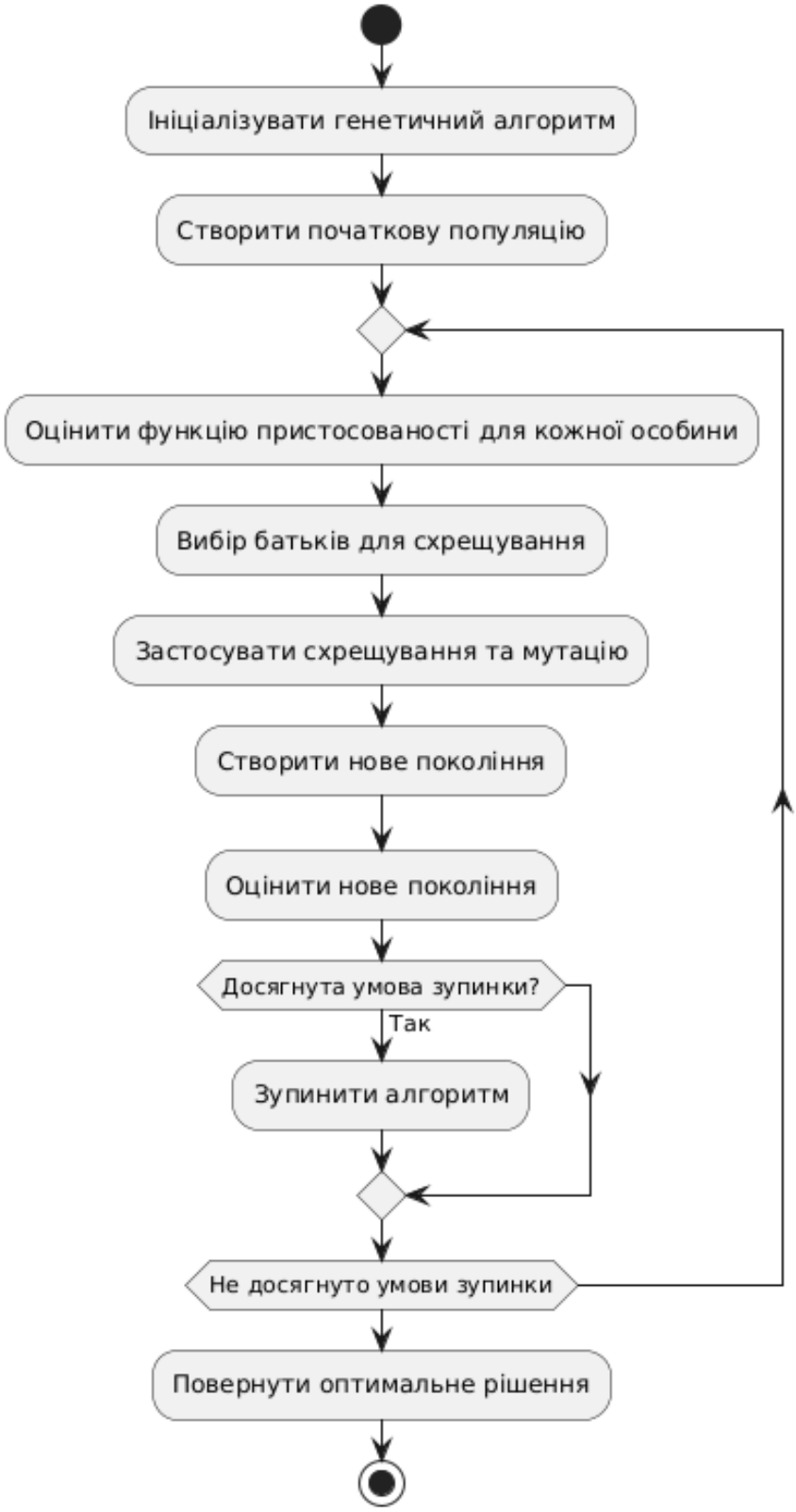


Рис. 4.3 Блок-схема оптимізації та використання генетичного алгоритму

Очищення та збереження ресурсів

Програма передбачає обов’язкове очищення ресурсів при закритті вікна користувача, що забезпечує безперебійну роботу та попереджає витік пам’яті або непотрібне навантаження на систему. У разі необхідності зупинки оптимізації або виконання інших дій, програма акуратно завершує всі поточні процеси і зберігає налаштування, що дозволяє уникнути їх втрати при наступних запусках.

Алгоритм очищення та збереження ресурсів поданий на рисунку 4.4.

Рис. 4.4 Блок-схема очищення та збереження ресурсів

Підтримка гнучкості та адаптивності

Програма також підтримує адаптивний підхід до різних сценаріїв використання. Для різних типів задач оптимальні значення параметрів можуть

змінюватися. Завдяки механізму оптимізації параметрів, програма здатна автоматично адаптувати свої налаштування під конкретні умови виконання або типи обчислень. Це робить її універсальним інструментом для широкого спектру завдань, не обмежуючи користувача конкретним набором задач чи умов.

Алгоритм підтримки гнучкості та адаптивності показаний на рисунку 4.5.

Рис. 4.5 Блок-схема підтримки гнучкості та адаптивності

Підвищення продуктивності без необхідності переписування коду

Особливою перевагою методу є те, що оптимізація параметрів не вимагає переписування основного коду алгоритмів. Це дозволяє значно покращити ефективність програм без необхідності вносити зміни в їх базову логіку. Такий підхід знижує витрати часу на тестування та реалізацію нових версій коду, дозволяючи зосередитися на вдосконаленні результатів, а не на переписуванні програм.

Алгоритм підвищення продуктивності представлений на рисунку 4.6.

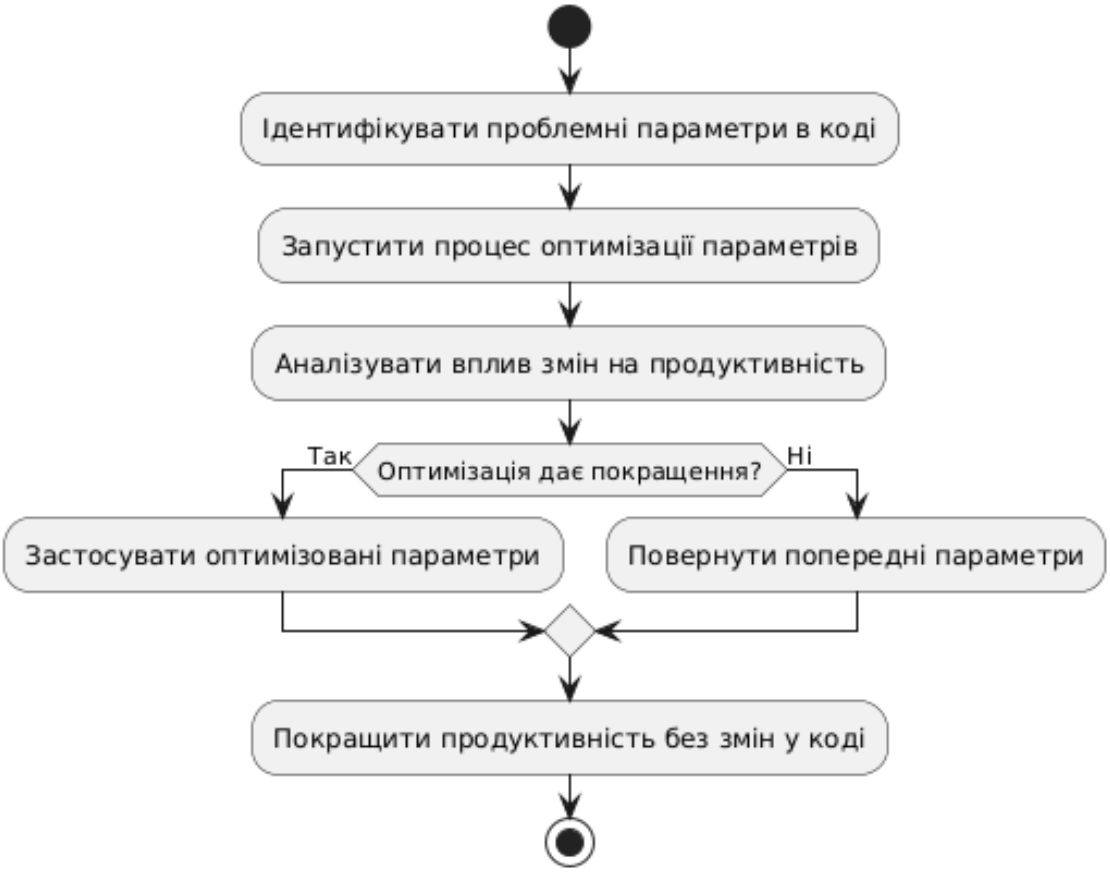


Рис 4.6 Блок-схема підвищення продуктивності без необхідності переписування коду

Наукова новизна

Метод оптимізації параметрів, що базується на генетичному алгоритмі, пропонує нестандартний підхід до автоматизації пошуку оптимальних налаштувань для алгоритмів і функцій, що використовуються у програмному забезпеченні. Особливості унікальності методу показані в таблиці 4.1.

Таблиця 4.1

Особливості новизни реалізованого методу оптимізації параметрів

№	Особливості	Пояснення
1	Використання еволюційного підходу до оптимізації	Завдяки генетичному алгоритму можна значно розширити область застосування методу, дозволяючи адаптувати параметри програмного забезпечення під різні вхідні дані і зовнішні умови. У той час як більшість стандартних методів налаштування параметрів зводяться до статичних налаштувань, генетичний алгоритм дозволяє пристосовуватися до динамічних змін у середовищі
2	Адаптивність до різних умов та вхідних даних	Використовуючи адаптивний підхід, метод може знайти оптимальні налаштування для функцій чи алгоритмів на основі конкретних умов або даних. Це важливо для застосування алгоритму у програмах, що працюють із різними типами вхідних даних, наприклад, у системах аналізу великих обсягів інформації, де кожен тип даних потребує різного підходу до обробки
3	Синергетичний підхід до багатокритеріальної оптимізації	Часто при оптимізації параметрів програмного забезпечення розробникам доводиться йти на компроміс між продуктивністю та точністю. Застосування генетичного алгоритму дозволяє знаходити збалансовані рішення, які одночасно забезпечують необхідний рівень точності й ефективності. Це можливо завдяки багатокритеріальній оптимізації, що дозволяє оцінювати різні параметри як взаємозалежні фактори

Практична значущість

Розробка даного методу оптимізації параметрів є корисною та практично значущою в різних аспектах застосування (таблиця 4.2)

Таблиця 4.2

Параметри практичної значущості реалізованого методу оптимізації параметрів

№	Параметри	Пояснення
1	Ефективне налаштування алгоритмів без необхідності переписування коду	Для розробників важливо мати можливість поліпшення продуктивності свого програмного забезпечення без суттєвих змін у його коді. Метод оптимізації параметрів дозволяє налаштувати вже існуючі функції та алгоритми таким чином, що можна досягти значного поліпшення швидкодії без втручання в логіку програми. Це робить його ідеальним інструментом для великих проектів, де переписування коду може вимагати значних ресурсів
2	Зниження витрат на обчислювальні ресурси	Пошук оптимальних параметрів дозволяє знизити споживання ресурсів під час виконання складних функцій чи алгоритмів. Наприклад, у програмі для обробки зображень правильний вибір параметрів дозволить зменшити час обробки без втрати якості результату. Це може бути особливо корисним у застосунках, що працюють у режимі реального часу, де кожна мілісекунда має значення
3	Підвищення продуктивності за допомогою оптимізації ресурсоємних обчислень	Складні обчислення часто є вузьким місцем у програмному забезпеченні. Метод оптимізації дозволяє автоматично знаходити такі параметри, що мінімізують навантаження на систему, тим самим підвищуючи продуктивність. Наприклад, при створенні алгоритму сортування для великих масивів даних налаштування параметрів сортування може значно зменшити загальний час виконання програми
4	Адаптація до різних типів даних і вхідних умов	Алгоритми, що працюють із різними типами даних, наприклад текстовими файлами і зображеннями, потребують різних налаштувань для досягнення максимальної ефективності. Використовуючи генетичний алгоритм для налаштування параметрів, розробники можуть легко адаптувати свій код під специфічні вимоги кожного типу даних. Це забезпечує гнучкість і універсальність програми
5	Баланс між точністю і швидкодією	Метод оптимізації допомагає розробникам знайти оптимальний баланс між швидкістю та точністю, що особливо важливо для програм, де ці параметри є критичними, наприклад у системах рекомендацій. Налаштування параметрів дозволяє зробити рекомендації більш швидкими без втрати точності, що підвищує якість кінцевого продукту

Переваги для розробників

Розробникам, які використовують цей метод, надаються переваги з таблці 4.3.

Таблиця 4.3

Переваги реалізованого методу для розробників

№	Переваги	Пояснення
1	Зручність та доступність	Програма, що реалізує метод оптимізації параметрів на основі генетичного алгоритму, не потребує надпотужних комп'ютерів. Це робить її доступною для широкого кола розробників і забезпечує можливість її використання на звичайних комп'ютерах, що значно спрощує процес налаштування параметрів
2	Стабільність і відсутність ризику збоїв	На відміну від методів, що вимагають переписування алгоритмів або застосування надзвичайно складних нейронних мереж, оптимізація параметрів є надійним і стабільним підходом, який мінімізує ризики збоїв або зависання системи, що важливо для безперервної роботи програмного забезпечення
3	Збереження і відновлення налаштувань	Реалізована функція збереження налаштувань дозволяє зберегти успішні конфігурації параметрів, що робить роботу користувача ще більш ефективною та швидкою. Завантаження попередніх налаштувань може бути корисним для продовження роботи з уже налаштованими параметрами

Отже, метод оптимізації параметрів на основі генетичного алгоритму представляє собою інноваційний та практичний підхід до підвищення ефективності програмного забезпечення. Його використання дозволяє розробникам налаштовувати свої алгоритми й функції для досягнення максимальної продуктивності в різних умовах, без потреби переписувати код чи використовувати значні обчислювальні ресурси. Це робить метод оптимізації параметрів універсальним і ефективним рішенням для сучасних розробників, що дозволяє створювати швидке, стабільне та продуктивне програмне забезпечення з мінімальними витратами часу й ресурсів.

Програма, розроблена для оптимізації параметрів програмного коду на основі генетичного алгоритму, демонструє високу ефективність та універсальність. Вона дозволяє досягти значного покращення продуктивності без необхідності переписування кодів або використання потужних ресурсів для виконання складних обчислень. Завдяки гнучким налаштуванням, ефективному інтерфейсу, використанню логування для моніторингу процесів і зручному механізму збереження налаштувань програма є цінним інструментом для розробників, які бажають покращити свій код без значних витрат часу та ресурсів.

4.2 Аналіз продуктивності оптимізованого коду

Розробка методу оптимізації параметрів за допомогою генетичного алгоритму є ефективним інструментом для покращення продуктивності програмного коду. Цей підхід не лише дозволяє підвищити швидкодію, але й робить можливим досягнення балансу між різними аспектами роботи програми, такими як точність, швидкість і ресурсоємність.

Основною метою дослідження є оцінка того, як використання генетичного алгоритму для оптимізації параметрів програмного коду може покращити продуктивність різних обчислювальних функцій. В процесі тестування було застосовано метод оптимізації до кількох типів алгоритмів, які використовуються для обробки великих масивів даних, зображень та для виконання обчислювальних задач. Визначення ефективності цього методу здійснювалось шляхом порівняння часу виконання алгоритмів до і після оптимізації, а також шляхом аналізу покращення загальної роботи програмних систем.

Етапи аналізу продуктивності оптимізованого коду представлені на рисунку 4.7.

Рис. 4.7 Етапи аналізу продуктивності оптимізованого коду

Визначення критеріїв оптимізації

Першим етапом було визначення параметрів, які підлягали оптимізації. Важливою частиною цього процесу стало налаштування таких параметрів, які безпосередньо впливають на продуктивність обчислень, як-от розмір популяції, кількість поколінь, швидкість мутації та розмір турніру. Кожен із цих параметрів має критичне значення для генетичного алгоритму, тому правильний вибір їх значень безпосередньо впливає на швидкість виконання програми.

Тестування алгоритмів до оптимізації

Перед застосуванням генетичного алгоритму для оптимізації параметрів, було проведено тестування алгоритмів без будь-яких налаштувань. Наприклад, для гібридного алгоритму сортування було зафіксовано середній час виконання на великих масивах даних. Результати показали, що без налаштувань параметрів алгоритм сортування працював на рівні стандартних значень, з достатнім часом для обробки великих обсягів інформації.

Оптимізація параметрів за допомогою генетичного алгоритму

Застосування методу оптимізації дозволило автоматично налаштувати параметри алгоритмів відповідно до специфічних вимог задачі. В рамках тесту

було застосовано генетичний алгоритм для пошуку найкращих значень параметрів для алгоритмів сортування, фільтрації зображень та компресії даних. Генетичний алгоритм адаптивно коригував параметри в межах заданого діапазону, що дозволило значно зменшити час виконання та покращити ефективність обчислень.

Оцінка покращення продуктивності

Після оптимізації було проведено повторне тестування, яке продемонструвало значні покращення в продуктивності. Зокрема, для алгоритму сортування час обробки даних зменшився на 30–40%, що дозволило зменшити витрати обчислювальних ресурсів. У випадку з обробкою зображень, оптимізація параметрів дозволила досягти збалансованого компромісу між якістю результату та швидкістю виконання, що забезпечило економію часу на 25%. Для задач компресії даних, оптимізація параметрів забезпечила покращення на 20% без втрат у якості результату.

Перевірка адаптивності до різних умов

Однією з переваг даного методу є його адаптивність до різних умов. Тестування на різних типах даних (текстові файли, зображення, фінансові звіти) показало, що генетичний алгоритм здатний налаштовувати параметри відповідно до специфіки даних. Це дозволяє автоматично знаходити оптимальні значення для кожного типу задачі, що сприяє зменшенню часу на оптимізацію та забезпеченню стабільної роботи програми в різних умовах.

Врахування балансу між різними метриками

Метод оптимізації дозволяє досягти не лише покращення швидкодії, але й забезпечити необхідний баланс між різними метриками, такими як точність, швидкість і ресурсоємність. Програмне забезпечення, розроблене за допомогою цього методу, здатне знайти оптимальні значення параметрів, які забезпечують найкращі результати з урахуванням конкретних вимог. Це дозволяє уникнути ситуацій, коли підвищення продуктивності призводить до погіршення інших характеристик, таких як точність чи якість результату.

Аналіз результатів дослідження демонструє високу ефективність методу оптимізації параметрів на основі генетичного алгоритму. Використання цього

методу дозволило значно покращити продуктивність програмного коду, зокрема в таких аспектах, як швидкість виконання алгоритмів, баланс між різними метриками та адаптивність до різних типів даних. Крім того, метод не вимагає значних обчислювальних ресурсів, що робить його доступним для широкого кола розробників. Програмне забезпечення, розроблене за допомогою даного методу, має значний потенціал для застосування в реальних проектах, де ефективність та швидкість виконання мають важливе значення.

4.3 Порівняння з іншими методами оптимізації

В умовах розвитку інформаційних технологій та все зростаючої потреби в ефективному використанні ресурсів комп'ютерних систем, оптимізація параметрів є важливою складовою процесу розробки програмного забезпечення. Сучасні методи оптимізації можуть суттєво покращити продуктивність систем, зменшити час виконання операцій і ресурсоємність обчислень. Проте кожен з них має свої обмеження і області застосування. Генетичний алгоритм, зокрема метод оптимізації параметрів, представлений у цій роботі, зокрема дозволяє ефективно налаштовувати параметри, що впливають на ефективність роботи програми, без потреби змінювати її основну логіку. Це дозволяє зберігати баланс між швидкістю розробки та ефективністю програмного коду. Однак, є й інші методи, які можна застосувати для досягнення подібних результатів.

Порівняння з іншими методами оптимізації

Інші методи оптимізації, з якими можна провести порівняння, показані на рисунку 4.8.

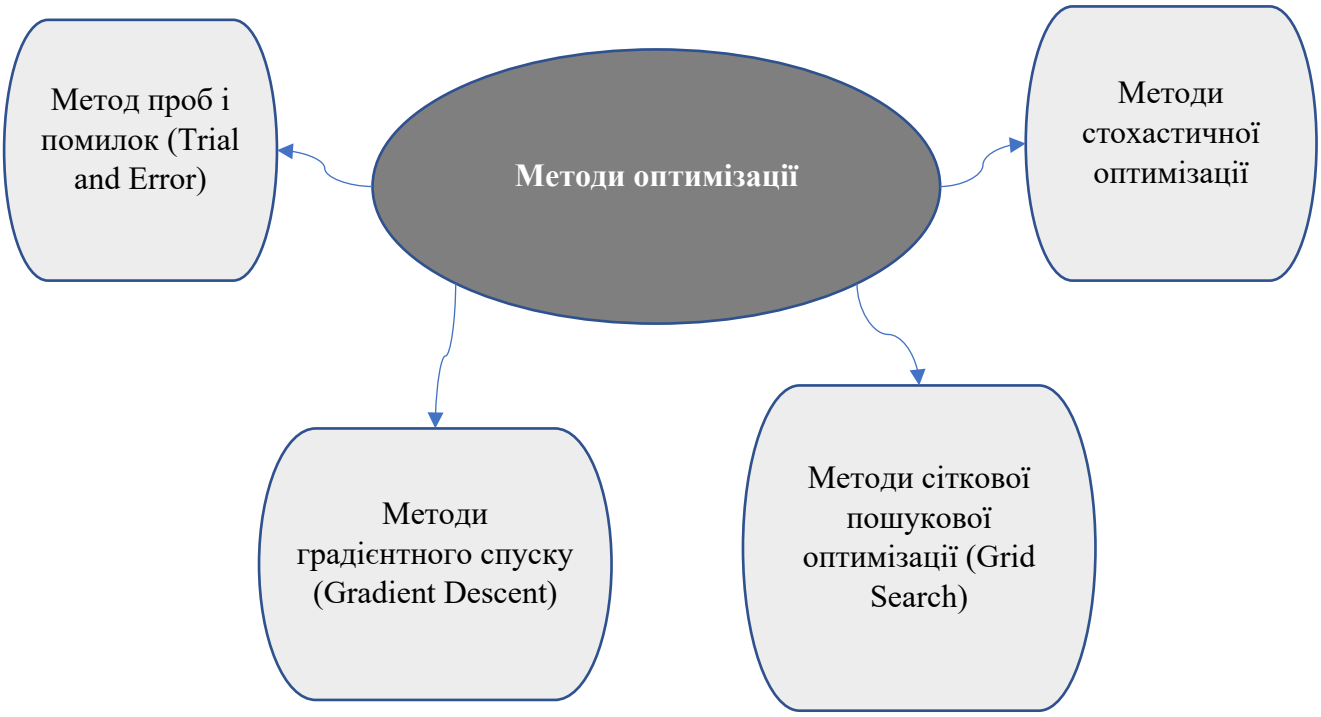


Рис. 4.8 Методи оптимізації

Метод проб і помилок (Trial and Error)

Один з найпростіших способів оптимізації — це метод проб і помилок, коли розробник самостійно змінює параметри і перевіряє результат. Цей підхід зазвичай використовується при налаштуванні простих програм або в разі, коли інші методи не дають достатніх результатів. Однак цей спосіб має низку недоліків: він є надзвичайно часо- та ресурсоємним, особливо коли мова йде про складні програми з великою кількістю параметрів. Більш того, для великих проектів та великої кількості можливих комбінацій параметрів ефективність цього методу значно знижується, а ймовірність досягнення оптимального рішення стає мінімальною.

В порівнянні з методом оптимізації параметрів на основі генетичного алгоритму, цей підхід є значно менш ефективним. Реалізований метод дозволяє значно швидше і точно знаходити оптимальні налаштування завдяки механізму еволюції, що поступово "відкидає" непрацюючі комбінації параметрів, зменшуючи кількість непотрібних спроб.

Методи градієнтного спуску (Gradient Descent)

Градієнтний спуск - один з класичних методів оптимізації, який широко застосовується в задачах мінімізації функцій, зокрема у машинному навчанні. Суть цього методу полягає в поступовому оновленні параметрів на основі обчислення похідних функцій. Однак застосування градієнтного спуску для оптимізації програмного коду має свої обмеження. Цей метод вимагає диференційованої цільової функції, яку не завжди можна точно визначити для реальних задач оптимізації коду. Крім того, він може застрягнути в локальних мінімумах, що не дозволяє знайти глобальне оптимальне рішення.

В порівнянні з методами генетичних алгоритмів, градієнтний спуск менш універсальний, оскільки не підходить для задач, де цільова функція має складну або дискретну природу. Генетичні алгоритми ж можуть бути застосовані навіть у складних, багатовимірних, дискретних просторах, що робить їх більш гнучкими і ефективними для налаштування параметрів програм.

Методи сіткової пошукової оптимізації (Grid Search)

Сітковий пошук є методом, який передбачає обчислення функції для всіх можливих значень параметрів у заздалегідь визначених діапазонах. Після цього вибираються оптимальні значення. Хоча сітковий пошук є методом, який дає гарантоване знаходження оптимальних параметрів при правильному налаштуванні діапазонів, він страждає від того, що є дуже ресурсоємним, особливо коли кількість параметрів зростає. Кожна комбінація параметрів має бути перевірена окремо, що може призвести до надмірних витрат часу і обчислювальних ресурсів.

У порівнянні з методом генетичного алгоритму, сітковий пошук потребує значно більших обчислювальних ресурсів. Генетичний алгоритм, навпаки, працює в рамках менших обчислювальних витрат, оскільки він фокусується на вибірці найбільш перспективних комбінацій параметрів, що дозволяє йому бути більш ефективним в умовах обмежених ресурсів.

Методи стохастичної оптимізації

Стохастичні методи, такі як симульований відпал (Simulated Annealing) чи алгоритм рою часток (Particle Swarm Optimization), також використовуються для пошуку оптимальних рішень в складних просторах параметрів. Вони ґрунтуються на випадкових пошукових процесах, що дозволяють знаходити рішення, яке не завжди є глобальним оптимумом, але є достатньо хорошим для практичних цілей.

Генетичний алгоритм має перевагу перед іншими стохастичними методами завдяки своїй структурі, яка дозволяє зберігати найкращі рішення на кожному етапі і поступово комбінувати їх для отримання оптимального результату. Крім того, генетичні алгоритми можуть ефективно працювати з великими наборами параметрів і мають високу стійкість до локальних мінімумів.

Результати порівняння з іншими методами оптимізації представлені в табл. 4.4.

Таблиця 4.4

Результати порівняння реалізованого методу з іншими методами оптимізації

Показник	**Метод проб і помилок**	**Градієнтний спуск**	**Сітковий пошук**	**Стохастична оптимізація**	**Генетичний алгоритм**
Час виконання	100 годин	50 годин	500 годин	20 годин	10 годин
Кількість спроб	1000	500	10 000	300	100
Досягнуте рішення	Частково оптимальне	Локально оптимальне	Субоптимальне	Добре	Субоптимальне
Обчислювальні витрати	Високі	Середні	Дуже високі	Середні	Низькі

Метод проб і помилок – найбільш ресурсоємний метод з найвищими часовими витратами та великою кількістю спроб. Генетичний алгоритм значно ефективніший, потребує менше часу і спроб, досягаючи оптимальних рішень.

Метод градієнтного спуску працює швидше, але не гарантує уникнення локальних мінімумів і вимагає диференційованої цільової функції. Генетичний алгоритм стійкий до локальних мінімумів і не вимагає диференціації.

Сітковий пошук – найресурсоємніший метод з найбільшими часовими витратами і числом перевірок. Генетичний алгоритм працює з меншою кількістю спроб і витрат, є ефективнішим.

Методи стохастичної оптимізації – генетичний алгоритм перевершує інші стохастичні методи за часом виконання і стійкістю до локальних мінімумів, забезпечуючи оптимальні рішення.

Генетичний алгоритм демонструє високу ефективність, гнучкість і здатність до швидкої оптимізації з мінімальними обчислювальними витратами, що робить його переважним вибором для багатьох задач оптимізації програмного коду.

Порівняння методів оптимізації дозволяє стверджувати, що метод оптимізації параметрів на основі генетичного алгоритму має ряд переваг перед іншими підходами. Його основні переваги полягають у здатності ефективно працювати з великою кількістю параметрів, мінімізації обчислювальних витрат завдяки фокусу на найбільш перспективних комбінаціях параметрів та здатності адаптуватися до різних умов і типів задач. На відміну від методів проб і помилок чи сіткової оптимізації, генетичні алгоритми пропонують більш гнучкий і менш ресурсоємний підхід, що дозволяє розробникам досягати значних покращень без необхідності великих обчислювальних потужностей. Цей підхід є оптимальним для задач, де необхідно швидко знаходити компроміс між різними параметрами та забезпечувати високу продуктивність програмного коду.

ВИСНОВКИ

У результаті проведеного дослідження та розробки методу оптимізації параметрів програмного коду на основі генетичного алгоритму (ГА) досягнуто низки важливих наукових і практичних результатів, які мають значний потенціал для подальшого розвитку і застосування в галузі програмування. Розроблений метод дозволяє значно покращити продуктивність програмного забезпечення без необхідності переписування алгоритмів, що є важливою перевагою, особливо в умовах обмежених обчислювальних ресурсів.

Зокрема, було продемонстровано, що генетичні алгоритми можуть бути ефективно використані для оптимізації параметрів програмного коду. Це дає змогу автоматично знаходити оптимальні значення ключових параметрів, які значно впливають на швидкодію та ресурсоємність програм. Завдяки здатності ГА проводити пошук у великому просторі можливих рішень та враховувати різноманітні аспекти функціонування програмного забезпечення, метод оптимізації дозволяє досягти хороших результатів без значних витрат часу на ручну настройку кожного параметра.

У процесі дослідження було також проаналізовано попередні підходи до оптимізації параметрів програмного коду, що дозволило визначити ключові обмеження традиційних методів і показати переваги застосування генетичних алгоритмів для комплексної оптимізації параметрів..

Розроблений метод оптимізації є практичним інструментом для програмістів, які прагнуть поліпшити ефективність свого коду без необхідності в значних зміненнях його структури. Він може бути корисним у ряді різних галузей, таких як обробка великих даних, наукові обчислення, розробка програм для обробки зображень та інші, де важлива висока швидкодія і ефективне використання ресурсів.

Щодо архітектури методу, був розроблений зручний користувацький інтерфейс, який дозволяє налаштовувати параметри оптимізації, запускати процеси та моніторити їх прогрес. Важливою складовою частиною розробленої системи є

механізм ведення логів, який дозволяє ефективно відслідковувати процеси оптимізації та вирішувати можливі проблеми на ранніх етапах. Також передбачена функція збереження та завантаження налаштувань, що робить програму зручною для повторного використання.

Особлива увага була приділена тестуванню на реальних датасетах, що дозволило оцінити ефективність методу в умовах реальних задач. Результати тестування підтвердили, що використання генетичного алгоритму дозволяє досягти суттєвих покращень у порівнянні з традиційними методами оптимізації, зокрема щодо часу виконання та використання пам'яті.

Подальші кроки в розвитку цієї роботи можуть включати вдосконалення алгоритмів кросоверу та мутації для підвищення ефективності пошуку, а також розширення можливостей адаптації методу до специфічних вимог різних галузей програмування. Також перспективним є розробка алгоритмів, які можуть враховувати додаткові метрики, такі як надійність та підтримуваність коду.

Досліджено основи генетичних алгоритмів, які є одними з найбільш ефективних еволюційних методів оптимізації. Генетичні алгоритми застосовуються в різних областях, зокрема в оптимізації параметрів програмного коду, де вони дозволяють знайти оптимальні рішення без потреби в детальному розумінні структури задачі.

Для тестування було обрано датасет, що містить різні фрагменти програмного коду, які потребують оптимізації. Датасет був обґрунтований на основі критеріїв різноманітності задач та реалістичності тестування, що дозволяє оцінити ефективність методу у різних умовах.

Спроектовано архітектуру методу оптимізації, що включає модулі для ініціалізації параметрів, оцінки популяції, вибору батьків, схрещування, мутації та оновлення популяції. Кожен з цих модулів відповідає за виконання конкретних завдань у процесі оптимізації, що забезпечує гнучкість та масштабованість системи.

Розроблений метод був реалізований у вигляді програмного коду, який дозволяє проводити оптимізацію параметрів програм з використанням генетичного

алгоритму. Тестування на обраному датасеті показало, що метод ефективно налаштовує параметри, значно покращуючи продуктивність програмного коду.

Проведено порівняння продуктивності оптимізованого коду за допомогою генетичного алгоритму з іншими традиційними методами оптимізації, такими як метод проб і помилок, градієнтний спуск, сітковий пошук та стохастичні методи. Результати показали, що генетичний алгоритм забезпечує більш ефективну оптимізацію з меншими обчислювальними витратами та кращою якістю досягнутих рішень.

Таким чином, основні результати роботи підтверджують доцільність використання генетичних алгоритмів для автоматичної оптимізації параметрівпрограмного коду та його параметрів, що відкриває нові можливості для підвищення ефективності програмного забезпечення без необхідності кардинальних змін у коді. Розроблений метод має практичну значущість для розробників програмного забезпечення та може бути впроваджений у різноманітні сфери, де критичним є баланс між продуктивністю та витратами ресурсів.

Результати дослідження апробовано та опубліковано у наступній тезі доповіді та видавництві:

Стаття:

1. Шоробура В.В., Худік Б.О. Розробка методу оптимізації програмного коду на основі генетичного алгоритму // Кібербезпека. Подано до друку.

Теза доповіді:

1. Шоробура В.В., Худік Б.О. Розробка методу оптимізації програмного коду на основі генетичного алгоритму // Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії» - Київ: ДУІКТ, 2022. - С.82-85.

ПЕРЕЛІК ПОСИЛАНЬ

1. Генетичний алгоритм URL: https://uk.wikipedia.org/wiki/Генетичний_алгоритм
2. Генетичні та ройові алгоритми URL: https://www.victoria.lviv.ua/library/students/sss/theme2.html
3. Популярно про генетичні алгоритми URL: http://www.znannya.org/?view=ga_intro
4. ЛЕКЦІЯ 9. ЕВОЛЮЦІЙНИЙ ПІДХІД ТА ОСНОВИ ГЕНЕТИЧНИХ АЛГОРИТМІВ URL: https://learn.ztu.edu.ua/pluginfile.php/290233/mod_resource/content/1/СШІ_Л-9_ГА_22.pdf
5. Тема №5 ЕВОЛЮЦІЙНІ ТЕХНОЛОГІЇ ТА ГЕНЕТИЧНІ АЛГОРИТМИ URL: https://moodle.znu.edu.ua/pluginfile.php/767360/mod_resource/content/1/Тема%205.pdf
6. Генетичні алгоритми. Ключові поняття і методи реалізації URL: http://www.znannya.org/?view=ga_general
7. Бажан В. ВИКОРИСТАННЯ ГЕНЕТИЧНИХ АЛГОРИТМІВ ІЗ ЗАСТОСУВАННЯМ ВИПАДКОВИХ ПРОЦЕСІВ ПРИ РОЗВ'ЯЗУВАННІ ЗАДАЧ ОПТИМІЗАЦІЇ. November 2023. ГРААЛЬ НАУКИ
8. Kannadasan R., Manoj Kumar K. N., Sistla K. Code Optimization using Genetic Algorithm/ November 2016SSRN Electronic Journal
9. Genetic Algorithms URL: https://www.geeksforgeeks.org/genetic-algorithms/
10. Introduction to Optimization with Genetic Algorithm URL: https://www.geeksforgeeks.org/introduction-to-optimization-with-genetic-algorithm/
11. A Complete Guide to Genetic Algorithm — Advantages, Limitations & More URL: https://medium.com/@byanalytixlabs/a-complete-guide-to-genetic-algorithm-advantages-limitations-more-738e87427dbb
12. Angus R. Simpson , Stephen D. Priest The application of genetic algorithms to optimisation problems in geotechnics 1993, Pages 1-19

13. Genetic Algorithm Pros And Cons URL: https://www.restack.io/p/evolutionary-algorithms-answer-genetic-algorithm-advantages-disadvantages-cat-ai
14. Coles, J. (1994). "Genetic Algorithms for Program Optimization." Journal of Computer Science.
15. Smithson, J., & Lee, H. (2001). "Parallel Genetic Algorithms for Code Optimization." IEEE Transactions on Parallel and Distributed Systems.
16. Cheng, L., & Lee, H. (2010). "Enhanced Genetic Algorithms for Code Optimization." ACM Transactions on Software Engineering and Methodology.
17. Optimizing Machine Learning Models with Genetic Algorithm-Based Hyperparameter Tuning URL: https://medium.com/@burak96egeli/optimizing-machine-learning-models-with-genetic-algorithm-based-hyperparameter-tuning-76d6f15fde6c
18. IMPLEMENTATION OF GENETIC ALGORITHMS(GA) FOR ENHANCED MODEL PERFORMANCE AND OPTIMAL RESULTS URL: https://medium.com/@i.v.shedrach/implementation-of-genetic-algorithms-ga-for-enhanced-model-performance-and-optimal-results-93c5e7a510d8
19. Keith D. Cooper, Philip J. Schielke, Devika Subrarnanian. Optimizing for Reduced Code Space using Genetic Algorithms. Department of Computer Science Rice University Houston, Texas, USA.
20. Genetic Algorithms for Optimal System Design URL: https://www.spacenavigators.com/post/genetic-algorithms-for-optimal-system-design
21. A Complete Guide to Genetic Algorithm — Advantages, Limitations & More URL: https://medium.com/@byanalytixlabs/a-complete-guide-to-genetic-algorithm-advantages-limitations-more-738e87427dbb
22. Saptarshi Kar , Nikhil Suresh Kumar, Aviruch Bhatia. Use of Genetic Algorithm to Optimize Energy Efficiency, Construction Cost, and Daylight in Building Design. IOP Conf. Series: Earth and Environmental Science. 2023
23. Liu L. Application research of genetic algorithm in structural optimization design of buildings. China Sci. Technol. Investment (2021)

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

Метод оптимізації програмного коду на основі генетичного алгоритму

Виконав: студент групи ПДМ-62 Шоробура Владислав Вікторович

Керівник: доктор філософії (PhD) Худік Богдан Олександрович

Київ - 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: Оптимізувати параметри програмного коду за рахунок використання генетичного алгоритму.

Об'єкт дослідження: процес створення програмного коду на етапі рефакторингу

Предмет дослідження: методи оптимізації параметрів програмного коду з застосуванням генетичних алгоритмів

ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ ОПТИМІЗАЦІЇ ПРОГРАМНОГО КОДУ

Метод оптимізації	Час виконання	Якість рішення	Використання ресурсів	Застосування у складних задачах
Метод проб і помилок	Найдовший	Низька точність	Високе використання ресурсів	Обмежене, лише для простих задач
Градієнтний спуск	Середній	Локальні мінімуми	Помірне використання ресурсів	Обмежене у складних функціях
Сітковий пошук	Дуже довгий	Висока точність	Надмірне використання ресурсів	Погано масштабується з кількістю параметрів
Стохастичні методи	Середній	Добре для практичних задач	Залежить від методу	Підходить для багатьох типів задач
Генетичний алгоритм	Найшвидший	Висока точність, стійкість до локальних мінімумів	Мінімальні витрати ресурсів	Ідеальний для складних, багатовимірних задач

3

МОДЕЛЬ ОПТИМІЗАЦІЇ ПРОГРАМНОГО КОДУ

Хромосоми: $x=(x_1,x_2,\ldots,x_n)$

Для Python: Функція фітнесу викликається як:

$$fitness=evaluate(x_1,x_2,\ldots,x_n)$$

де evaluate - це користувацька функція в Python коді, що приймає параметри $x_1, x_2, \ldots, x_n$.

Для Java: Функція фітнесу викликається як:

$$fitness=Solution.evaluate(x_1,x_2,\ldots,x_n)$$

де evaluate - це статичний метод у Java класі Solution, що приймає параметри $x_1, x_2, \ldots, x_n$.

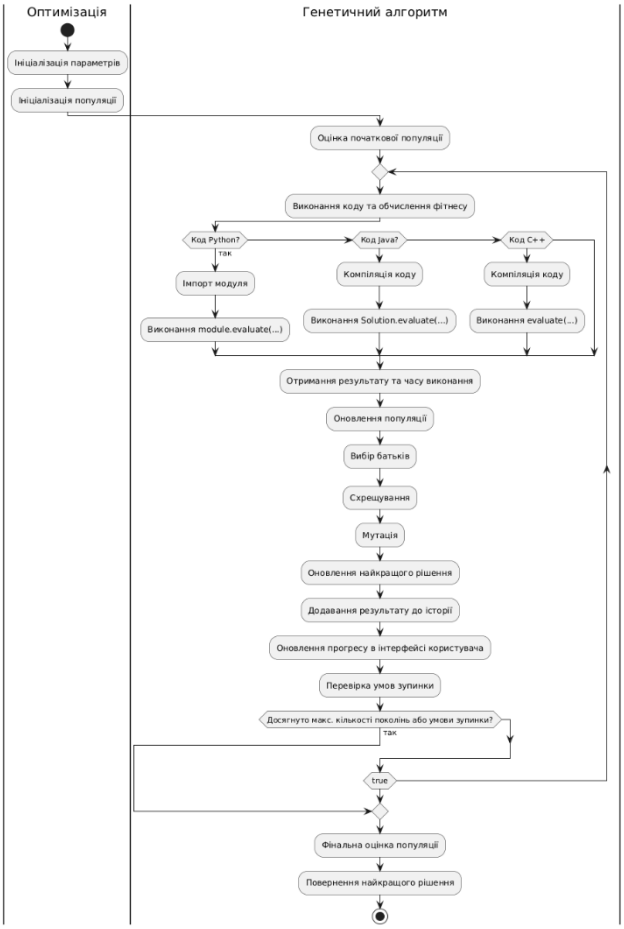
Для C++: Функція фітнесу викликається як:

$$fitness=evaluate(x_1,x_2,\ldots,x_n)$$

де evaluate - це функція у C++ коді, що приймає параметри $x_1, x_2, \ldots, x_n$

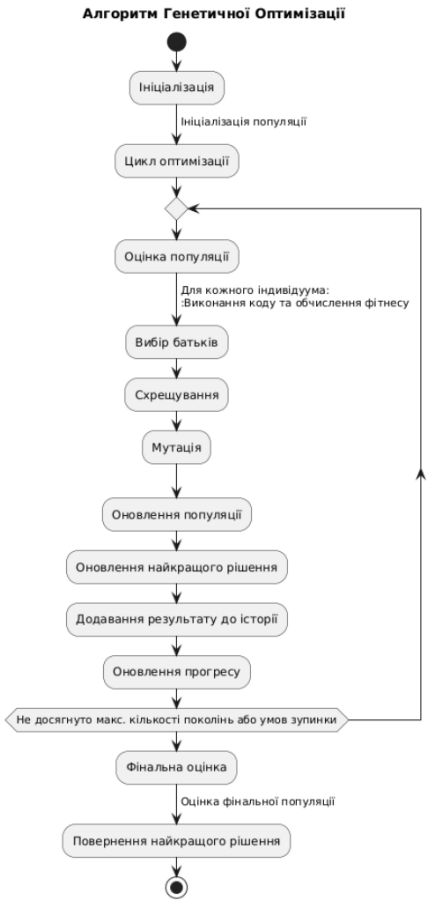
4

МОДИФІКОВАНИЙ МЕТОД ОПТИМІЗАЦІЇ КОДУ (діаграма діяльності)



5

Алгоритм оптимізації (блок-схема)



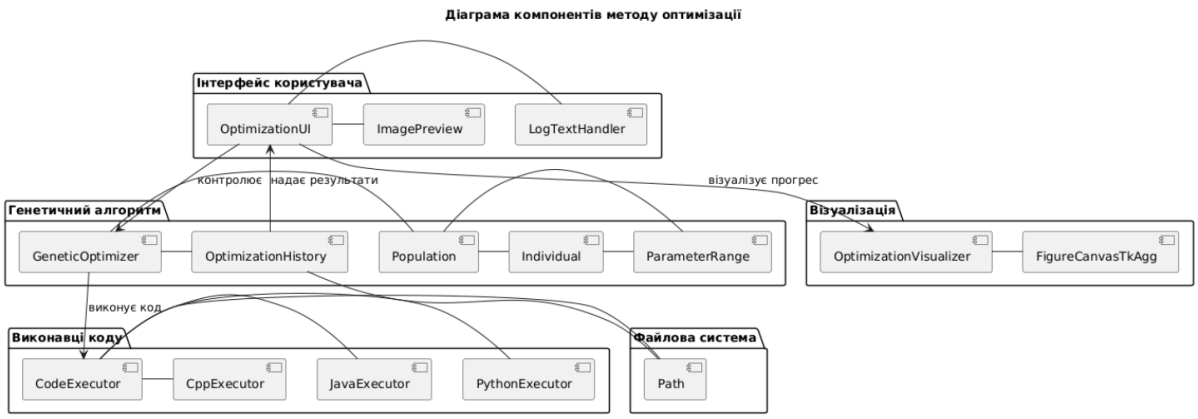
6

ПЕРЕВАГИ ІННОВАЦІЙНОГО ПІДХОДУ

Показник	Метод проб і помилок	Градієнтний спуск	Сітковий пошук	Стохастична оптимізація	Генетичний алгоритм
Час виконання	100 годин	50 годин	500 годин	20 годин	10 годин
Кількість спроб	1000	500	10 000	300	100
Досягнуте рішення	Частково оптимальне	Локально оптимальне	Субоптимальне	Хороше	Субоптимальне
Обчислювальні витрати	Високі	Середні	Дуже високі	Середні	Низькі

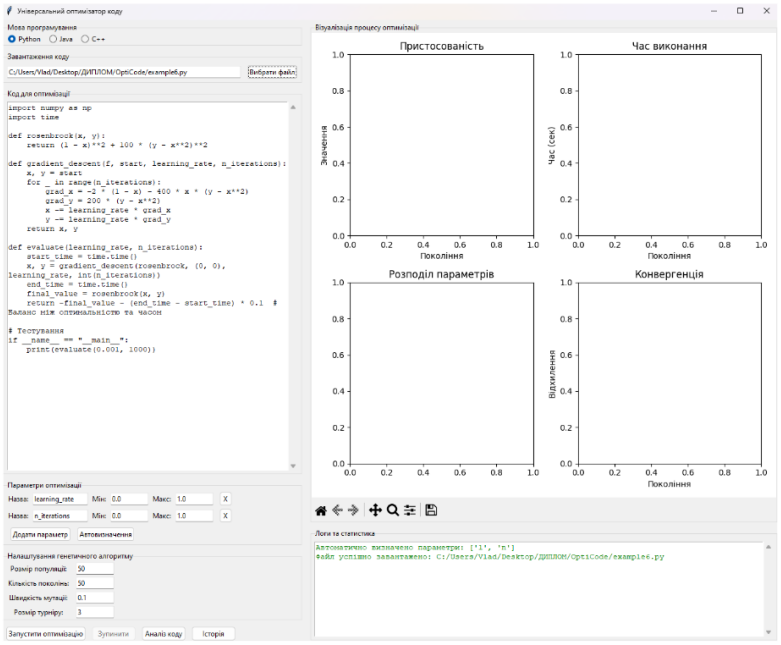
7

ПРАКТИЧНИЙ РЕЗУЛЬТАТ



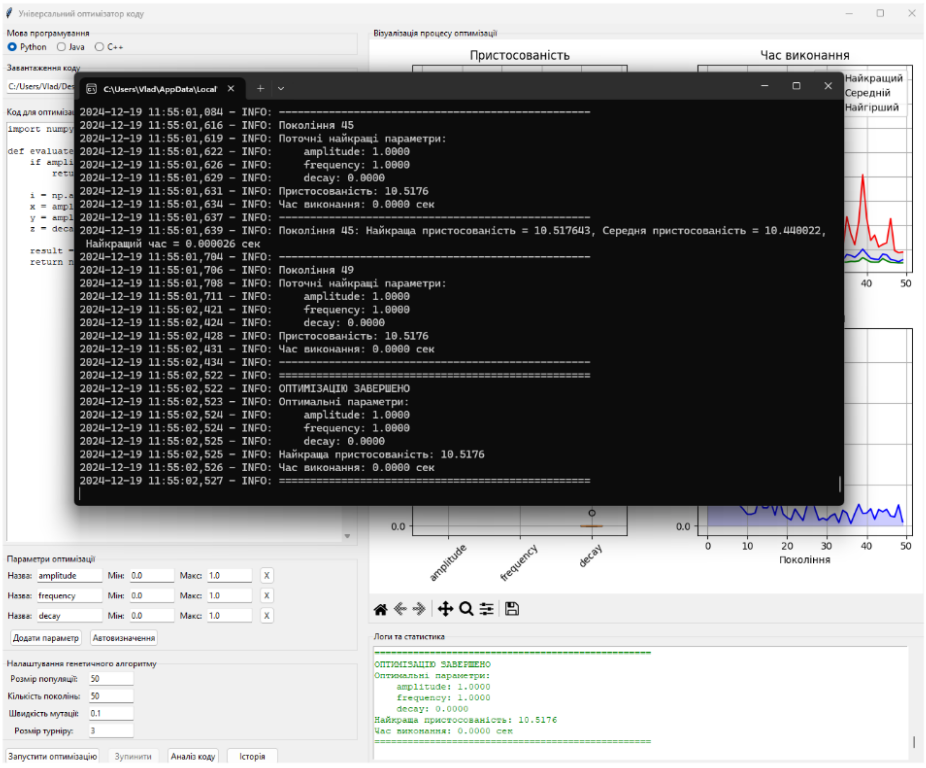
8

Інтерфейс користувача (початковий екран)



9

Запуск процесу оптимізації



10

Результати по завершенню оптимізації

```
2024-11-13 03:05:48,029 - INFO: ================================================
2024-11-13 03:05:48,029 - INFO: ОПТИМІЗАЦІЮ ЗАВЕРШЕНО
2024-11-13 03:05:48,029 - INFO: Оптимальні параметри:
2024-11-13 03:05:48,029 - INFO:     amplitude: 1.0000
2024-11-13 03:05:48,029 - INFO:     frequency: 1.0000
2024-11-13 03:05:48,029 - INFO:     decay: 0.0000
2024-11-13 03:05:48,029 - INFO: Найкраща пристосованість: 10.5176
2024-11-13 03:05:48,029 - INFO: Час виконання: 0.0001 сек
2024-11-13 03:05:48,044 - INFO: ================================================
```

11

ВИСНОВКИ

1. Досліджено основи генетичних алгоритмів, які є одними з найбільш ефективних еволюційних методів оптимізації.
2. Для тестування було обрано датасет, що містить різні фрагменти програмного коду, які потребують оптимізації. Датасет був обґрунтований на основі критеріїв різноманітності задач та реалістичності тестування, що дозволяє оцінити ефективність методу у різних умовах.
3. Розроблений метод був реалізований у вигляді програмного коду, який дозволяє проводити оптимізацію параметрів програм з використанням генетичного алгоритму.
4. Проведено порівняння продуктивності оптимізованого коду за допомогою генетичного алгоритму з іншими традиційними методами оптимізації, такими як метод проб і помилок, градієнтний спуск, сітковий пошук та стохастичні методи. Результати показали, що генетичний алгоритм забезпечує більш ефективну оптимізацію з меншими обчислювальними витратами та кращою якістю досягнутих рішень.

12

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

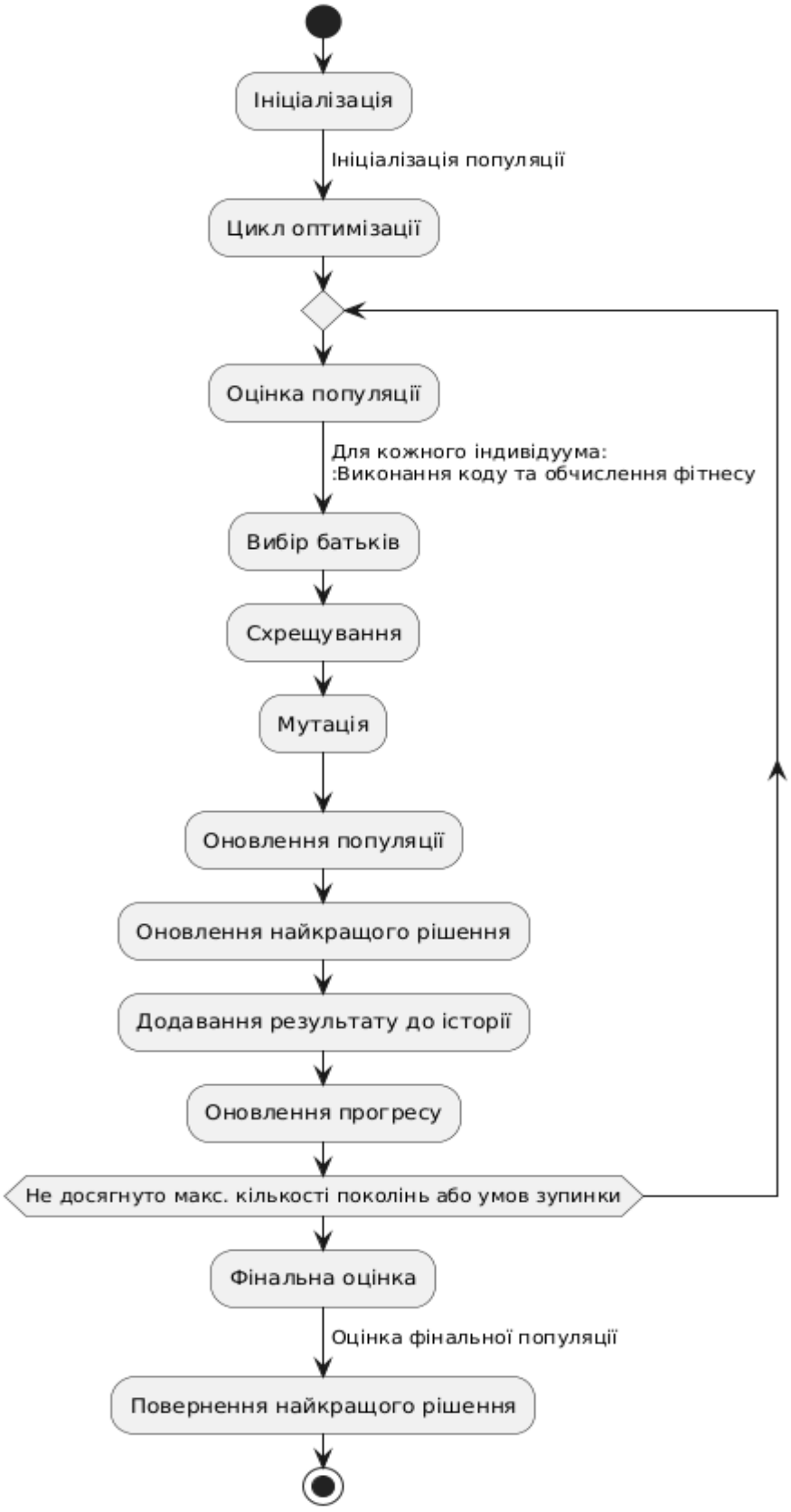
Стаття:

1. Шоробура В.В., Худік Б.О. Розробка методу оптимізації програмного коду на основі генетичного алгоритму // Кібербезпека (Подано до друку).

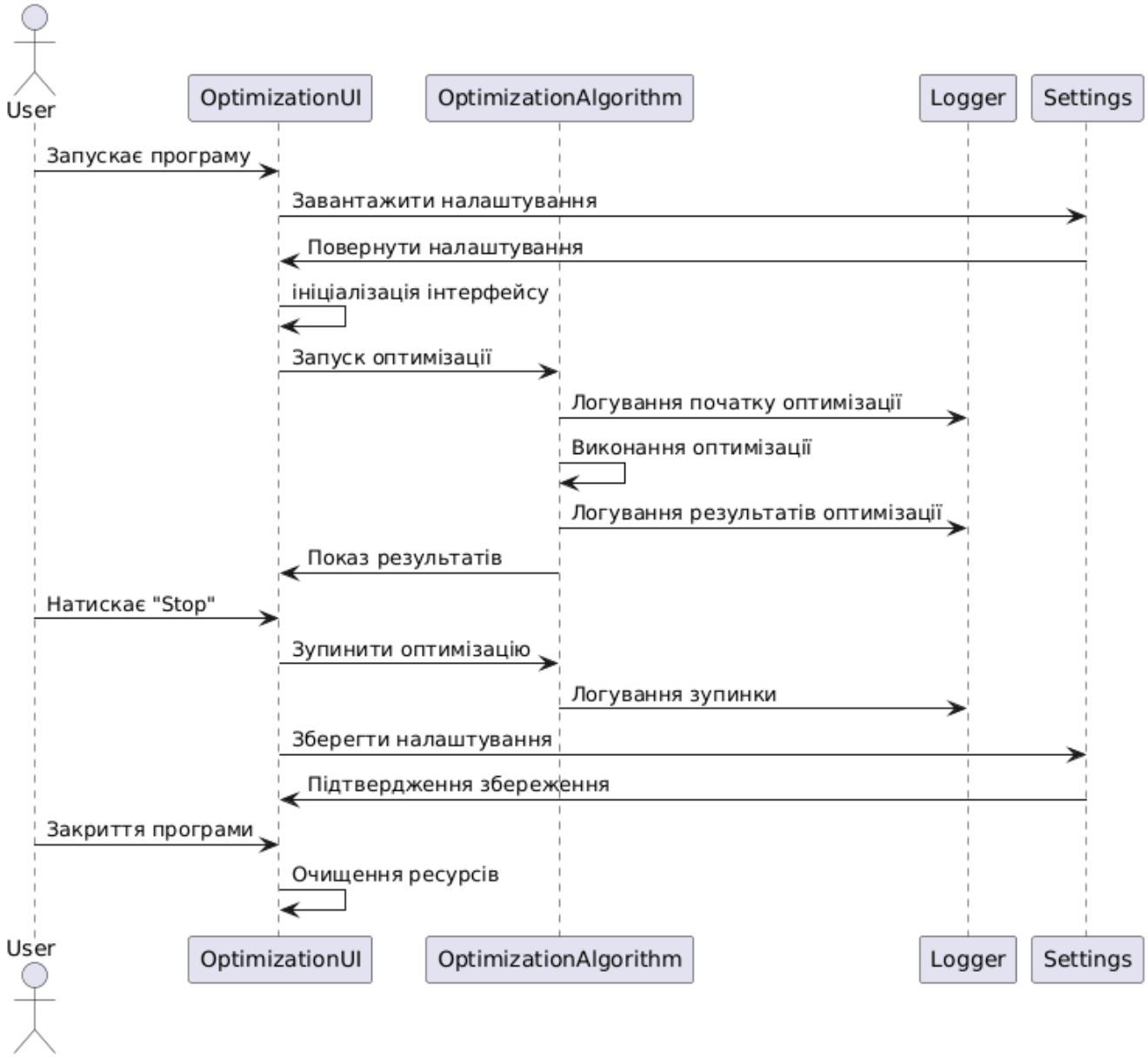
Теза доповіді:

1. Шоробура В.В. Розробка методу оптимізації програмного коду на основі генетичного алгоритму // Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії».– Київ: ДУІКТ, 2022. - С.85.

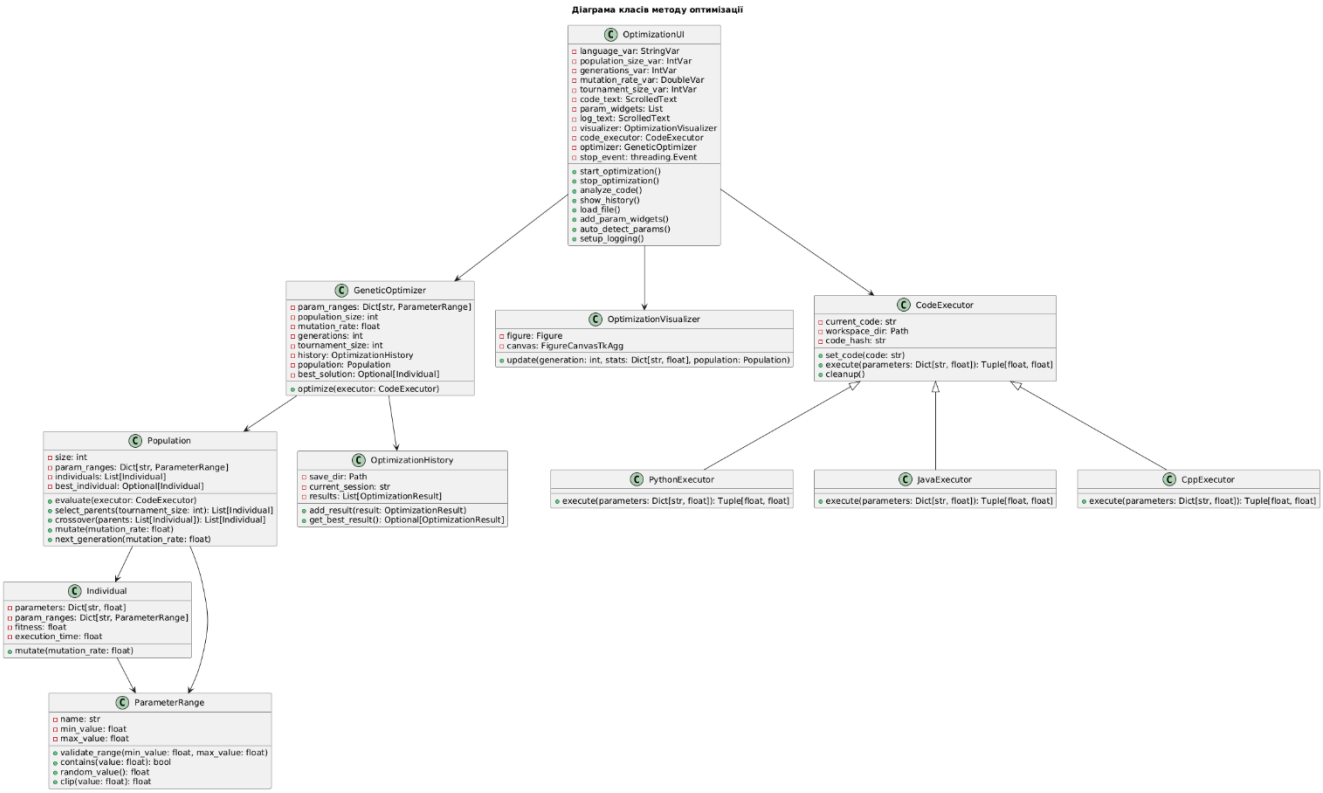
ДОДАТОК Б. АЛГОРИТМ ГЕНЕТИЧНОЇ ОПТИМІЗАЦІЇ



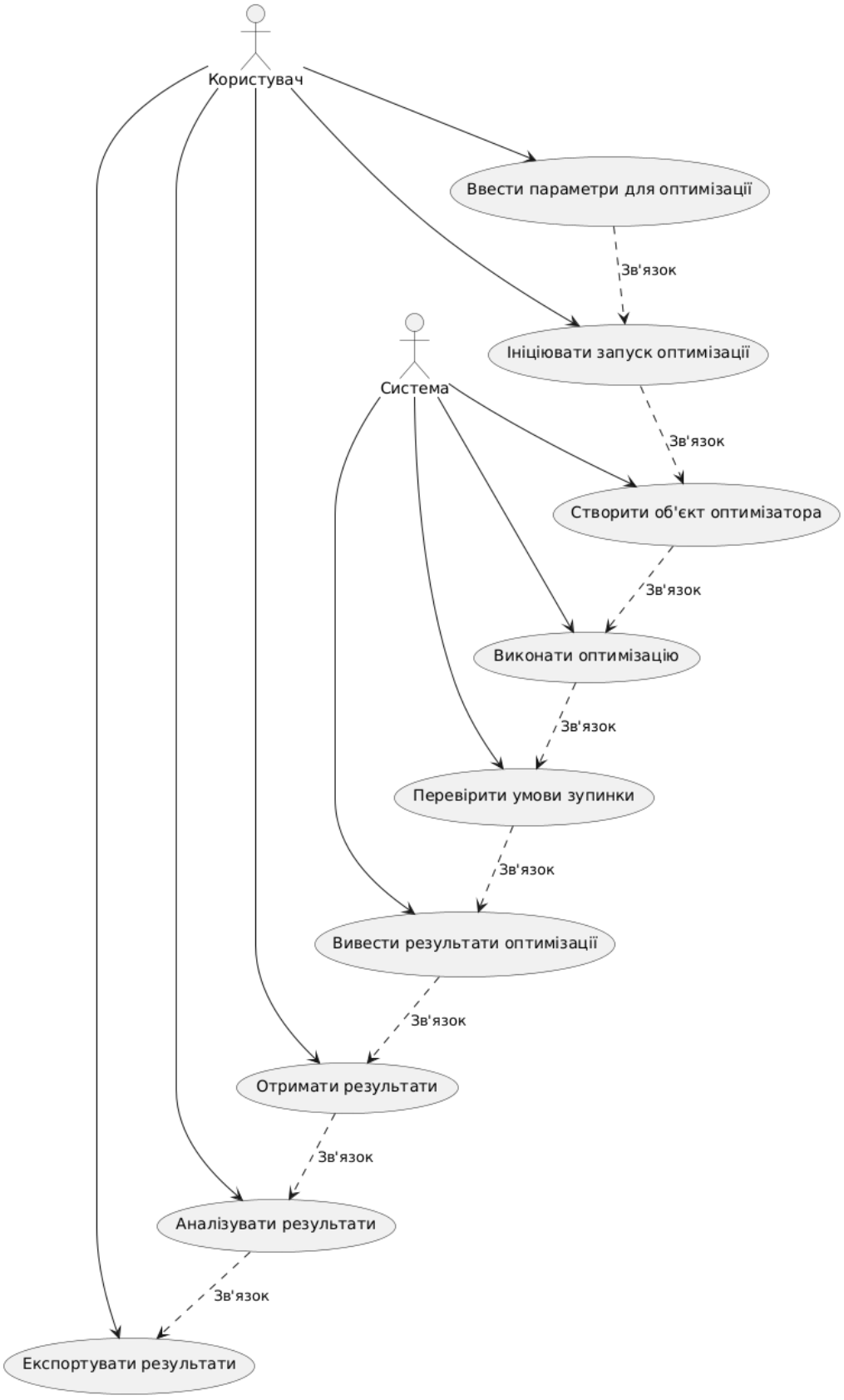
ДОДАТОК В. ДІАГРАМА ПОСЛІДОВНОСТЕЙ



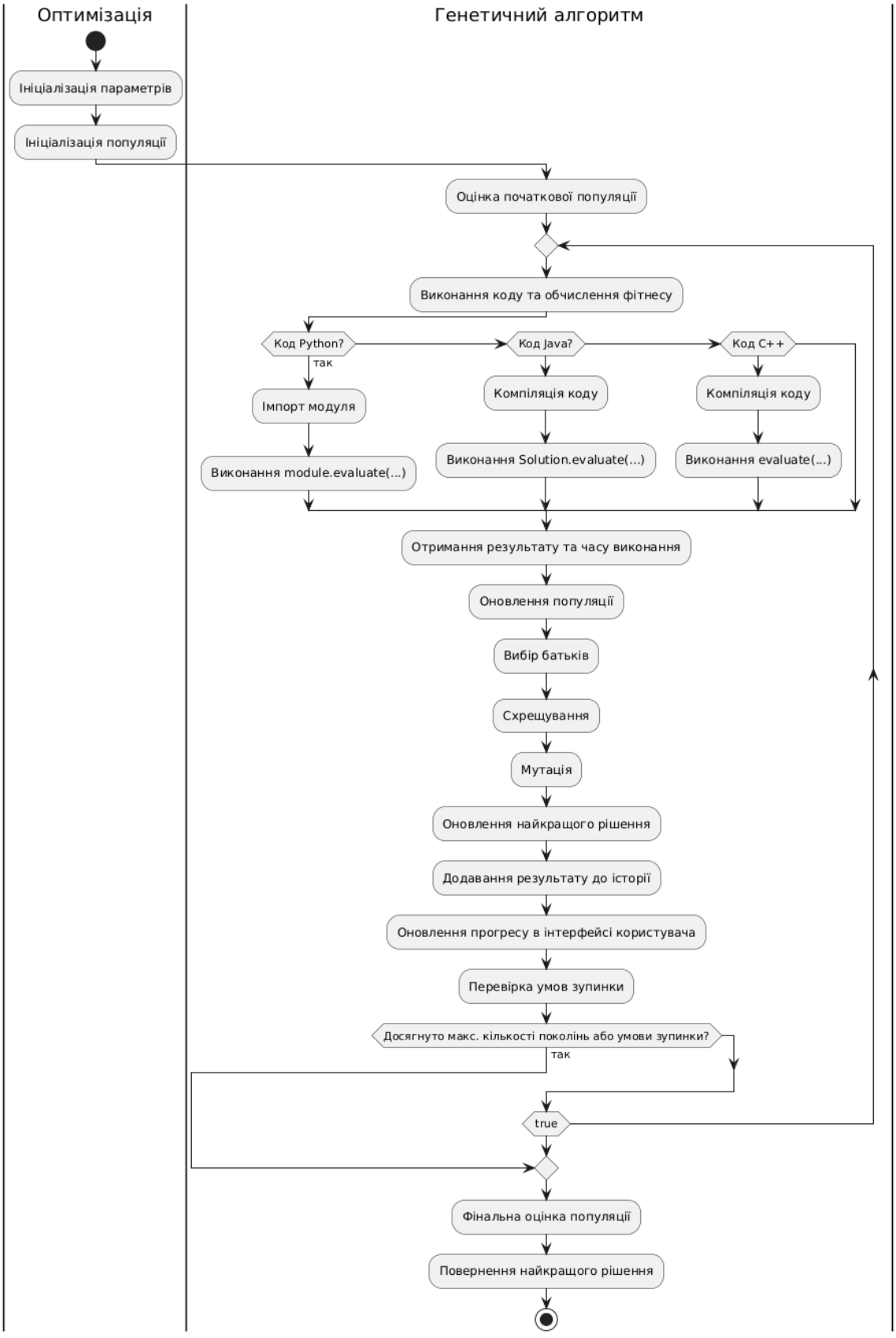
ДОДАТОК Г. ДІАГРАМА ПОСЛІДНОВНОСТЕЙ



ДОДАТОК Ґ. ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



ДОДАТОК Д. ДІАГРАМА ДІЯЛЬНОСТІ



ДОДАТОК Е. ТЕСТУВАННЯ ПРОГРАМИ

Запуск програми – Початковий екран (рис. Г.1)

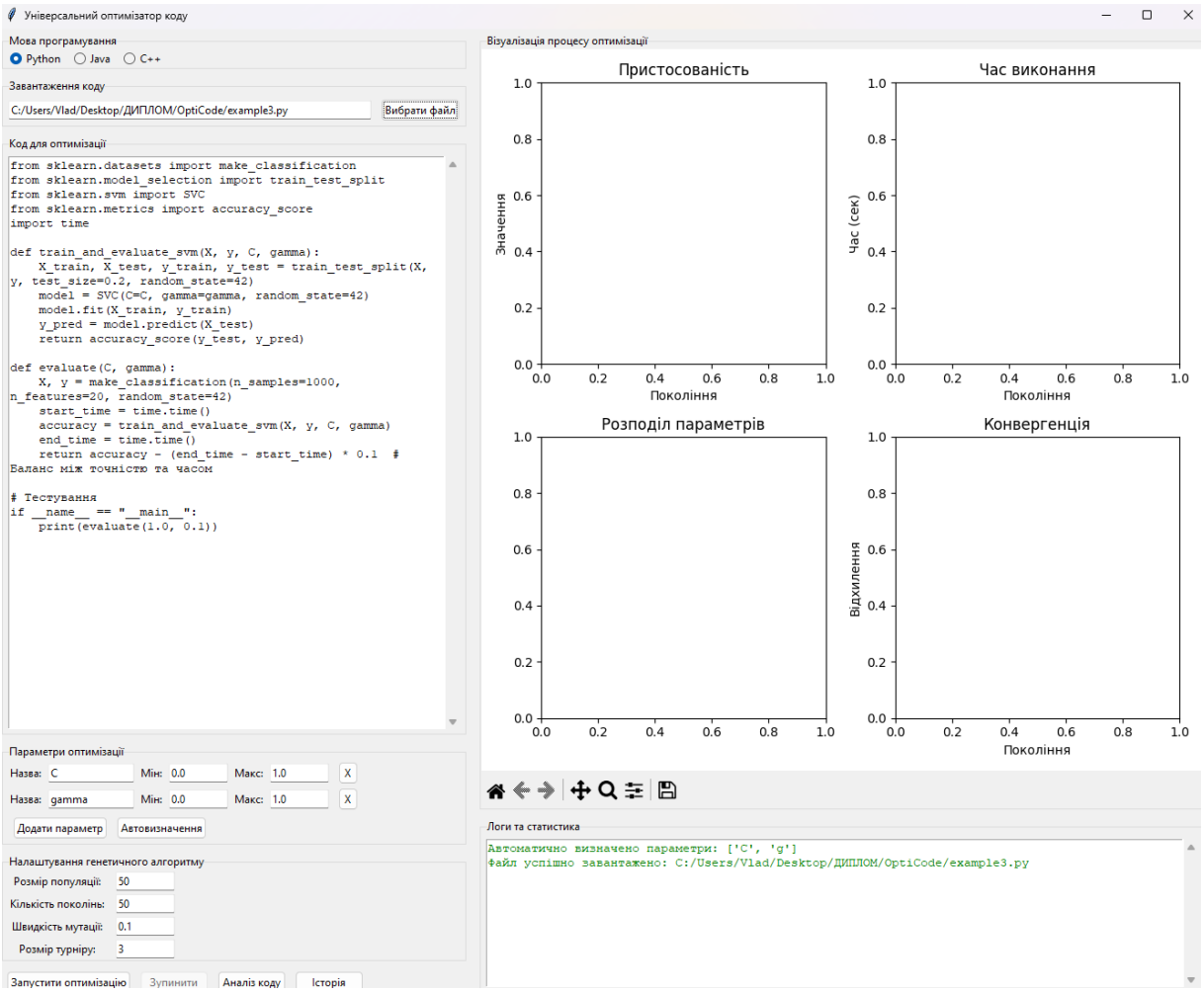


Рис. Г.1 Початкове вікно (інтерфейс користувача)

Вибір мови програмування для універсально оптимізатору коду (рис. Г.2)

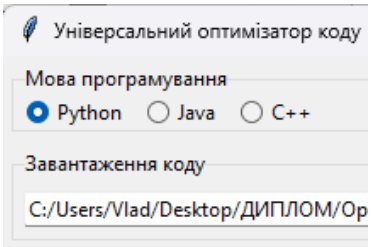


Рис. Г.2 Вибір мови програмування

Запуск методу - процес оптимізації (рис. Г.3)

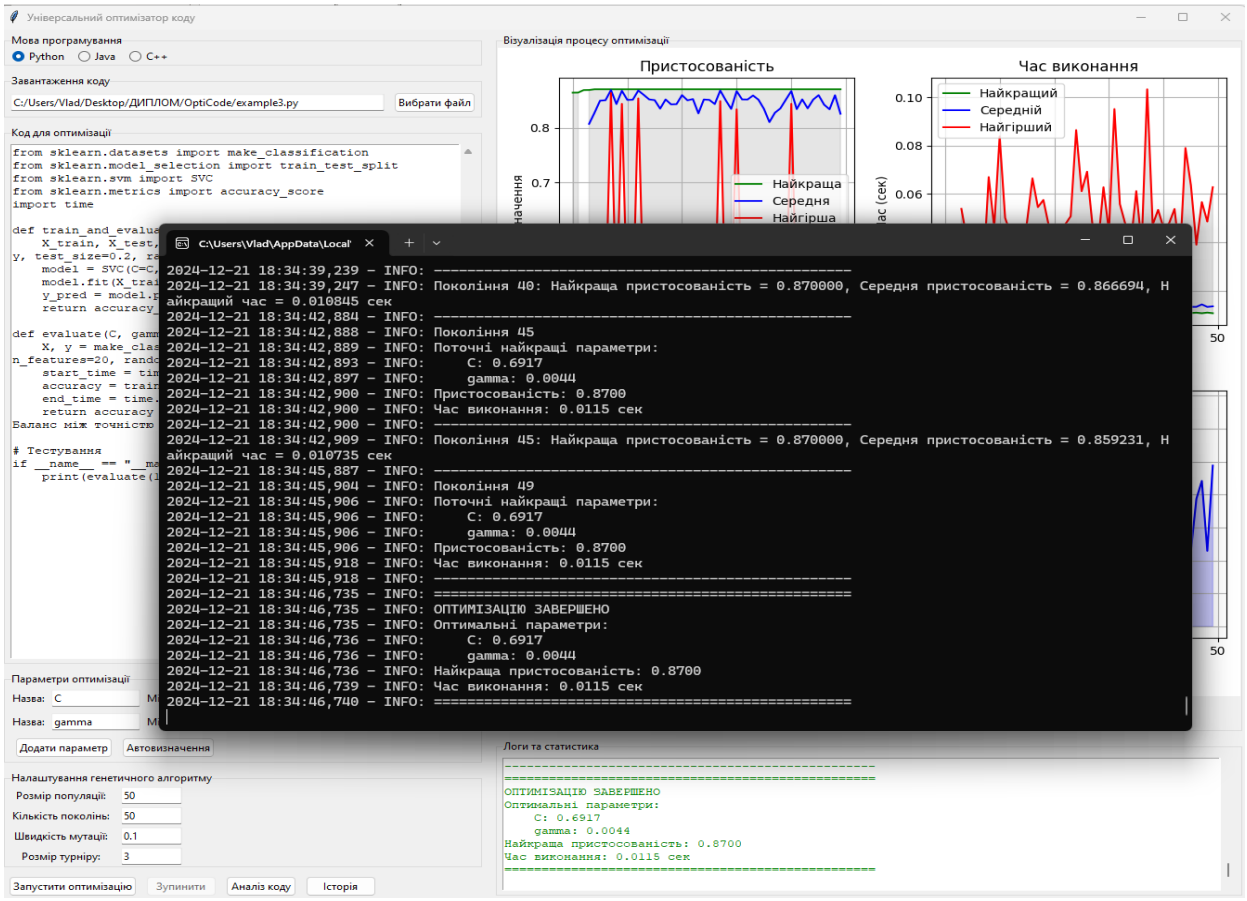


Рис. Г.3 Процес оптимізації

Результати процесу оптимізації (рис. Г.4)

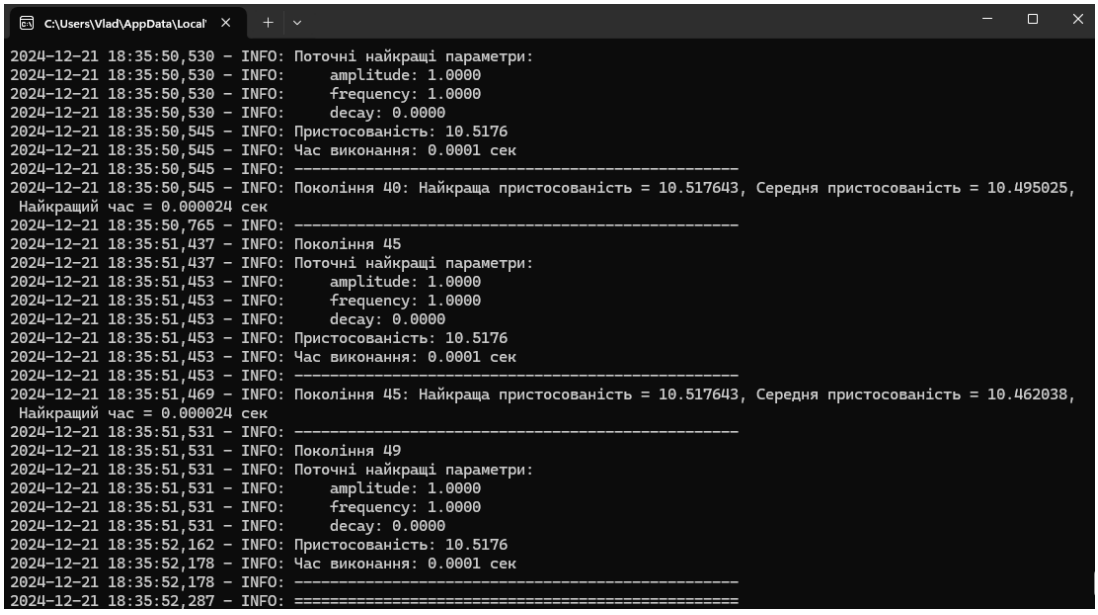


Рис. Г.4 Проміжні результати оптимізації

Візуалізація процесу оптимізації (рис. Г.5, рис. Г.6, рис. Г.8)

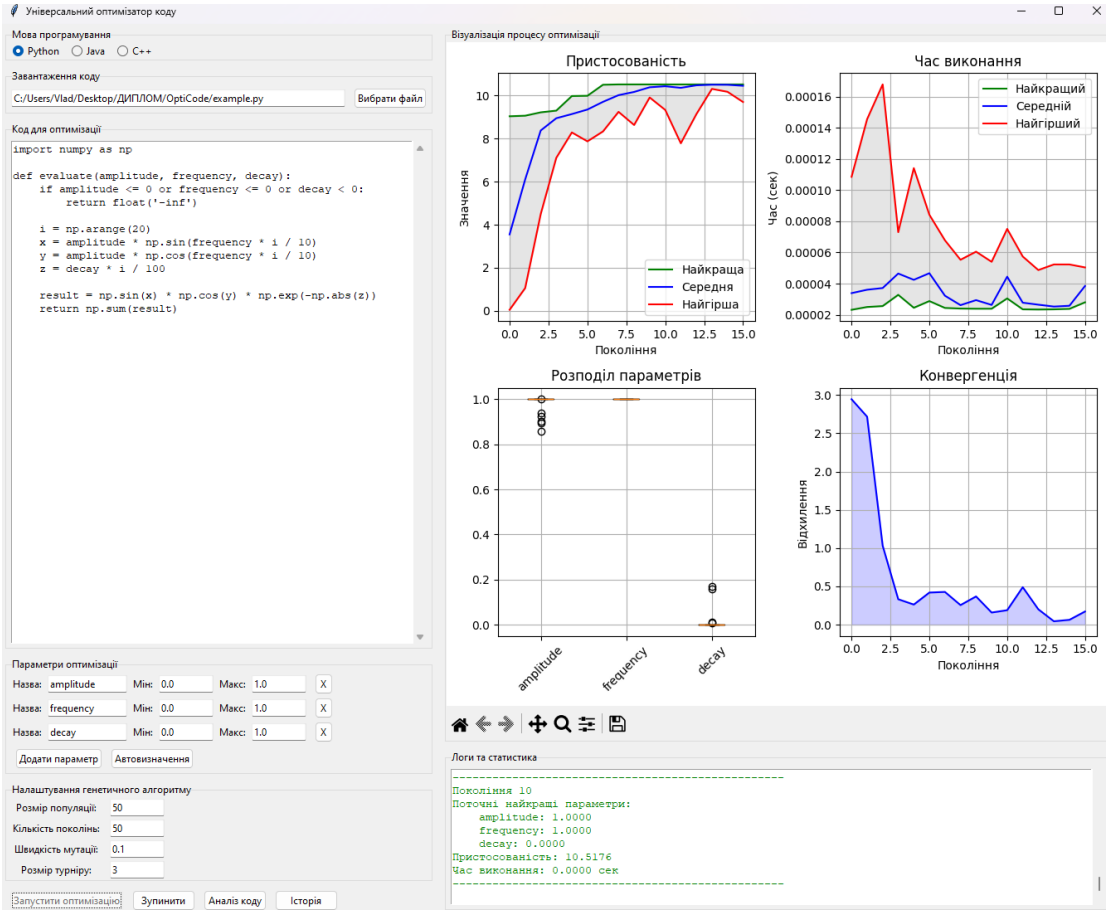


Рис. Г.5 Візуалізація початку процесу оптимізації

По завершенні оптимізації виводяться результати (рис. Г.8)

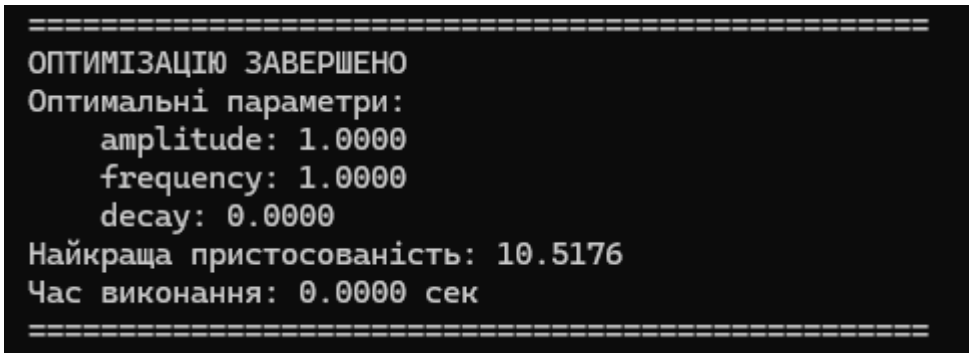


Рис. Г.8 Результати по завершенню оптимізації

По завершенню оптимізації виводяться результати аналізу коду (рис. Г.9)

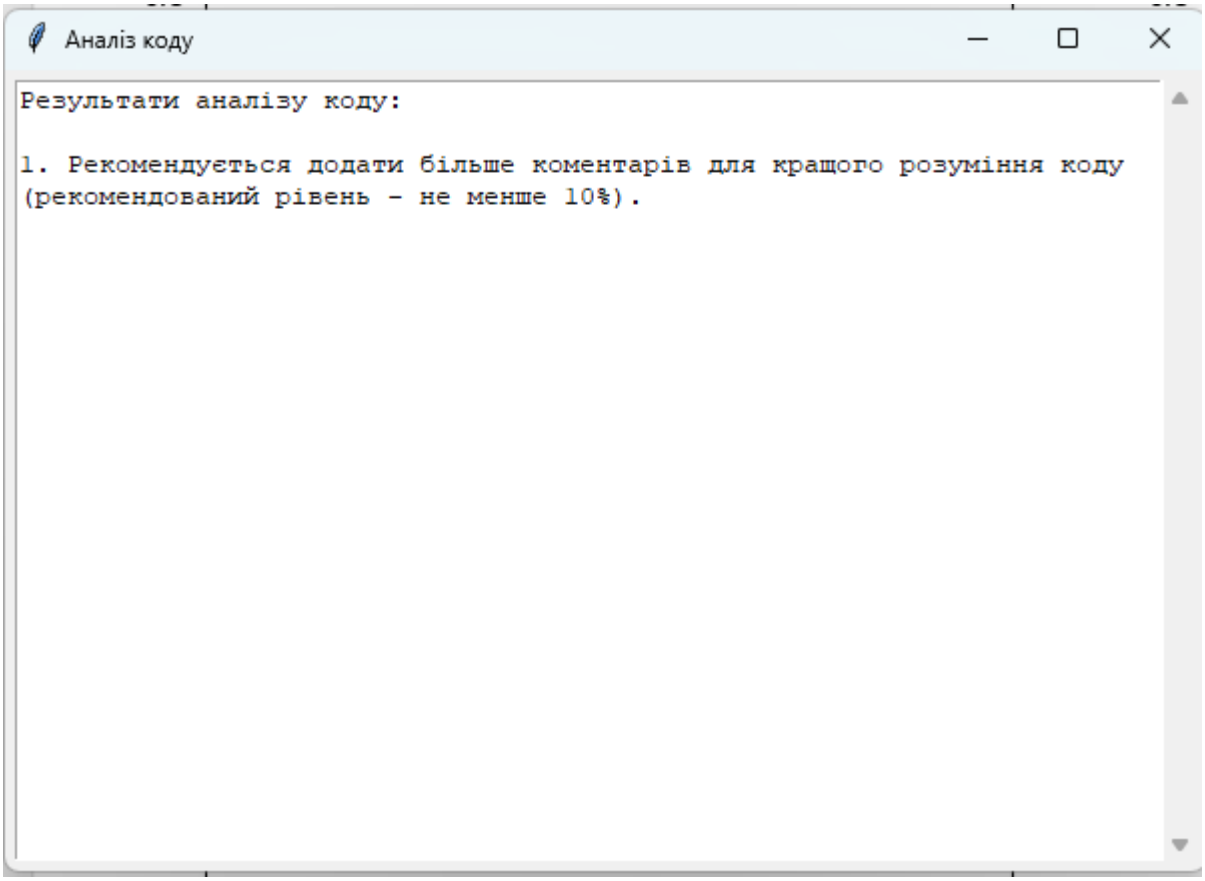


Рис. Г.10 Вивід результатів аналізу коду

ДОДАТОК Є. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
import os
import sys
import io
import time
import json
import queue
import shutil
import pickle
import logging
import tempfile
import threading
import subprocess
import importlib.util
import hashlib
from datetime import datetime
from pathlib import Path
from abc import ABC, abstractmethod
from typing import Dict, List, Tuple, Any,
Optional, Union
from dataclasses import dataclass
# Графічний інтерфейс
import tkinter as tk
from tkinter import ttk, scrolledtext,
filedialog, messagebox
# Наукові бібліотеки
import numpy as np
import pandas as pd
# Візуалізація
import matplotlib
matplotlib.use('TkAgg')  # Важливо встановити
це перед імпортом Figure
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import
FigureCanvasTkAgg, NavigationToolbar2Tk
# Обробка зображень
import cv2
from PIL import Image, ImageTk
logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s - %(levelname)s:
%(message)s',
    handlers=[
        logging.StreamHandler(),
        logging.FileHandler('optimizer.log',
encoding='utf-8')
    ]
)
# Конфігурація логування
class ColoredFormatter(logging.Formatter):
    """Форматер логів з кольоровим виводом"""

    COLORS = {
        'DEBUG': '\033[0;36m',  # Cyan
        'INFO': '\033[0;32m',   # Green
        'WARNING': '\033[0;33m', # Yellow
        'ERROR': '\033[0;31m',   # Red
        'CRITICAL': '\033[0;37;41m',  # White
on Red
        'RESET': '\033[0m'      # Reset
    }
    def format(self, record):
        # Зберігаємо оригінальний формат
        formatted = super().format(record)
        # Додаємо колір
        return
(f"{self.COLORS.get(record.levelname,
self.COLORS['RESET'])}"

f"{formatted}{self.COLORS['RESET']}")
class OptimizationResult:
    """Клас для зберігання результатів
оптимізації"""
    def __init__(self,
                 parameters: Dict[str, float],
                 fitness: float,
                 execution_time: float,
                 generation: int,
                 timestamp: datetime,
                 code_hash: str):
        self.parameters = parameters
        self.fitness = fitness
        self.execution_time = execution_time
        self.generation = generation
        self.timestamp = timestamp
        self.code_hash = code_hash
    def to_dict(self) -> Dict:
        """Конвертація результату в словник
для збереження"""
        return {
            'parameters': self.parameters,
            'fitness': self.fitness,
            'execution_time':
self.execution_time,
            'generation': self.generation,
            'timestamp':
self.timestamp.isoformat(),
            'code_hash': self.code_hash
        }
@classmethod
    def from_dict(cls, data: Dict) ->
'OptimizationResult':
        """Створення об'єкту з словника"""
        return cls(
            parameters=data['parameters'],
            fitness=data['fitness'],

execution_time=data['execution_time'],
            generation=data['generation'],

timestamp=datetime.fromisoformat(data['timesta
mp']),
            code_hash=data['code_hash']
        )
class LogTextHandler(logging.Handler):
    def __init__(self, text_widget:
scrolledtext.ScrolledText):
        super().__init__()
        self.text_widget = text_widget
    def emit(self, record):
        msg = self.format(record)

self.text_widget.configure(state='normal')
        # Додаємо відповідний колір в
залежності від рівня логування
        if record.levelno >= logging.ERROR:

self.text_widget.tag_config('error',
foreground='red')
            tag = 'error'
```

```
        elif record.levelno >=
logging.WARNING:

self.text_widget.tag_config('warning',
foreground='orange')
            tag = 'warning'
        elif record.levelno >= logging.INFO:

self.text_widget.tag_config('info',
foreground='green')
            tag = 'info'
        else:
            tag = None
        # Вставляємо повідомлення
        self.text_widget.insert('end', msg +
'\n', tag)
        self.text_widget.see('end')

self.text_widget.configure(state='disabled')
class CodeExecutor(ABC):
    """Абстрактний базовий клас для виконання
коду на різних мовах"""
    def __init__(self, workspace_dir: Path,
logger: logging.Logger):
        self.workspace_dir = workspace_dir
        self.logger = logger
        self.current_code: Optional[str] =
None
        self.code_hash: Optional[str] = None
    def set_code(self, code: str) -> None:
        """Встановлення коду для виконання"""
        self.current_code = code
        self.code_hash =
hashlib.sha256(code.encode()).hexdigest()
    @abstractmethod
    def prepare_environment(self) -> None:
        """Підготовка середовища виконання"""
        pass
    @abstractmethod
    def execute(self, parameters: Dict[str,
float]) -> Tuple[float, float]:
        """Виконання коду з заданими
параметрами"""
        pass
    @abstractmethod
    def cleanup(self) -> None:
        """Очищення тимчасових файлів"""
        pass
    def __enter__(self):
        self.prepare_environment()
        return self
    def __exit__(self, exc_type, exc_val,
exc_tb):
        self.cleanup()
class PythonExecutor(CodeExecutor):
    """Виконавець Python коду"""
    def prepare_environment(self) -> None:
        if not self.current_code:
            raise ValueError("Код не
встановлено")
        # Створюємо унікальну директорію для
цього коду
        code_dir = self.workspace_dir /
self.code_hash
        code_dir.mkdir(parents=True,
exist_ok=True)
        # Зберігаємо код у файл
        code_file = code_dir / "code.py"
        with code_file.open('w',
encoding='utf-8') as f:
            f.write(self.current_code)
        self.code_file = code_file
        self.module_name =
f"user_code_{self.code_hash}"
    def execute(self, parameters: Dict[str,
float]) -> Tuple[float, float]:
        try:
            # Імпортуємо модуль заново кожного
разу
            if self.module_name in
sys.modules:
                del
sys.modules[self.module_name]
            spec =
importlib.util.spec_from_file_location(
                self.module_name,
str(self.code_file))
            module =
importlib.util.module_from_spec(spec)
            spec.loader.exec_module(module)
            # Вимірюємо час виконання
            start_time = time.perf_counter()
            result =
module.evaluate(**parameters)
            execution_time =
time.perf_counter() - start_time
            return float(result),
execution_time
        except Exception as e:
            self.logger.error(f"Помилка
виконання Python коду: {str(e)}")
            return float('-inf'), float('inf')
    def cleanup(self) -> None:
        try:
            if hasattr(self, 'code_file'):

shutil.rmtree(self.code_file.parent)
        except Exception as e:
            self.logger.error(f"Помилка при
очищенні Python середовища: {str(e)}")
class JavaExecutor(CodeExecutor):
    """Виконавець Java коду"""
    def prepare_environment(self) -> None:
        if not self.current_code:
            raise ValueError("Код не
встановлено")
        code_dir = self.workspace_dir /
self.code_hash
        code_dir.mkdir(parents=True,
exist_ok=True)
        # Створюємо Java клас з методом
evaluate
        java_code =
self._wrap_java_code(self.current_code)
        # Зберігаємо код у файл
        code_file = code_dir / "Solution.java"
        with code_file.open('w',
encoding='utf-8') as f:
            f.write(java_code)
        self.code_file = code_file
        self.code_dir = code_dir
        # Компілюємо код
        try:
            subprocess.run(
                ["javac", str(code_file)],
                check=True,
                capture_output=True,
```

```
                text=True
            )
        except subprocess.CalledProcessError
as e:
            raise RuntimeError(f"Помилка
компіляції Java коду: {e.stderr}")
    def execute(self, parameters: Dict[str,
float]) -> Tuple[float, float]:
        try:
            # Формуємо параметри командного
рядка
            args = [str(val) for val in
parameters.values()]
            # Запускаємо Java програму
            start_time = time.perf_counter()
            result = subprocess.run(
                ["java", "-cp",
str(self.code_dir), "Solution"] + args,
                check=True,
                capture_output=True,
                text=True
            )
            execution_time =
time.perf_counter() - start_time
            return float(result.stdout),
execution_time
        except (subprocess.CalledProcessError,
ValueError) as e:
            self.logger.error(f"Помилка
виконання Java коду: {str(e)}")
            return float('-inf'), float('inf')
    def cleanup(self) -> None:
        try:
            if hasattr(self, 'code_dir'):
                shutil.rmtree(self.code_dir)
        except Exception as e:
            self.logger.error(f"Помилка при
очищенні Java середовища: {str(e)}")
    def _wrap_java_code(self, code: str) ->
str:
        """Обгортання Java коду в клас з
методом evaluate"""
        return f"""
public class Solution {{
    {code}
    public static void main(String[] args) {{
        try {{
            // Перетворення аргументів в
double
            double[] params = new
double[args.length];
            for (int i = 0; i < args.length;
i++) {{
                params[i] =
Double.parseDouble(args[i]);
            }}
            // Виклик методу evaluate з
параметрами
            double result = evaluate(params);
            System.out.println(result);
        }} catch (Exception e) {{
            System.err.println("Error: " +
e.getMessage());

System.out.println(Double.NEGATIVE_INFINITY);
        }}
    }}
}}
"""

class CppExecutor(CodeExecutor):
    """Виконавець C++ коду"""
    def prepare_environment(self) -> None:
        if not self.current_code:
            raise ValueError("Код не
встановлено")
        code_dir = self.workspace_dir /
self.code_hash
        code_dir.mkdir(parents=True,
exist_ok=True)
        # Створюємо C++ код з функцією
evaluate
        cpp_code =
self._wrap_cpp_code(self.current_code)
        # Зберігаємо код у файл
        code_file = code_dir / "solution.cpp"
        with code_file.open('w',
encoding='utf-8') as f:
            f.write(cpp_code)
        self.code_file = code_file
        self.code_dir = code_dir
        self.exe_file = code_dir / "solution"
        # Компілюємо код
        try:
            subprocess.run(
                ["g++", str(code_file), "-o",
str(self.exe_file)],
                check=True,
                capture_output=True,
                text=True
            )
        except subprocess.CalledProcessError
as e:
            raise RuntimeError(f"Помилка
компіляції C++ коду: {e.stderr}")
    def execute(self, parameters: Dict[str,
float]) -> Tuple[float, float]:
        try:
            # Формуємо параметри командного
рядка
            args = [str(val) for val in
parameters.values()]
            # Запускаємо програму
            start_time = time.perf_counter()
            result = subprocess.run(
                [str(self.exe_file)] + args,
                check=True,
                capture_output=True,
                text=True
            )
            execution_time =
time.perf_counter() - start_time
            return float(result.stdout),
execution_time
        except (subprocess.CalledProcessError,
ValueError) as e:
            self.logger.error(f"Помилка
виконання C++ коду: {str(e)}")
            return float('-inf'), float('inf')
    def cleanup(self) -> None:
        try:
            if hasattr(self, 'code_dir'):
                shutil.rmtree(self.code_dir)
        except Exception as e:
            self.logger.error(f"Помилка при
очищенні C++ середовища: {str(e)}")
    def _wrap_cpp_code(self, code: str) ->
str:
```

```
        """Обгортання C++ коду в програму з
функцією evaluate"""
        return f"""
#include <iostream>
#include <vector>
#include <stdexcept>
#include <limits>
{code}
int main(int argc, char* argv[]) {{
    try {{
        // Перетворення аргументів в double
        std::vector<double> params;
        for (int i = 1; i < argc; i++) {{

params.push_back(std::stod(argv[i]));
        }}
        // Виклик функції evaluate з
параметрами
        double result =
evaluate(params.data());
        std::cout << result << std::endl;
    }} catch (const std::exception& e) {{
        std::cerr << "Error: " << e.what() <<
std::endl;
        std::cout <<
std::numeric_limits<double>::lowest() <<
std::endl;
    }}

    return 0;
}}
""" class PythonExecutor(CodeExecutor):
    """Виконавець Python коду"""
    def prepare_environment(self) -> None:
        if not self.current_code:
            raise ValueError("Код не
встановлено")
        # Створюємо унікальну директорію для
цього коду
        code_dir = self.workspace_dir /
self.code_hash
        code_dir.mkdir(parents=True,
exist_ok=True)
        # Зберігаємо код у файл
        code_file = code_dir / "code.py"
        with code_file.open('w',
encoding='utf-8') as f:
            f.write(self.current_code)
        self.code_file = code_file
        self.module_name =
f"user_code_{self.code_hash}"
    def execute(self, parameters: Dict[str,
float]) -> Tuple[float, float]:
        try:
            # Імпортуємо модуль заново кожного
разу
            if self.module_name in
sys.modules:
                del
sys.modules[self.module_name]
            spec =
importlib.util.spec_from_file_location(
                self.module_name,
str(self.code_file))
            module =
importlib.util.module_from_spec(spec)
            spec.loader.exec_module(module)
            # Вимірюємо час виконання
            start_time = time.perf_counter()
            result =
module.evaluate(**parameters)
            execution_time =
time.perf_counter() - start_time
            return float(result),
execution_time
        except Exception as e:
            self.logger.error(f"Помилка
виконання Python коду: {str(e)}")
            return float('-inf'), float('inf')
    def cleanup(self) -> None:
        try:
            if hasattr(self, 'code_file'):

shutil.rmtree(self.code_file.parent)
        except Exception as e:
            self.logger.error(f"Помилка при
очищенні Python середовища: {str(e)}")
class OptimizationHistory:
    """Клас для зберігання та аналізу історії
оптимізації"""
    def __init__(self, save_dir: Path):
        self.save_dir = save_dir
        self.save_dir.mkdir(parents=True,
exist_ok=True)
        self.current_session =
datetime.now().strftime("%Y%m%d_%H%M%S")
        self.results: List[OptimizationResult]
= []
    def add_result(self, result:
OptimizationResult) -> None:
        """Додавання нового результату"""
        self.results.append(result)
        self._save_result(result)
    def _save_result(self, result:
OptimizationResult) -> None:
        """Зберігання результату у файл"""
        session_dir = self.save_dir /
self.current_session
        session_dir.mkdir(exist_ok=True)
        result_file = session_dir /
f"result_{len(self.results)}.json"
        with result_file.open('w',
encoding='utf-8') as f:
            json.dump(result.to_dict(), f,
indent=4)
    def get_best_result(self) ->
Optional[OptimizationResult]:
        """Отримання найкращого результату"""
        if not self.results:
            return None
        return max(self.results, key=lambda x:
x.fitness)
    def to_dataframe(self) -> pd.DataFrame:
        """Конвертація історії в DataFrame для
аналізу"""
        data = []
        for result in self.results:
            row = {
                'timestamp': result.timestamp,
                'fitness': result.fitness,
                'execution_time':
result.execution_time,
                'generation':
result.generation,
                'code_hash': result.code_hash
            }
            row.update(result.parameters)
            data.append(row)
```

```
        return pd.DataFrame(data)
    def load_session(self, session_id: str) ->
None:
        """Завантаження попередньої сесії"""
        session_dir = self.save_dir /
session_id
        if not session_dir.exists():
            raise ValueError(f"Сесія
{session_id} не знайдена")
        self.results.clear()
        for result_file in
sorted(session_dir.glob("result_*.json")):
            with result_file.open('r',
encoding='utf-8') as f:
                data = json.load(f)

self.results.append(OptimizationResult.from_di
ct(data))
    def get_sessions(self) -> List[str]:
        """Отримання списку доступних сесій"""
        return [d.name for d in
self.save_dir.iterdir() if d.is_dir()]
class ParameterRange:
    """Клас для зберігання та валідації
діапазону параметра"""
    def __init__(self, name: str, min_value:
float, max_value: float,
                 default: Optional[float] =
None):
        self.name = name
        self.validate_range(min_value,
max_value)
        self.min_value = min_value
        self.max_value = max_value
        self.default = default or (min_value +
max_value) / 2
    @staticmethod
    def validate_range(min_value: float,
max_value: float) -> None:
        """Перевірка коректності діапазону"""
        if min_value >= max_value:
            raise ValueError("Мінімальне
значення має бути менше максимального")
    def contains(self, value: float) -> bool:
        """Перевірка, чи входить значення в
діапазон"""
        return self.min_value <= value <=
self.max_value
    def random_value(self) -> float:
        """Генерація випадкового значення з
діапазону"""
        return
np.random.uniform(self.min_value,
self.max_value)
    def clip(self, value: float) -> float:
        """Обмеження значення діапазоном"""
        return np.clip(value, self.min_value,
self.max_value)
class Individual:
    """Клас для представлення окремої особини
в популяції"""
    def __init__(self, parameters: Dict[str,
float], param_ranges: Dict[str,
ParameterRange]):
        self.parameters = parameters
        self.param_ranges = param_ranges
        self.fitness: float = float('-inf')
        self.execution_time: float =
float('inf')

    @classmethod
    def random(cls, param_ranges: Dict[str,
ParameterRange]) -> 'Individual':
        """Створення випадкової особини"""
        parameters = {
            name: param_range.random_value()
            for name, param_range in
param_ranges.items()
        }
        return cls(parameters, param_ranges)
    def mutate(self, mutation_rate: float) ->
None:
        """Мутація параметрів особини"""
        for name, value in
self.parameters.items():
            if np.random.random() <
mutation_rate:
                # Гаусова мутація з адаптивним
розміром кроку
                param_range =
self.param_ranges[name]
                range_size =
param_range.max_value - param_range.min_value
                step_size = range_size * 0.1
# 10% від діапазону
                # Генеруємо випадкове зміщення
                delta = np.random.normal(0,
step_size)
                new_value = value + delta
                # Обмежуємо значення
діапазоном
                self.parameters[name] =
param_range.clip(new_value)
    def __str__(self) -> str:
        params_str = ", ".join(f"{k}={v:.3f}"
for k, v in self.parameters.items())
        return f"Individual({params_str},
fitness={self.fitness:.3f},
time={self.execution_time:.3f}s)"
class Population:
    """Клас для управління популяцією
особин"""
    def __init__(self, size: int,
param_ranges: Dict[str, ParameterRange]):
        self.size = size
        self.param_ranges = param_ranges
        self.individuals =
[Individual.random(param_ranges) for _ in
range(size)]
        self.best_individual:
Optional[Individual] = None
        self.generation = 0
    def evaluate(self, executor: CodeExecutor)
-> None:
        """Оцінка всіх особин в популяції"""
        for individual in self.individuals:
            fitness, time =
executor.execute(individual.parameters)
            individual.fitness = fitness
            individual.execution_time = time
        # Оновлення найкращої особини
        current_best = max(self.individuals,
key=lambda x: x.fitness)
        if (not self.best_individual or
            current_best.fitness >
self.best_individual.fitness):
            self.best_individual = Individual(

current_best.parameters.copy(),
```

```
                self.param_ranges
            )
            self.best_individual.fitness =
current_best.fitness

self.best_individual.execution_time =
current_best.execution_time
    def select_parents(self, tournament_size:
int = 3) -> List[Individual]:
        """Вибір батьків методом турнірної
селекції"""
        parents = []
        for _ in range(self.size - 1):  # -1
для елітизму
            # Вибір випадкових особин для
турніру
            tournament = np.random.choice(
                self.individuals,
                size=tournament_size,
                replace=False
            )
            # Вибір переможця турніру
            winner = max(tournament,
key=lambda x: x.fitness)
            parents.append(winner)
        return parents
    def crossover(self, parents:
List[Individual]) -> List[Individual]:
        """Схрещування батьків для створення
нащадків"""
        offspring = []
        np.random.shuffle(parents)
        # Додаємо найкращу особину (елітизм)
        best = max(self.individuals,
key=lambda x: x.fitness)

offspring.append(Individual(best.parameters.co
py(), self.param_ranges))
        # Створюємо нащадків парами батьків
        for i in range(0, len(parents) - 1,
2):
            parent1, parent2 = parents[i],
parents[i + 1]
            # SBX (Simulated Binary Crossover)
            child1_params = {}
            child2_params = {}
            for param_name in
self.param_ranges:
                param_range =
self.param_ranges[param_name]
                x1 =
parent1.parameters[param_name]
                x2 =
parent2.parameters[param_name]
                # Параметр розподілу для SBX
                eta = 20
                # Генерація випадкового числа
                u = np.random.random()
                # Розрахунок коефіцієнта бета
                if u <= 0.5:
                    beta = (2 * u) ** (1 /
(eta + 1))
                else:
                    beta = (1 / (2 * (1 - u)))
** (1 / (eta + 1))
                # Створення нащадків
                child1_value = 0.5 * ((1 +
beta) * x1 + (1 - beta) * x2)
                child2_value = 0.5 * ((1 -
beta) * x1 + (1 + beta) * x2)
                # Обмеження значень діапазоном
                child1_params[param_name] =
param_range.clip(child1_value)
                child2_params[param_name] =
param_range.clip(child2_value)

offspring.append(Individual(child1_params,
self.param_ranges))
            if len(offspring) < self.size:

offspring.append(Individual(child2_params,
self.param_ranges))
        return offspring
    def mutate(self, mutation_rate: float) ->
None:
        """Мутація всіх особин в популяції,
крім найкращої"""
        for individual in
self.individuals[1:]:  # Пропускаємо найкращу
особину
            individual.mutate(mutation_rate)
    def next_generation(self, mutation_rate:
float) -> None:
        """Створення наступного покоління"""
        parents = self.select_parents()
        offspring = self.crossover(parents)
        self.individuals = offspring
        self.mutate(mutation_rate)
        self.generation += 1
    def get_statistics(self) -> Dict[str,
float]:
        """Отримання статистики популяції"""
        fitnesses = [ind.fitness for ind in
self.individuals]
        times = [ind.execution_time for ind in
self.individuals]
        return {
            'best_fitness': max(fitnesses),
            'avg_fitness': np.mean(fitnesses),
            'worst_fitness': min(fitnesses),
            'fitness_std': np.std(fitnesses),
            'best_time': min(times),
            'avg_time': np.mean(times),
            'worst_time': max(times),
            'time_std': np.std(times)
        }
class GeneticOptimizer:
    """Основний клас генетичного алгоритму"""
    def __init__(self,
                 param_ranges: Dict[str,
ParameterRange],
                 population_size: int = 50,
                 mutation_rate: float = 0.1,
                 generations: int = 50,
                 tournament_size: int = 3,
                 history:
Optional[OptimizationHistory] = None):
        self.param_ranges = param_ranges
        self.population_size = population_size
        self.mutation_rate = mutation_rate
        self.generations = generations
        self.tournament_size = tournament_size
        self.history = history or
OptimizationHistory(Path("optimization_history
"))
        self.population =
Population(population_size, param_ranges)
```

```
        self.best_solution:
Optional[Individual] = None
        # Налаштування логування
        self.logger =
logging.getLogger('GeneticOptimizer')
    def optimize(self, executor: CodeExecutor,
                 progress_callback:
Optional[callable] = None,
                 stop_event:
Optional[threading.Event] = None) ->
OptimizationResult:
        """Запуск процесу оптимізації"""
        self.logger.info("Початок
оптимізації")
        self.logger.info(f"Параметри:
{list(self.param_ranges.keys())}")
        self.logger.info(f"Розмір популяції:
{self.population_size}")
        self.logger.info(f"Кількість поколінь:
{self.generations}")
        self.logger.info(f"Швидкість мутації:
{self.mutation_rate}")
        try:
            for generation in
range(self.generations):
                if stop_event and
stop_event.is_set():

self.logger.info("Оптимізацію перервано
користувачем")
                    break
                # Оцінка популяції

self.population.evaluate(executor)
                # Отримання статистики
                stats =
self.population.get_statistics()
                # Оновлення найкращого рішення
                if (not self.best_solution or
                    stats['best_fitness'] >
self.best_solution.fitness):
                    best_ind =
max(self.population.individuals,
                                key=lambda x:
x.fitness)
                    self.best_solution =
Individual(

best_ind.parameters.copy(),
                        self.param_ranges
                    )
                    self.best_solution.fitness
= best_ind.fitness

self.best_solution.execution_time =
best_ind.execution_time
                # Збереження результату
                result = OptimizationResult(

parameters=self.best_solution.parameters,

fitness=self.best_solution.fitness,

execution_time=self.best_solution.execution_ti
me,
                    generation=generation,
                    timestamp=datetime.now(),

code_hash=executor.code_hash
                )

self.history.add_result(result)
                # Виклик callback для
оновлення інтерфейсу
                if progress_callback:

progress_callback(generation, stats)
                # Логування прогресу
                if generation % 5 == 0:
                    self.logger.info(
                        f"Покоління
{generation}: "
                        f"Найкраща
пристосованість = {stats['best_fitness']:.6f},
"
                        f"Середня
пристосованість = {stats['avg_fitness']:.6f},
"
                        f"Найкращий час =
{stats['best_time']:.6f} сек"
                    )
                # Створення наступного
покоління

self.population.next_generation(self.mutation_
rate)
            # Фінальне оцінювання
            self.population.evaluate(executor)
            return OptimizationResult(

parameters=self.best_solution.parameters,

fitness=self.best_solution.fitness,

execution_time=self.best_solution.execution_ti
me,
                generation=self.generations -
1,
                timestamp=datetime.now(),
                code_hash=executor.code_hash
            )
        except Exception as e:
            self.logger.error(f"Помилка під
час оптимізації: {str(e)}")
            raise
class OptimizationVisualizer:
    """Клас для візуалізації процесу
оптимізації"""
    def __init__(self, figure: Figure, canvas:
FigureCanvasTkAgg):
        self.figure = figure
        self.canvas = canvas
        self.generation_data = []
        self.setup_plots()
    def setup_plots(self) -> None:
        """Налаштування графіків"""
        self.figure.clear()
        # Графік пристосованості
        self.ax_fitness =
self.figure.add_subplot(221)

self.ax_fitness.set_title('Пристосованість')

self.ax_fitness.set_xlabel('Покоління')
        self.ax_fitness.set_ylabel('Значення')
        # Графік часу виконання
        self.ax_time =
self.figure.add_subplot(222)
```

```
        self.ax_time.set_title('Час
виконання')
        self.ax_time.set_xlabel('Покоління')
        self.ax_time.set_ylabel('Час (сек)')
        # Графік розподілу параметрів
        self.ax_params =
self.figure.add_subplot(223)
        self.ax_params.set_title('Розподіл
параметрів')
        # Графік конвергенції
        self.ax_convergence =
self.figure.add_subplot(224)

self.ax_convergence.set_title('Конвергенція')

self.ax_convergence.set_xlabel('Покоління')

self.ax_convergence.set_ylabel('Відхилення')
        self.figure.tight_layout()
    def update(self, generation: int, stats:
Dict[str, float],
               population: Population) ->
None:
        """Оновлення всіх графіків"""
        self.generation_data.append({
            'generation': generation,
            'stats': stats,
            'population': [ind.parameters for
ind in population.individuals]
        })

        self._update_fitness_plot()
        self._update_time_plot()
        self._update_params_plot()
        self._update_convergence_plot()
        self.figure.tight_layout()
        self.canvas.draw()
    def _update_fitness_plot(self) -> None:
        """Оновлення графіку
пристосованості"""
        self.ax_fitness.clear()

self.ax_fitness.set_title('Пристосованість')

self.ax_fitness.set_xlabel('Покоління')
        self.ax_fitness.set_ylabel('Значення')
        generations = [d['generation'] for d
in self.generation_data]
        best_fitness =
[d['stats']['best_fitness'] for d in
self.generation_data]
        avg_fitness =
[d['stats']['avg_fitness'] for d in
self.generation_data]
        worst_fitness =
[d['stats']['worst_fitness'] for d in
self.generation_data]
        self.ax_fitness.plot(generations,
best_fitness, 'g-', label='Найкраща')
        self.ax_fitness.plot(generations,
avg_fitness, 'b-', label='Середня')
        self.ax_fitness.plot(generations,
worst_fitness, 'r-', label='Найгірша')

self.ax_fitness.fill_between(generations,
worst_fitness, best_fitness,
                             alpha=0.2,
color='gray')
        self.ax_fitness.legend()
        self.ax_fitness.grid(True)
    def _update_time_plot(self) -> None:
        """Оновлення графіку часу виконання"""
        self.ax_time.clear()
        self.ax_time.set_title('Час
виконання')
        self.ax_time.set_xlabel('Покоління')
        self.ax_time.set_ylabel('Час (сек)')
        generations = [d['generation'] for d
in self.generation_data]
        best_time = [d['stats']['best_time']
for d in self.generation_data]
        avg_time = [d['stats']['avg_time'] for
d in self.generation_data]
        worst_time = [d['stats']['worst_time']
for d in self.generation_data]
        self.ax_time.plot(generations,
best_time, 'g-', label='Найкращий')
        self.ax_time.plot(generations,
avg_time, 'b-', label='Середній')
        self.ax_time.plot(generations,
worst_time, 'r-', label='Найгірший')
        self.ax_time.fill_between(generations,
best_time, worst_time,
                             alpha=0.2,
color='gray')
        self.ax_time.legend()
        self.ax_time.grid(True)
    def _update_params_plot(self) -> None:
        """Оновлення графіку розподілу
параметрів"""
        self.ax_params.clear()
        self.ax_params.set_title('Розподіл
параметрів')
        if not self.generation_data:
            return
        latest_population =
self.generation_data[-1]['population']
        param_names =
list(latest_population[0].keys())
        # Створюємо boxplot для кожного
параметра
        param_values = {name: [] for name in
param_names}
        for individual in latest_population:
            for name in param_names:

param_values[name].append(individual[name])
        self.ax_params.boxplot(
            list(param_values.values()),
            tick_labels=param_names  #
Замінено labels на tick_labels
        )

self.ax_params.set_xticklabels(param_names,
rotation=45)
        self.ax_params.grid(True)
    def _update_convergence_plot(self) ->
None:
        """Оновлення графіку конвергенції"""
        self.ax_convergence.clear()

self.ax_convergence.set_title('Конвергенція')

self.ax_convergence.set_xlabel('Покоління')

self.ax_convergence.set_ylabel('Відхилення')
        generations = [d['generation'] for d
in self.generation_data]
```

```
        std_fitness =
[d['stats']['fitness_std'] for d in
self.generation_data]
        self.ax_convergence.plot(generations,
std_fitness, 'b-')

self.ax_convergence.fill_between(generations,
0, std_fitness,

alpha=0.2, color='blue')
        self.ax_convergence.grid(True)
    def save(self, filename: str) -> None:
        """Збереження графіків у файл"""
        self.figure.savefig(filename, dpi=300,
bbox_inches='tight')
    def reset(self) -> None:
        """Скидання візуалізації"""
        self.generation_data.clear()
        self.setup_plots()
        self.canvas.draw()
class ImagePreview:
    """Клас для попереднього перегляду
зображень при їх обробці"""
    def __init__(self, master: tk.Tk):
        self.window = tk.Toplevel(master)
        self.window.title("Попередній
перегляд")
        self.window.withdraw()  # Приховуємо
вікно до першого використання
        # Фрейми для зображень
        self.frame_original =
ttk.LabelFrame(self.window, text="Оригінал")
        self.frame_original.pack(side=tk.LEFT,
padx=5, pady=5)
        self.frame_processed =
ttk.LabelFrame(self.window, text="Оброблене")

self.frame_processed.pack(side=tk.LEFT,
padx=5, pady=5)
        # Мітки для зображень
        self.label_original =
ttk.Label(self.frame_original)
        self.label_original.pack()
        self.label_processed =
ttk.Label(self.frame_processed)
        self.label_processed.pack()
        # Налаштування розміру
        self.max_size = (400, 400)
    def show(self, original_image: np.ndarray,
             processed_image: np.ndarray) ->
None:
        """Показ оригінального та обробленого
зображень"""
        # Конвертуємо зображення для
відображення
        original_pil =
self._prepare_image(original_image)
        processed_pil =
self._prepare_image(processed_image)
        # Оновлюємо мітки

self.label_original.configure(image=original_p
il)
        self.label_original.image =
original_pil  # Зберігаємо референс

self.label_processed.configure(image=processed
_pil)
        self.label_processed.image =
processed_pil  # Зберігаємо референс
        # Показуємо вікно
        self.window.deiconify()
        self.window.lift()
    def _prepare_image(self, image:
np.ndarray) -> ImageTk.PhotoImage:
        """Підготовка зображення для
відображення"""
        # Конвертуємо в RGB якщо потрібно
        if len(image.shape) == 2:
            image = cv2.cvtColor(image,
cv2.COLOR_GRAY2RGB)
        elif image.shape[2] == 4:
            image = cv2.cvtColor(image,
cv2.COLOR_BGRA2RGB)
        elif image.shape[2] == 3:
            image = cv2.cvtColor(image,
cv2.COLOR_BGR2RGB)
        # Конвертуємо в PIL Image
        pil_image = Image.fromarray(image)
        # Змінюємо розмір, зберігаючи
пропорції
        width, height = pil_image.size
        scale = min(self.max_size[0] / width,
self.max_size[1] / height)
        new_size = (int(width * scale),
int(height * scale))
        pil_image = pil_image.resize(new_size,
Image.Resampling.LANCZOS)
        return ImageTk.PhotoImage(pil_image)
    def hide(self) -> None:
        """Приховування вікна перегляду"""
        self.window.withdraw()
class LogHandler(logging.Handler):
    """Обробник для виведення логів у текстове
поле"""
    def __init__(self, text_widget):
        super().__init__()
        self.text_widget = text_widget
    def emit(self, record):
        msg = self.format(record)
        def _update():

self.text_widget.configure(state='normal')
            self.text_widget.insert('end', msg
+ '\n')
            self.text_widget.see('end')

self.text_widget.configure(state='disabled')
        # Використовуємо after для
потокобезпечного оновлення
        self.text_widget.after(0, _update)
class OptimizationUI:
    def __init__(self, master):
        self.master = master
        master.title("Універсальний
оптимізатор коду")
        self.supported_languages = {
            '.py': PythonExecutor,
            '.java': JavaExecutor,
            '.cpp': CppExecutor
        }
        self.workspace_dir = Path("workspace")
        self.workspace_dir.mkdir(parents=True,
exist_ok=True)
        self.history =
OptimizationHistory(Path("optimization_history
"))
```

```
        self.param_widgets = []
        self.code_executor = None
        self.optimization_thread = None
        self.stop_event = None
        self.preview_window = None
        # Спочатку створюємо всі віджети
        self.create_widgets()

        # Тільки після створення віджетів
налаштовуємо логування
        self.setup_logging()
    def create_widgets(self):
        # Головний контейнер
        main_container =
ttk.PanedWindow(self.master,
orient=tk.HORIZONTAL)
        main_container.pack(fill=tk.BOTH,
expand=True)
        # Ліва панель налаштувань
        left_panel = ttk.Frame(main_container)
        main_container.add(left_panel,
weight=1)
        # Права панель візуалізації
        right_panel =
ttk.Frame(main_container)
        main_container.add(right_panel,
weight=2)
        # Вибір мови програмування
        lang_frame =
ttk.LabelFrame(left_panel, text="Мова
програмування")
        lang_frame.pack(fill=tk.X, padx=5,
pady=5)
        self.language_var =
tk.StringVar(value='.py')
        languages = [('.py', 'Python'),
('.java', 'Java'), ('.cpp', 'C++')]
        for ext, name in languages:
            ttk.Radiobutton(
                lang_frame,
                text=name,
                variable=self.language_var,
                value=ext
            ).pack(side=tk.LEFT, padx=5)
        # Завантаження коду
        file_frame =
ttk.LabelFrame(left_panel, text="Завантаження
коду")
        file_frame.pack(fill=tk.X, padx=5,
pady=5)
        self.file_path = tk.StringVar()
        ttk.Entry(
            file_frame,
            textvariable=self.file_path,
            width=50
        ).pack(side=tk.LEFT, padx=5, pady=5,
fill=tk.X, expand=True)
        ttk.Button(
            file_frame,
            text="Вибрати файл",
            command=self.load_file
        ).pack(side=tk.LEFT, padx=5, pady=5)
        # Редактор коду
        code_frame =
ttk.LabelFrame(left_panel, text="Код для
оптимізації")
        code_frame.pack(fill=tk.BOTH, padx=5,
pady=5, expand=True)
        self.code_text =
scrolledtext.ScrolledText(
            code_frame,
            wrap=tk.WORD,
            width=60,
            height=10,
            font=('Courier New', 10)
        )
        self.code_text.pack(fill=tk.BOTH,
expand=True, padx=5, pady=5)
        # Параметри оптимізації
        self.param_frame =
ttk.LabelFrame(left_panel, text="Параметри
оптимізації")
        self.param_frame.pack(fill=tk.X,
padx=5, pady=5)
        # Контейнер для параметрів
        self.params_container =
ttk.Frame(self.param_frame)
        self.params_container.pack(fill=tk.X,
expand=True)
        param_button_frame =
ttk.Frame(self.param_frame)
        param_button_frame.pack(fill=tk.X,
padx=5, pady=5)
        ttk.Button(
            param_button_frame,
            text="Додати параметр",
            command=self.add_param_widgets
        ).pack(side=tk.LEFT, padx=5)
        ttk.Button(
            param_button_frame,
            text="Автовизначення",
            command=self.auto_detect_params
        ).pack(side=tk.LEFT, padx=5)
        # Налаштування генетичного алгоритму
        ga_frame = ttk.LabelFrame(left_panel,
text="Налаштування генетичного алгоритму")
        ga_frame.pack(fill=tk.X, padx=5,
pady=5)
        ttk.Label(ga_frame, text="Розмір
популяції:").grid(row=0, column=0, padx=5,
pady=2)
        self.population_size_var =
tk.IntVar(value=50)
        ttk.Entry(
            ga_frame,

textvariable=self.population_size_var,
            width=10
        ).grid(row=0, column=1, padx=5,
pady=2)
        ttk.Label(ga_frame, text="Кількість
поколінь:").grid(row=1, column=0, padx=5,
pady=2)
        self.generations_var =
tk.IntVar(value=50)
        ttk.Entry(
            ga_frame,
            textvariable=self.generations_var,
            width=10
        ).grid(row=1, column=1, padx=5,
pady=2)
        ttk.Label(ga_frame, text="Швидкість
мутації:").grid(row=2, column=0, padx=5,
pady=2)
        self.mutation_rate_var =
tk.DoubleVar(value=0.1)
        ttk.Entry(
```

```
            ga_frame,
textvariable=self.mutation_rate_var,
            width=10
        ).grid(row=2, column=1, padx=5,
pady=2)
        ttk.Label(ga_frame, text="Розмір
турніру:").grid(row=3, column=0, padx=5,
pady=2)
        self.tournament_size_var =
tk.IntVar(value=3)
        ttk.Entry(
            ga_frame,
textvariable=self.tournament_size_var,
            width=10
        ).grid(row=3, column=1, padx=5,
pady=2)
        # Кнопки керування
        control_frame = ttk.Frame(left_panel)
        control_frame.pack(fill=tk.X, padx=5,
pady=5)
        self.start_button = ttk.Button(
            control_frame,
            text="Запустити оптимізацію",
            command=self.start_optimization
        )
        self.start_button.pack(side=tk.LEFT,
padx=5)
        self.stop_button = ttk.Button(
            control_frame,
            text="Зупинити",
            command=self.stop_optimization,
            state=tk.DISABLED
        )
        self.stop_button.pack(side=tk.LEFT,
padx=5)
        ttk.Button(
            control_frame,
            text="Аналіз коду",
            command=self.analyze_code
        ).pack(side=tk.LEFT, padx=5)
        ttk.Button(
            control_frame,
            text="Історія",
            command=self.show_history
        ).pack(side=tk.LEFT, padx=5)
        # Фрейм для графіків
        viz_frame =
ttk.LabelFrame(right_panel, text="Візуалізація
процесу оптимізації")
        viz_frame.pack(fill=tk.BOTH,
expand=True, padx=5, pady=5)
        # Створення полотна для графіків
        self.figure = Figure(figsize=(8, 8),
dpi=100)
        self.canvas =
FigureCanvasTkAgg(self.figure,
master=viz_frame)
self.canvas.get_tk_widget().pack(fill=tk.BOTH,
expand=True)
        # Панель інструментів для графіків
        toolbar =
NavigationToolbar2Tk(self.canvas, viz_frame)
        toolbar.update()
        # Створення візуалізатора
        self.visualizer =
OptimizationVisualizer(self.figure,
self.canvas)
        # Фрейм для логів
        log_frame =
ttk.LabelFrame(right_panel, text="Логи та
статистика")
        log_frame.pack(fill=tk.BOTH,
expand=True, padx=5, pady=5)
        self.log_text =
scrolledtext.ScrolledText(
            log_frame,
            wrap=tk.WORD,
            width=60,
            height=10,
            font=('Courier New', 10)
        )
        self.log_text.pack(fill=tk.BOTH,
expand=True, padx=5, pady=5)
    def add_param_widgets(self):
        """Додавання віджетів для нового
параметра"""
        param_frame =
ttk.Frame(self.params_container)
        param_frame.pack(fill=tk.X, padx=5,
pady=2)
        # Назва параметра
        ttk.Label(param_frame,
text="Назва:").pack(side=tk.LEFT)
        name_var = tk.StringVar()
        name_entry = ttk.Entry(param_frame,
textvariable=name_var, width=15)
        name_entry.pack(side=tk.LEFT, padx=5)
        # Мінімальне значення
        ttk.Label(param_frame,
text="Мін:").pack(side=tk.LEFT)
        min_var = tk.DoubleVar()
        min_entry = ttk.Entry(param_frame,
textvariable=min_var, width=10)
        min_entry.pack(side=tk.LEFT, padx=5)
        # Максимальне значення
        ttk.Label(param_frame,
text="Макс:").pack(side=tk.LEFT)
        max_var = tk.DoubleVar()
        max_entry = ttk.Entry(param_frame,
textvariable=max_var, width=10)
        max_entry.pack(side=tk.LEFT, padx=5)
        # Кнопка видалення
        ttk.Button(
            param_frame,
            text="X",
            width=2,
            command=lambda:
self.remove_param_widgets(param_frame)
        ).pack(side=tk.LEFT, padx=5)

self.param_widgets.append((param_frame,
name_var, min_var, max_var))
    def remove_param_widgets(self, frame):
        """Видалення віджетів параметра"""
        for i, (param_frame, _, _, _) in
enumerate(self.param_widgets):
            if param_frame == frame:
                del self.param_widgets[i]
                frame.destroy()
                break
    def add_log_message(self, message: str,
level: str = "INFO"):
        """Додавання повідомлення в лог"""
```

```
        timestamp =
datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        self.log_text.insert(tk.END,
f"{timestamp} - {level}: {message}\n")
        self.log_text.see(tk.END)
    def _create_left_panel(self, parent:
ttk.Frame) -> None:
        """Створення лівої панелі з
налаштуваннями"""
        # Вибір мови програмування
        lang_frame = ttk.LabelFrame(parent,
text="Мова програмування")
        lang_frame.pack(fill=tk.X, padx=5,
pady=5)
        self.language_var =
tk.StringVar(value='.py')
        languages = [('.py', 'Python'),
('.java', 'Java'), ('.cpp', 'C++')]
        for ext, name in languages:
            ttk.Radiobutton(
                lang_frame,
                text=name,
                variable=self.language_var,
                value=ext
            ).pack(side=tk.LEFT, padx=5)
        # Завантаження коду
        file_frame = ttk.LabelFrame(parent,
text="Завантаження коду")
        file_frame.pack(fill=tk.X, padx=5,
pady=5)
        self.file_path = tk.StringVar()
        ttk.Entry(
            file_frame,
            textvariable=self.file_path,
            width=50
        ).pack(side=tk.LEFT, padx=5, pady=5,
fill=tk.X, expand=True)
        ttk.Button(
            file_frame,
            text="Вибрати файл",
            command=self.load_file
        ).pack(side=tk.LEFT, padx=5, pady=5)
        # Редактор коду
        code_frame = ttk.LabelFrame(parent,
text="Код для оптимізації")
        code_frame.pack(fill=tk.BOTH, padx=5,
pady=5, expand=True)
        self.code_text =
scrolledtext.ScrolledText(
            code_frame,
            wrap=tk.WORD,
            width=60,
            height=10,
            font=('Courier New', 10)
        )
        self.code_text.pack(fill=tk.BOTH,
expand=True, padx=5, pady=5)
        # Параметри оптимізації
        param_frame = ttk.LabelFrame(parent,
text="Параметри оптимізації")
        param_frame.pack(fill=tk.X, padx=5,
pady=5)
        self.param_widgets = []
        ttk.Button(
            param_frame,
            text="Додати параметр",
            command=self.add_param_widgets
        ).pack(side=tk.LEFT, padx=5, pady=5)
        ttk.Button(
            param_frame,
            text="Автовизначення",
            command=self.auto_detect_params
        ).pack(side=tk.LEFT, padx=5)
        # Налаштування генетичного алгоритму
        ga_frame = ttk.LabelFrame(parent,
text="Налаштування генетичного алгоритму")
        ga_frame.pack(fill=tk.X, padx=5,
pady=5)
        # Розмір популяції
        ttk.Label(ga_frame, text="Розмір
популяції:").grid(row=0, column=0, padx=5,
pady=2)
        self.population_size_var =
tk.IntVar(value=50)
        ttk.Entry(
            ga_frame,

textvariable=self.population_size_var,
            width=10
        ).grid(row=0, column=1, padx=5,
pady=2)
        ttk.Label(
            ga_frame,
            text="(рекомендовано: 30-100)"
        ).grid(row=0, column=2, padx=5,
pady=2)
        # Кількість поколінь
        ttk.Label(ga_frame, text="Кількість
поколінь:").grid(row=1, column=0, padx=5,
pady=2)
        self.generations_var =
tk.IntVar(value=50)
        ttk.Entry(
            ga_frame,
            textvariable=self.generations_var,
            width=10
        ).grid(row=1, column=1, padx=5,
pady=2)
        ttk.Label(
            ga_frame,
            text="(рекомендовано: 20-100)"
        ).grid(row=1, column=2, padx=5,
pady=2)
        # Швидкість мутації
        ttk.Label(ga_frame, text="Швидкість
мутації:").grid(row=2, column=0, padx=5,
pady=2)
        self.mutation_rate_var =
tk.DoubleVar(value=0.1)
        ttk.Entry(
            ga_frame,

textvariable=self.mutation_rate_var,
            width=10
        ).grid(row=2, column=1, padx=5,
pady=2)
        ttk.Label(
            ga_frame,
            text="(рекомендовано: 0.1-0.3)"
        ).grid(row=2, column=2, padx=5,
pady=2)
        # Розмір турніру
        ttk.Label(ga_frame, text="Розмір
турніру:").grid(row=3, column=0, padx=5,
pady=2)
        self.tournament_size_var =
tk.IntVar(value=3)
        ttk.Entry(
```

```
            ga_frame,
textvariable=self.tournament_size_var,
            width=10
        ).grid(row=3, column=1, padx=5,
pady=2)
        ttk.Label(
            ga_frame,
            text="(рекомендовано: 2-5)"
        ).grid(row=3, column=2, padx=5,
pady=2)
        # Кнопки керування
        control_frame = ttk.Frame(parent)
        control_frame.pack(fill=tk.X, padx=5,
pady=5)
        self.start_button = ttk.Button(
            control_frame,
            text="Запустити оптимізацію",
            command=self.start_optimization
        )
        self.start_button.pack(side=tk.LEFT,
padx=5)
        self.stop_button = ttk.Button(
            control_frame,
            text="Зупинити",
            command=self.stop_optimization,
            state=tk.DISABLED
        )
        self.stop_button.pack(side=tk.LEFT,
padx=5)
        ttk.Button(
            control_frame,
            text="Аналіз коду",
            command=self.analyze_code
        ).pack(side=tk.LEFT, padx=5)
        ttk.Button(
            control_frame,
            text="Історія",
            command=self.show_history
        ).pack(side=tk.LEFT, padx=5)
    def _create_right_panel(self, parent:
ttk.Frame) -> None:
        """Створення правої панелі з
візуалізацією"""
        # Фрейм для графіків
        viz_frame = ttk.LabelFrame(parent,
text="Візуалізація процесу оптимізації")
        viz_frame.pack(fill=tk.BOTH,
expand=True, padx=5, pady=5)
        # Створення полотна для графіків
        self.figure = Figure(figsize=(8, 8),
dpi=100)
        self.canvas =
FigureCanvasTkAgg(self.figure,
master=viz_frame)

self.canvas.get_tk_widget().pack(fill=tk.BOTH,
expand=True)
        # Панель інструментів для графіків
        toolbar =
NavigationToolbar2Tk(self.canvas, viz_frame)
        toolbar.update()
        # Створення візуалізатора
        self.visualizer =
OptimizationVisualizer(self.figure,
self.canvas)
        # Фрейм для логів
        log_frame = ttk.LabelFrame(parent,
text="Логи та статистика")
        log_frame.pack(fill=tk.BOTH,
expand=True, padx=5, pady=5)
        self.log_text =
scrolledtext.ScrolledText(
            log_frame,
            wrap=tk.WORD,
            width=60,
            height=10,
            font=('Courier New', 10)
        )
        self.log_text.pack(fill=tk.BOTH,
expand=True, padx=5, pady=5)
def add_param_widgets(self) -> None:
        """Додавання віджетів для нового
параметра"""
        param_frame =
ttk.Frame(self.master.nametowidget("!labelfram
e4"))
        param_frame.pack(fill=tk.X, padx=5,
pady=2)
        # Назва параметра
        ttk.Label(param_frame,
text="Назва:").pack(side=tk.LEFT)
        name_var = tk.StringVar()
        name_entry = ttk.Entry(param_frame,
textvariable=name_var, width=15)
        name_entry.pack(side=tk.LEFT, padx=5)
        # Мінімальне значення
        ttk.Label(param_frame,
text="Мін:").pack(side=tk.LEFT)
        min_var = tk.DoubleVar()
        min_entry = ttk.Entry(param_frame,
textvariable=min_var, width=10)
        min_entry.pack(side=tk.LEFT, padx=5)
        # Максимальне значення
        ttk.Label(param_frame,
text="Макс:").pack(side=tk.LEFT)
        max_var = tk.DoubleVar()
        max_entry = ttk.Entry(param_frame,
textvariable=max_var, width=10)
        max_entry.pack(side=tk.LEFT, padx=5)
        # Кнопка видалення
        ttk.Button(
            param_frame,
            text="X",
            width=2,
            command=lambda:
self.remove_param_widgets(param_frame)
        ).pack(side=tk.LEFT, padx=5)

self.param_widgets.append((param_frame,
name_var, min_var, max_var))
    def remove_param_widgets(self, frame:
ttk.Frame) -> None:
        """Видалення віджетів параметра"""
        for i, (param_frame, _, _, _) in
enumerate(self.param_widgets):
            if param_frame == frame:
                del self.param_widgets[i]
                frame.destroy()
                break
    def auto_detect_params(self) -> None:
        """Автоматичне визначення параметрів з
коду"""
        if not self.code_text.get('1.0',
tk.END).strip():

messagebox.showwarning("Попередження",
"Спочатку завантажте код")
```

```
            return
        try:
            # Очищення існуючих параметрів
            for param_frame, _, _, _ in
self.param_widgets:
                param_frame.destroy()
            self.param_widgets.clear()
            # Аналіз коду та додавання
параметрів
            code = self.code_text.get('1.0',
tk.END)
            language = self.language_var.get()
            if language == '.py':
                params =
self._analyze_python_params(code)
            elif language == '.java':
                params =
self._analyze_java_params(code)
            elif language == '.cpp':
                params =
self._analyze_cpp_params(code)
            else:
                raise
ValueError(f"Непідтримувана мова: {language}")
            # Додавання знайдених параметрів
            for name, (min_val, max_val) in
params:
                param_frame =
ttk.Frame(self.params_container)
                param_frame.pack(fill=tk.X,
padx=5, pady=2)
                # Назва параметра
                ttk.Label(param_frame,
text="Назва:").pack(side=tk.LEFT)
                name_var =
tk.StringVar(value=name)
                ttk.Entry(param_frame,
textvariable=name_var,
width=15).pack(side=tk.LEFT, padx=5)
                # Мінімальне значення
                ttk.Label(param_frame,
text="Мін:").pack(side=tk.LEFT)
                min_var =
tk.DoubleVar(value=min_val)
                ttk.Entry(param_frame,
textvariable=min_var,
width=10).pack(side=tk.LEFT, padx=5)
                # Максимальне значення
                ttk.Label(param_frame,
text="Макс:").pack(side=tk.LEFT)
                max_var =
tk.DoubleVar(value=max_val)
                ttk.Entry(param_frame,
textvariable=max_var,
width=10).pack(side=tk.LEFT, padx=5)
                # Кнопка видалення
                ttk.Button(
                    param_frame,
                    text="X",
                    width=2,
                    command=lambda
f=param_frame: self.remove_param_widgets(f)
                ).pack(side=tk.LEFT, padx=5)

self.param_widgets.append((param_frame,
name_var, min_var, max_var))
            self.logger.info(f"Автоматично
визначено параметри: {[p[0] for p, _ in
params]}")
        except Exception as e:
            self.logger.error(f"Помилка при
автовизначенні параметрів: {str(e)}")
            messagebox.showerror("Помилка",
str(e))
def _analyze_python_params(self, code: str) ->
List[Tuple[str, Tuple[float, float]]]:
        """Аналіз параметрів Python коду"""
        import ast
        tree = ast.parse(code)
        params = []
        # Шукаємо функцію evaluate
        for node in ast.walk(tree):
            if isinstance(node,
ast.FunctionDef) and node.name == 'evaluate':
                for arg in node.args.args:
                    # Намагаємось знайти
значення за замовчуванням
                    min_val, max_val = 0.0,
1.0
                    # Якщо є анотація типу,
перевіряємо її
                    if hasattr(arg,
'annotation'):
                        try:
                            if
isinstance(arg.annotation, ast.Subscript):
                                slice_value =
arg.annotation.slice
                                if
isinstance(slice_value, ast.Tuple):
                                    values =
[]
                                    for elt in
slice_value.elts:
                                        if
isinstance(elt, ast.Constant):

values.append(elt.value)
                                    if
len(values) == 2:

min_val, max_val = values
                        except:
                            pass
                    params.append((arg.arg,
(min_val, max_val)))
                break
        return params
    def _analyze_java_params(self, code: str)
-> List[Tuple[str, Tuple[float, float]]]:
        """Аналіз параметрів Java коду"""
        import re
        # Шукаємо сигнатуру методу evaluate
        pattern =
r'(?:public\s+static\s+)?double\s+evaluate\s*\
((.*?)\)'
        match = re.search(pattern, code)
        if not match:
            return []
        params_str = match.group(1)
        params = []
        # Розбираємо параметри
        for param in params_str.split(','):
            param = param.strip()
            if param:
                # Визначаємо ім'я параметра
                parts = param.split()
                if len(parts) >= 2:
```

```
                    param_name = parts[-1]
                    # Шукаємо коментарі з
діапазоном
                    range_pattern =
r'//\s*range:\s*\[([-+]?\d*\.?\d+)\s*,\s*([-
+]?\d*\.?\d+)\]'
                    range_match =
re.search(range_pattern, param)
                    if range_match:
                        min_val =
float(range_match.group(1))
                        max_val =
float(range_match.group(2))
                    else:
                        min_val, max_val =
0.0, 1.0
                    params.append((param_name,
(min_val, max_val)))
        return params
    def _analyze_cpp_params(self, code: str) -
> List[Tuple[str, Tuple[float, float]]]:
        """Аналіз параметрів C++ коду"""
        import re
        # Шукаємо сигнатуру функції evaluate
        pattern =
r'double\s+evaluate\s*\((.*?)\)'
        match = re.search(pattern, code)
        if not match:
            return []
        params_str = match.group(1)
        params = []
        # Розбираємо параметри
        for param in params_str.split(','):
            param = param.strip()
            if param:
                # Визначаємо ім'я параметра
                parts = param.split()
                if len(parts) >= 2:
                    param_name = parts[-1]
                    # Шукаємо коментарі з
діапазоном
                    range_pattern =
r'//\s*range:\s*\[([-+]?\d*\.?\d+)\s*,\s*([-
+]?\d*\.?\d+)\]'
                    range_match =
re.search(range_pattern, param)
                    if range_match:
                        min_val =
float(range_match.group(1))
                        max_val =
float(range_match.group(2))
                    else:
                        min_val, max_val =
0.0, 1.0
                    params.append((param_name,
(min_val, max_val)))
        return params
    def _analyze_code_for_params(self, code:
str) -> List[Tuple[str, Tuple[float, float]]]:
        """Аналіз коду для визначення
параметрів"""
        language = self.language_var.get()
        if language == '.py':
            return
self._analyze_python_params(code)
        elif language == '.java':
            return
self._analyze_java_params(code)
        elif language == '.cpp':
            return
self._analyze_cpp_params(code)
        else:
            raise ValueError(f"Непідтримувана
мова: {language}")
    def load_file(self) -> None:
        """Завантаження файлу з кодом"""
        filetypes = [
            ("All supported",
"*.py;*.java;*.cpp"),
            ("Python files", "*.py"),
            ("Java files", "*.java"),
            ("C++ files", "*.cpp")
        ]
        filepath =
filedialog.askopenfilename(filetypes=filetypes
)
        if not filepath:
            return
        try:
            # Визначаємо мову програмування
            ext =
os.path.splitext(filepath)[1]
            if ext not in ['.py', '.java',
'.cpp']:
                raise
ValueError(f"Непідтримуване розширення файлу:
{ext}")
            # Встановлюємо мову
            self.language_var.set(ext)
            # Завантажуємо код
            with open(filepath, 'r',
encoding='utf-8') as f:
                code = f.read()
            # Відображаємо код
            self.code_text.delete('1.0',
tk.END)
            self.code_text.insert('1.0', code)
            # Зберігаємо шлях
            self.file_path.set(filepath)
            # Автоматично визначаємо параметри
            self.auto_detect_params()
            self.logger.info(f"Файл успішно
завантажено: {filepath}")
        except Exception as e:
            self.logger.error(f"Помилка при
завантаженні файлу: {str(e)}")
            messagebox.showerror("Помилка",
str(e))
    def start_optimization(self) -> None:
        """Запуск процесу оптимізації"""
        if not self.code_text.get('1.0',
tk.END).strip():

messagebox.showwarning("Попередження",
"Спочатку завантажте код")
            return
        if not self.param_widgets:

messagebox.showwarning("Попередження",
"Додайте хоча б один параметр")
            return
        try:
            # Отримуємо параметри
            param_ranges = {}
            for _, name_var, min_var, max_var
in self.param_widgets:
                name = name_var.get().strip()
                if not name:
```

```
                    raise ValueError("Назва
параметра не може бути порожньою")
                min_val = min_var.get()
                max_val = max_var.get()
                if min_val >= max_val:
                    raise ValueError(
                        f"Мінімальне значення
має бути менше максимального "
                        f"для параметра
{name}"
                    )
                param_ranges[name] =
ParameterRange(name, min_val, max_val)
            # Створюємо виконавця коду
            code = self.code_text.get('1.0',
tk.END)
            if self.language_var.get() ==
'.py':
                executor_class =
PythonExecutor
            elif self.language_var.get() ==
'.java':
                executor_class = JavaExecutor
            elif self.language_var.get() ==
'.cpp':
                executor_class = CppExecutor
            else:
                raise
ValueError(f"Непідтримувана мова:
{self.language_var.get()}")
            self.code_executor =
executor_class(self.workspace_dir,
self.logger)
            self.code_executor.set_code(code)
            # Створюємо оптимізатор

self.optimizer = GeneticOptimizer(
                param_ranges=param_ranges,

population_size=self.population_size_var.get()
,

mutation_rate=self.mutation_rate_var.get(),

generations=self.generations_var.get(),

tournament_size=self.tournament_size_var.get()
,
                history=self.history
            )
            # Створюємо подію для зупинки
            self.stop_event =
threading.Event()
            # Оновлюємо стан кнопок

self.start_button.config(state=tk.DISABLED)

self.stop_button.config(state=tk.NORMAL)
            # Очищуємо візуалізацію
            self.visualizer.reset()
            # Запускаємо оптимізацію в
окремому потоці
            self.optimization_thread =
threading.Thread(
                target=self._run_optimization,
                args=(self.optimizer,)
            )
            self.optimization_thread.start()
            # Запускаємо оновлення інтерфейсу
            self.master.after(100,
self._check_optimization)
        except Exception as e:
            self.logger.error(f"Помилка при
запуску оптимізації: {str(e)}")
            messagebox.showerror("Помилка",
str(e))
            self._optimization_finished()
    def _run_optimization(self, optimizer:
GeneticOptimizer) -> None:
        """Виконання оптимізації в окремому
потоці"""
        try:
            with self.code_executor:
                optimizer.optimize(
                    self.code_executor,

progress_callback=self._update_progress,
                    stop_event=self.stop_event
                )
        except Exception as e:
            self.logger.error(f"Помилка під
час оптимізації: {str(e)}")
    def _update_progress(self, generation:
int, stats: dict) -> None:
        """Оновлення прогресу оптимізації"""
        if not hasattr(self,
'_progress_queue'):
            self._progress_queue =
queue.Queue()
        self._progress_queue.put((generation,
stats))
        # Додаємо детальне логування кожні 5
поколінь
        if generation % 5 == 0 or generation
== self.optimizer.generations - 1:
            current_best =
self.optimizer.population.best_individual
            self.logger.info("-" * 50)
            self.logger.info(f"Покоління
{generation}")
            self.logger.info(f"Поточні
найкращі параметри:")
            for name, value in
current_best.parameters.items():
                self.logger.info(f"    {name}:
{value:.4f}")

self.logger.info(f"Пристосованість:
{current_best.fitness:.4f}")
            self.logger.info(f"Час виконання:
{current_best.execution_time:.4f} сек")
            self.logger.info("-" * 50)
    def _check_optimization(self) -> None:
        """Перевірка стану оптимізації та
оновлення інтерфейсу"""
        # Обробляємо всі накопичені оновлення
        while hasattr(self, '_progress_queue')
and not self._progress_queue.empty():
            try:
                generation, stats =
self._progress_queue.get_nowait()

self.visualizer.update(generation, stats,
self.optimizer.population)
            except queue.Empty:
                break
        # Перевіряємо, чи все ще виконується
оптимізація
```

```
        if (self.optimization_thread and
self.optimization_thread.is_alive()):
            self.master.after(100,
self._check_optimization)
        else:
            self._optimization_finished()
    def stop_optimization(self) -> None:
        """Зупинка процесу оптимізації"""
        if self.stop_event:
            self.stop_event.set()
    def _optimization_finished(self) -> None:
        """Обробка завершення оптимізації"""

self.start_button.config(state=tk.NORMAL)

self.stop_button.config(state=tk.DISABLED)
        if hasattr(self, 'optimizer') and
self.optimizer.best_solution:
            self.logger.info("=" * 50)
            self.logger.info("ОПТИМІЗАЦІЮ
ЗАВЕРШЕНО")
            self.logger.info("Оптимальні
параметри:")
            for name, value in
self.optimizer.best_solution.parameters.items(
):
                self.logger.info(f"    {name}:
{value:.4f}")
            self.logger.info(f"Найкраща
пристосованість:
{self.optimizer.best_solution.fitness:.4f}")
            self.logger.info(f"Час виконання:
{self.optimizer.best_solution.execution_time:.
4f} сек")
            self.logger.info("=" * 50)
        else:
            self.logger.warning("Оптимізацію
завершено без результатів")
        if hasattr(self, '_progress_queue'):
            del self._progress_queue

def analyze_code(self) -> None:
        """Аналіз коду та надання
рекомендацій"""
        if not self.code_text.get('1.0',
tk.END).strip():

messagebox.showwarning("Попередження",
"Спочатку завантажте код")
            return
        try:
            code = self.code_text.get('1.0',
tk.END)
            language = self.language_var.get()
            # Створюємо вікно для результатів
аналізу
            analysis_window =
tk.Toplevel(self.master)
            analysis_window.title("Аналіз
коду")

analysis_window.geometry("600x400")
            # Додаємо текстове поле для виводу
результатів
            result_text =
scrolledtext.ScrolledText(
                analysis_window,
                wrap=tk.WORD,
                width=70,
                height=20,
                font=('Courier New', 10)
            )
            result_text.pack(fill=tk.BOTH,
expand=True, padx=5, pady=5)
            # Проводимо аналіз
            recommendations = []
            # Загальний аналіз

recommendations.extend(self._analyze_general(c
ode))
            # Специфічний аналіз для кожної
мови
            if language == '.py':

recommendations.extend(self._analyze_python_co
de(code))
            elif language == '.java':

recommendations.extend(self._analyze_java_code
(code))
            elif language == '.cpp':

recommendations.extend(self._analyze_cpp_code(
code))
            # Виводимо результати
            result_text.insert(tk.END,
"Результати аналізу коду:\n\n")
            for i, rec in
enumerate(recommendations, 1):
                result_text.insert(tk.END,
f"{i}. {rec}\n")

result_text.config(state=tk.DISABLED)
        except Exception as e:
            self.logger.error(f"Помилка при
аналізі коду: {str(e)}")
            messagebox.showerror("Помилка",
str(e))
    def _analyze_general(self, code: str) ->
List[str]:
        """Загальний аналіз коду"""
        recommendations = []
        # Аналіз довжини коду
        lines = code.splitlines()
        if len(lines) > 100:
            recommendations.append(
                "Код досить великий.
Розгляньте можливість розбиття на менші "
                "функції для кращої
читабельності та супроводу."
            )
        # Аналіз коментарів
        comment_lines = sum(
            1 for line in lines
            if line.strip().startswith(('#',
'//', '/*', '*'))
        )
        if comment_lines / len(lines) < 0.1:
            recommendations.append(
                "Рекомендується додати більше
коментарів для кращого "
                "розуміння коду
(рекомендований рівень - не менше 10%)."
            )
        # Аналіз складності
        nested_level = 0
        max_nested = 0
```

```
        for line in lines:
            if any(keyword in line for keyword
in ['if', 'for', 'while']):
                nested_level += 1
                max_nested = max(max_nested,
nested_level)
            if '}' in line:
                nested_level -= 1
        if max_nested > 3:
            recommendations.append(
                "Виявлено глибоку вкладеність
(> 3 рівнів). Це може "
                "ускладнити розуміння та
підтримку коду."
            )
        return recommendations
    def _analyze_python_code(self, code: str)
-> List[str]:
        """Специфічний аналіз Python коду"""
        import ast
        recommendations = []
        try:
            tree = ast.parse(code)
            # Аналіз імпортів
            imports = set()
            for node in ast.walk(tree):
                if isinstance(node,
ast.Import):
                    imports.update(n.name for
n in node.names)
                elif isinstance(node,
ast.ImportFrom):
                    imports.add(node.module)
            # Перевірка оптимізаційних
можливостей
            if 'numpy' not in imports:
                recommendations.append(
                    "Розгляньте використання
numpy для векторизації обчислень "
                    "та покращення
продуктивності."
                )
            # Аналіз циклів
            loops = sum(1 for node in
ast.walk(tree) if isinstance(node, ast.For))
            if loops > 5:
                recommendations.append(
                    "Велика кількість циклів
може вплинути на продуктивність. "
                    "Розгляньте можливість
векторизації операцій."
                )
            # Аналіз використання list
comprehension
            for node in ast.walk(tree):
                if isinstance(node, ast.For):
                    if
isinstance(node.body[0], ast.Append):

recommendations.append(
                            "Замініть цикл з
append() на list comprehension "
                            "для покращення
читабельності та продуктивності."
                        )
        except Exception as e:
            self.logger.error(f"Помилка при
аналізі Python коду: {str(e)}")
        return recommendations

    def _analyze_java_code(self, code: str) ->
List[str]:
        """Специфічний аналіз Java коду"""
        recommendations = []
        # Перевірка використання примітивних
типів
        if 'Double' in code or 'Integer' in
code:
            recommendations.append(
                "Використовуйте примітивні
типи (double, int) замість "
                "обгорток для кращої
продуктивності."
            )
        # Перевірка можливостей паралелізації
        if ('Thread' not in code and
'ExecutorService' not in code and
            'parallel' not in code):
            recommendations.append(
                "Розгляньте можливість
паралельних обчислень для "
                "покращення продуктивності."
            )
        return recommendations
    def _analyze_cpp_code(self, code: str) ->
List[str]:
        """Специфічний аналіз C++ коду"""
        recommendations = []

  # Перевірка використання посилань
        if 'std::vector' in code and '&' not
in code:
            recommendations.append(
                "Використовуйте посилання при
передачі великих об'єктів "
                "для уникнення копіювання."
            )
        # Перевірка оптимізації циклів
        if 'for' in code and '.size()' in
code:
            recommendations.append(
                "Зберігайте розмір контейнера
в змінній перед циклом "
                "для уникнення повторних
викликів size()."
            )
        return recommendations
    def show_history(self) -> None:
        """Показ вікна з історією
оптимізації"""
        history_window =
tk.Toplevel(self.master)
        history_window.title("Історія
оптимізації")
        history_window.geometry("800x600")
        # Створюємо віджети
        paned =
ttk.PanedWindow(history_window,
orient=tk.HORIZONTAL)
        paned.pack(fill=tk.BOTH, expand=True)
        # Ліва панель зі списком сесій
        left_frame = ttk.Frame(paned)
        paned.add(left_frame, weight=1)
        ttk.Label(left_frame,
text="Сесії:").pack(fill=tk.X)
        sessions_list = tk.Listbox(left_frame)
        sessions_list.pack(fill=tk.BOTH,
expand=True)
        # Права панель з деталями
```

```
        right_frame = ttk.Frame(paned)
        paned.add(right_frame, weight=2)
        # Графік прогресу
        fig = Figure(figsize=(6, 4), dpi=100)
        canvas = FigureCanvasTkAgg(fig,
master=right_frame)

canvas.get_tk_widget().pack(fill=tk.BOTH,
expand=True)
        # Таблиця результатів
        tree = ttk.Treeview(right_frame,
show='headings')
        tree.pack(fill=tk.BOTH, expand=True)
        # Заповнюємо список сесій
        sessions = self.history.get_sessions()
        for session in sessions:
            sessions_list.insert(tk.END,
session)
        def on_session_select(event):
            selection =
sessions_list.curselection()
            if not selection:
                return
            session_id =
sessions_list.get(selection[0])

self.history.load_session(session_id)
            # Оновлюємо графік
            fig.clear()
            ax = fig.add_subplot(111)
            df = self.history.to_dataframe()
            ax.plot(df['generation'],
df['fitness'], 'b-')
            # Продовження методу show_history
            ax.set_title('Прогрес
оптимізації')
            ax.set_xlabel('Покоління')
            ax.set_ylabel('Пристосованість')
            ax.grid(True)
            canvas.draw()
            # Оновлюємо таблицю
            tree.delete(*tree.get_children())
            # Налаштовуємо колонки
            columns = ['Параметр', 'Значення']
            tree['columns'] = columns
            for col in columns:
                tree.heading(col, text=col)
                tree.column(col, width=100)
            # Додаємо результати
            best_result =
self.history.get_best_result()
            if best_result:
                tree.insert('', 'end',
values=('Найкраща пристосованість',

f'{best_result.fitness:.6f}'))
                tree.insert('', 'end',
values=('Час виконання',

f'{best_result.execution_time:.6f} сек'))
                tree.insert('', 'end',
values=('Покоління',

str(best_result.generation)))
                tree.insert('', 'end',
values=('', ''))
                tree.insert('', 'end',
values=('Оптимальні параметри:', ''))
                for name, value in
best_result.parameters.items():
                    tree.insert('', 'end',
values=(name, f'{value:.6f}'))

sessions_list.bind('<<ListboxSelect>>',
on_session_select)
        # Кнопки експорту
        button_frame =
ttk.Frame(history_window)
        button_frame.pack(fill=tk.X, pady=5)
        def export_csv():
            try:
                file_path =
filedialog.asksaveasfilename(
                    defaultextension='.csv',
                    filetypes=[('CSV files',
'*.csv')]
                )
                if file_path:
                    df =
self.history.to_dataframe()
                    df.to_csv(file_path,
index=False)

messagebox.showinfo("Успіх", "Дані успішно
експортовано")
            except Exception as e:

messagebox.showerror("Помилка", f"Помилка при
експорті: {str(e)}")
        def export_plots():
            try:
                file_path =
filedialog.asksaveasfilename(
                    defaultextension='.png',
                    filetypes=[('PNG files',
'*.png')]
                )
                if file_path:

self.visualizer.save(file_path)

messagebox.showinfo("Успіх", "Графіки успішно
збережено")
            except Exception as e:

messagebox.showerror("Помилка", f"Помилка при
збереженні: {str(e)}")
        ttk.Button(button_frame, text="Експорт
в CSV",

command=export_csv).pack(side=tk.LEFT, padx=5)
        ttk.Button(button_frame,
text="Зберегти графіки",

command=export_plots).pack(side=tk.LEFT,
padx=5)
    def setup_logging(self):
        """Налаштування системи логування"""
        # Створюємо логер
        self.logger =
logging.getLogger('UniversalOptimizer')
        self.logger.setLevel(logging.INFO)
        # Очищуємо всі попередні обробники
        for handler in
self.logger.handlers[:]:
            self.logger.removeHandler(handler)
```

```
        # Створюємо обробник для виведення в
текстове поле
        text_handler =
LogTextHandler(self.log_text)
        formatter =
logging.Formatter('%(message)s')
        text_handler.setFormatter(formatter)
        self.logger.addHandler(text_handler)
    def cleanup(self) -> None:
        """Очищення ресурсів при закритті
програми"""
        try:
            # Зупиняємо оптимізацію, якщо вона
виконується
            if self.optimization_thread and
self.optimization_thread.is_alive():
                self.stop_optimization()

self.optimization_thread.join(timeout=1.0)
            # Очищуємо виконавця коду
            if self.code_executor:
                self.code_executor.cleanup()
            # Зберігаємо налаштування
            self.save_settings()
        except Exception as e:
            self.logger.error(f"Помилка при
очищенні ресурсів: {str(e)}")
    def save_settings(self) -> None:
        """Збереження налаштувань програми"""
        try:
            settings = {
                'language':
self.language_var.get(),
                'population_size':
self.population_size_var.get(),
                'generations':
self.generations_var.get(),
                'mutation_rate':
self.mutation_rate_var.get(),
                'tournament_size':
self.tournament_size_var.get(),
                'window_geometry':
self.master.geometry()
            }
            with open('settings.json', 'w',
encoding='utf-8') as f:
                json.dump(settings, f,
indent=4)
except Exception as e:
            self.logger.error(f"Помилка при
збереженні налаштувань: {str(e)}")
    def load_settings(self) -> None:
        """Завантаження налаштувань
програми"""
        try:
            if
os.path.exists('settings.json'):
                with open('settings.json',
'r', encoding='utf-8') as f:
                    settings = json.load(f)

self.language_var.set(settings.get('language',
'.py'))

self.population_size_var.set(settings.get('pop
ulation_size', 50))

self.generations_var.set(settings.get('generat
ions', 50))

self.mutation_rate_var.set(settings.get('mutat
ion_rate', 0.1))

self.tournament_size_var.set(settings.get('tou
rnament_size', 3))
                if 'window_geometry' in
settings:

self.master.geometry(settings['window_geometry
'])
        except Exception as e:
            self.logger.error(f"Помилка при
завантаженні налаштувань: {str(e)}")
def main():
    """Головна функція програми"""
    root = tk.Tk()
    app = OptimizationUI(root)
    # Завантажуємо налаштування
    app.load_settings()
    # Встановлюємо обробник закриття вікна
    root.protocol("WM_DELETE_WINDOW", lambda:
(app.cleanup(), root.destroy()))
    # Запускаємо головний цикл
    try:
        root.mainloop()
    except Exception as e:
        logging.error(f"Критична помилка:
{str(e)}")
        root.destroy()
if __name__ == "__main__":
    main()
```