Міністерство освіти і науки України
Український державний університет науки і технологій

Комп'ютерні технології і системи
(назва факультету)

Комп'ютерні інформаційні технології
(повна назва кафедри)

Пояснювальна записка

до кваліфікаційної роботи
ОС магістр

на тему: ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ ГЕНЕТИЧНИХ АЛГОРИТМІВ

за освітньою програмою Інженерія програмного забезпечення

зі спеціальності: 121 Інженерія програмного забезпечення

Виконав: студент Групи: ПЗ2426 / Тетяна Михайлова /

Керівник: / доцент Вадим Андрющенко /

Нормоконтролер: / доцент Світлана Волкова /

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

Дніпро – 2026 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Computer technologies and systems

Computer information technology

Explanatory Note
to the Thesis for
Master's Degree

on the topic: Analysis of the Time Performance of Genetic Algorithms

according to educational curriculum: Software engineering

in the Speciality: 121 Software engineering

Done by the student of the group: PZ2426 / Tetiana MYKHAILOVA /

Scientific Supervisor: / Vadym ANDRIUSHCHENKO /

Normative controller : / Svitlana VOLKOVA /

Dnipro – 2026

РЕФЕРАТ

НАЗВА: ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ ГЕНЕТИЧНИХ АЛГОРИТМІВ

Analysis of the Time Performance of Genetic Algorithms

АВТОР: Михайлова Тетяна Олександрівна

НАУКОВИЙ КЕРІВНИК: Андрющенко Вадим Олександрович

Пояснювальна записка до кваліфікаційної роботи магістра виконана на 97 сторінках, містить 39 рисунків, 4 додатки, 39 джерел

Магістерська робота присвячена дослідженню часової ефективності генетичних алгоритмів та аналізу впливу їх основних параметрів на тривалість виконання алгоритму. Генетичні алгоритми належать до стохастичних методів оптимізації та широко застосовуються для розв’язання складних задач у різних галузях інформаційних технологій, проте їх часові характеристики значною мірою залежать від конфігурації параметрів та властивостей оптимізаційної задачі.

У роботі проаналізовано теоретичні засади генетичних алгоритмів, їх структуру, оператори селекції, кросинговеру та мутації, а також критерії зупинки. Окрему увагу приділено питанням оцінювання часової ефективності стохастичних алгоритмів і чинникам, що впливають на обчислювальні витрати.

Розроблено програмний інструмент мовою Python для експериментального дослідження часової ефективності генетичних алгоритмів. Програма реалізує класичний генетичний алгоритм із можливістю налаштування основних параметрів, автоматично вимірює час виконання та кількість поколінь до зупинки, зберігає результати у CSV-файл і будує графіки для подальшого аналізу. Для експериментів використано стандартні тестові функції оптимізації Sphere, Rastrigin та Rosenbrock.

У межах роботи проведено серію експериментів для шести конфігурацій алгоритму, що дозволило проаналізувати вплив кількості поколінь, розміру популяції, розмірності задачі, ймовірності кросинговеру та параметрів мутації на час виконання генетичного алгоритму. Отримані результати підтверджують, що найбільший вплив на часову ефективність мають розмір популяції, кількість поколінь та параметри мутації, причому ступінь їх впливу залежить від складності оптимізаційної задачі.

Результати дослідження можуть бути використані для обґрунтованого налаштування параметрів генетичних алгоритмів з метою підвищення їх швидкодії та ефективності.

Ключові слова: генетичний алгоритм, часова ефективність, оптимізація, параметри алгоритму, еволюційні обчислення, Python.

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: Компютерних технологій і систем

Кафедра: компютерні інформаційні технології

Рівень вищої освіт: ОС магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри _____

“___”_____ 202___року

ЗАВДАННЯ

На кваліфікційну роботу ОС маістр

студенту Михайловій Тетяні Олександрівні

1. Тема роботи: Дослідження часової ефективності генетичних алгоритмів

Керівник Андрющенко Вадим Олександрович

2. Строк подання студентом роботи: 10.01.2026 р.

3.Вихідні данні до роботи:

4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):

4.1 Теоретичні основи дослідження часової ефективності генетичних алгоритмів.

4.2 Аналіз існуючих методів та програмних засобів.

4.3 Програмна реалізація та аналіз часової ефективності генетичних алгоритмів.

5. Перелік графічного матеріалу:

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етпів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Дослідження параметрів генетичних алгоритмів	1.08.2025-1.09.2025	20%
2	Дослідження існуючих програмних засобів для еволюційних алгоритмів	2.09.2025-22.09.2025	30%
3	Дослідження мови програмування Python для написання програми	23.09.2025-15.10.2025	40%
4	Розробка програми для проведення експерименту	16.10.2025-15.11.2025	50%
5	Проведення експерименту та аналіз результатів	16.11.2025-9.12.2025	80%
6	Оформлення пояснювальної записки та демонстраційних матеріалів	10.12.2025-9.01.2026	90%
7	Подання кваліфікаційної роботи до кафедри	10.01.2026	100%
8	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	22.01.2026	

Студент Тетяна МИХАЙЛОВА

Виклдач Вадим АНДРЮЩЕНКО

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАК

ГА- генетичні алгоритми

GA- genetic algorithms

CSV- Comma-Separated Values

ВСТУП

Сучасні завдання оптимізації характеризуються високим рівнем складності, нелінійністю та великою розмірністю пошукового простору. Це ускладнює застосування традиційних детермінованих методів, які часто виявляються недостатньо ефективними або потребують значних обчислювальних ресурсів.

В таких умовах активно розвиваються стохастичні методи оптимізації, зокрема генетичні алгоритми (ГА). ГА базуються на принципах природної селекції та спадковості і широко застосовуються в різних галузях:

- Машинне навчання

- Оптимізація моделей штучного інтелекту
- Пошук архітектур нейронних мереж
- Навчання агентів у системах з підкріпленням
- Логістика
- Енергетика
- Робототехніка
- Біоінформатика
- Фармакологія (зокрема, пошук нових лікарських молекул).

Збільшення масштабів застосування ГА викликає зростання вимог до їх продуктивності. Часто обчислювальні витрати можуть бути значними, особливо при збільшенні розмірності задачі або для задач реального часу.

Часові характеристики генетичних алгоритмів залежать від багатьох факторів, таких як:

- Розмір популяції
- Тип селекції
- Кросинговер
- Мутація
- Ймовірності операторів
- Вид функції пристосованості
- Критерій зупинки.

Незважаючи на велику кількість досліджень, присвячених точності та якості роботи ГА, систематичний аналіз часової ефективності та впливу різних конфігурацій алгоритму на його швидкість недостатньо висвітлений.

Тому оцінка часової ефективності генетичних алгоритмів є важливою проблемою як з теоретичної, так і з прикладної точки зору. Вона дозволить підвищити продуктивність еволюційних методів в різних галузях.

Об’єкт дослідження

Процес функціонування генетичних алгоритмів у задачах оптимізації.

Предмет дослідження

Часові характеристики роботи генетичного алгоритму залежно від його параметрів, конфігурацій та властивостей задачі.

Мета роботи

Дослідити вплив параметрів генетичного алгоритму на його часову ефективність та розробити програмний інструмент для проведення експериментів, збору статистичних даних і порівняння різних конфігурацій ГА.

Для досягнення мети необхідно розв’язати такі завдання:

1. Проаналізувати наукові джерела щодо застосування генетичних алгоритмів і підходів до оцінювання їх ефективності.
2. Розглянути та порівняти існуючі програмні засоби для моделювання еволюційних алгоритмів.
3. Розробити програмний інструмент для дослідження та порівняння часової ефективності ГА.
4. Реалізувати набір тестових функцій оптимізації, придатних для експериментів.
5. Провести експериментальні дослідження залежності часу виконання ГА від його параметрів та характеристик задачі.
6. Виконати статистичний аналіз отриманих результатів і сформулювати рекомендації щодо оптимальних конфігурацій ГА.

Методи дослідження

У роботі застосовуються методи еволюційних обчислень, математичного моделювання, експериментальних вимірювань часу, статистичного аналізу, а також методи програмної реалізації алгоритмів у середовищі Python.

Наукова новизна

Полягає у проведенні систематичного дослідження часової ефективності генетичного алгоритму для різних конфігурацій, аналізі впливу основних параметрів ГА на швидкодію та розробленні програмного інструменту для автоматизації експериментів і збору метрик продуктивності.

Практичне значення роботи

Результати роботи можуть бути застосовані для вибору оптимальних параметрів ГА у задачах штучного інтелекту, машинного навчання, логістики, інженерного проєктування, біоінформатики та інших галузях. Розроблений програмний інструмент може використовуватися у навчальному процесі та як

експериментальна платформа для подальших досліджень еволюційних алгоритмів.

Структура роботи

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ ГЕНЕТИЧНИХ АЛГОРИТМІВ

1.1 Обґрунтування вибору теми та постановка проблеми

Сучасні задачі оптимізації, що виникають у штучному інтелекті, машинному навчанні, інженерному проєктуванні, логістичних системах, біоінформатиці та інших високотехнологічних галузях, характеризуються значною обчислювальною складністю. Ці задачі часто мають великий пошуковий простір, багато локальних мінімумів, відсутність аналітичних градієнтів або їхню складну обчислюваність. Класичні детерміновані методи оптимізації в таких випадках або не гарантують збіжності, або потребують значних часових ресурсів.

Важливу роль у вирішенні таких задач відіграють стохастичні методи оптимізації, зокрема генетичні алгоритми (ГА). Їхня популяційна природа та випадкові оператори дозволяють ефективно досліджувати складні простори рішень і знаходити наближені оптимуми там, де традиційні методи неефективні.

Незважаючи на значний розвиток теорії еволюційних алгоритмів, проблема часової ефективності ГА залишається недостатньо дослідженою. Більшість наукових праць зосереджені на точності, якості або збіжності

алгоритму, тоді як питання швидкодії, масштабованості та впливу параметрів на час роботи розглядаються поверхово або лише в контексті конкретних завдань.

У багатьох сучасних застосуваннях (штучний інтелект, обробка великих даних, медичні симуляції, оптимізація технологічних процесів) час виконання алгоритму є критичним фактором, що визначає його придатність.

Тому, необхідно визначити, як параметри ГА впливають на його часову ефективність, та сформулювати загальні рекомендації щодо вибору конфігурацій ГА.

1.2 Сфера застосування генетичних алгоритмів у сучасних технологіях

Генетичні алгоритми (ГА) належать до класу еволюційних обчислень і відзначаються універсальністю у розв’язанні задач, для яких складно або неможливо застосувати традиційні методи оптимізації. Завдяки популяційному підходу, використанню випадкових операторів та здатності ефективно досліджувати складні, нерегулярні й багатовимірні простори рішень, ГА широко застосовуються в сучасних високотехнологічних галузях.

Однією з найважливіших сучасних сфер застосування генетичних алгоритмів є штучний інтелект. Зі стрімким зростанням складності моделей машинного навчання, збільшенням обсягів даних та потребою у гнучкому налаштуванні параметрів алгоритмів, здатність генетичних алгоритмів виконувати глобальний пошук у багатовимірних просторах оптимізації набуває особливого значення. Еволюційні методи дозволяють знаходити ефективні рішення в ситуаціях, коли традиційні оптимізаційні підходи виявляються надто повільними або обмеженими структурою задачі.

Одним із ключових напрямів є оптимізація гіперпараметрів моделей машинного навчання. Налаштування складних моделей — нейронних мереж, методів опорних векторів, градієнтних бустінгів — потребує пошуку оптимальних значень десятків параметрів. ГА дозволяють здійснювати глобальний пошук у цьому просторі, уникаючи недоліків повного перебору чи грід-пошуку, які швидко стають обчислювально непідйомними. Завдяки

еволюційному підходу можливо значно зменшити кількість необхідних ітерацій та виконати оптимізацію навіть тоді, коли поверхня помилки має складний, нерівномірний ландшафт.

Важливим напрямом застосування ГА є еволюційний пошук архітектур нейронних мереж (Neural Architecture Search). Сучасні дослідження Google та DeepMind показали, що еволюційні методи здатні автоматично генерувати архітектури, які конкурують або перевершують моделі, створені вручну досвідченими інженерами. У цьому процесі ГА «еволюціонують» структури мережі: додають або видаляють шари, змінюють їх параметри, комбінують успішні архітектурні модулі. Такий підхід дає можливість отримати нові, ефективні архітектури без ручного втручання.

У деяких діалогових системах та моделях генерації дій еволюційний підхід використовується для оптимізації стратегій відповіді або комбінування кількох поведінкових моделей. ГА здатні знаходити оптимальні послідовності дій, що максимізують корисність або якість відповіді, працюючи в умовах, де методи, засновані на градієнті, або непридатні, або недостатньо стабільні.

У цілому роль генетичних алгоритмів у штучному інтелекті постійно зростає. Сучасні задачі AI характеризуються величезними просторами параметрів, складною структурою оптимізації та необхідністю автоматизованого пошуку. У таких умовах еволюційні методи виступають природним і ефективним інструментом глобальної оптимізації, що підсилює актуальність дослідження їх часової ефективності та продуктивності.

Генетичні алгоритми (ГА) знаходять широке застосування у сфері біоінформатики та фармакології, де вони сприяють вирішенню завдань, що мають ключове значення для розвитку медичних технологій та наукових досліджень.

В фармакології ГА використовуються для розробки нових лікарських препаратів (drug discovery). Завдяки здатності еволюційно оптимізувати структури молекул, ГА дозволяють оцінити їх взаємодію з білками, прогнозувати токсичність, біодоступність та інші фармакокінетичні властивості. Це значно скорочує час на попередній відбір кандидатів для подальших лабораторних досліджень.

У сфері біоінформатики ГА застосовуються для вирішення різноманітних завдань, таких як вирівнювання ДНК-послідовностей, виявлення генетичних мотивів, реконструкція білкових структур, моделювання еволюційних процесів та аналіз мутацій. Їх ефективність особливо помітна у задачах з великим об'ємом даних, де класичні методи стають неефективними через високу складність.

ГА також знаходять застосування в персоналізованій медицині для оптимізації схем лікування з урахуванням індивідуальних параметрів пацієнта. Це дозволяє визначити оптимальні комбінації препаратів, дозувань та графіків прийому, враховуючи численні взаємозалежні фактори.

Генетичні алгоритми знаходять широке застосування в сучасних технологіях, зокрема в галузях інженерії, робототехніки та логістики. Їх ефективність обумовлена здатністю розв'язувати складні оптимізаційні задачі, знаходячи нестандартні та інноваційні рішення.

В робототехніці генетичні алгоритми використовуються для:

- Оптимізації траєкторій руху роботів.
- Планування поведінки та стратегій дій.
- Проектування нових форм та конструкцій роботів.
- Одночасного налаштування як структури, так і алгоритмів управління.

Еволюційні методи дозволяють створювати незвичайні форми та схеми поведінки, які важко отримати за допомогою традиційних підходів.

В інженерії генетичні алгоритми застосовуються для:

- Оптимізації форм конструкцій.
- Мінімізації маси та покращення аеродинамічних властивостей.
- Проектування електронних схем.
- Налаштування складних технологічних процесів.

Генетичні алгоритми ефективно працюють з високорозмірними моделями, враховуючи вплив кожного параметра на декілька цільових метрик одночасно.

В логістиці генетичні алгоритми використовуються для розв'язання задач:

- Маршрутизації транспорту (TSP, VRP).
- Розподілу ресурсів.
- Планування поставок.
- Оптимізації складських систем.
- Побудови ефективних графіків роботи.

Комбінаторна природа цих задач робить еволюційні методи надзвичайно ефективними та часто більш продуктивними, ніж класичні оптимізаційні методи.

1.3 Теоретичні основи генетичних алгоритмів

Генетичні алгоритми (ГА) належать до класу еволюційних обчислень і базуються на імітації природної селекції та спадковості. Вони є потужним інструментом глобальної оптимізації, здатним ефективно працювати у задачах з великими, нелінійними та слабоформалізованими просторами пошуку. Теоретичні основи ГА були закладені в роботах Голдберга, Голланда, Мітчелл та ін., а детальні математичні аспекти наведено у монографії Кононюка [27].

1.3.1 Основні оператори ГА (селекція, кросинговер, мутація)

Генетичний алгоритм імітує процеси природної еволюції та використовує біологічно натхненні оператори для поступового вдосконалення рішень. Основними елементами, що формують нові покоління, є селекція, кросинговер та мутація. Ефективність алгоритму залежить від взаємодії цих операторів, які впливають на швидкість зближення до оптимального розв'язку, збереження різноманітності популяції та здатність уникнути потрапляння в локальні мінімуми. [9][13]

Селекція — це механізм відбору особин (кандидатних рішень), які будуть брати участь у створенні наступного покоління. Вона забезпечує відбір кращих рішень згідно з функцією пристосованості (fitness). Мета селекції — підвищити частоту появи успішних ознак у популяції [2].

Найпоширеніші методи:

- Пропорційна селекція (roulette wheel selection)
  Імовірність вибору рішення пропорційна його пристосованості. Метод описано в роботах Голдберга [2].
- Турнірна селекція (tournament selection)
  З кількох випадкових особин обирається найкраща. Турнірна селекція забезпечує контроль селективного тиску та є однією з найпопулярніших технік [14].
- Рангова селекція (rank selection)
  Імовірність вибору залежить не від абсолютного значення fitness, а від

рангу особини в популяції. Метод зменшує ризик домінування однієї особини [14].

Кросинговер (crossover) — оператор, що створює нових особин шляхом поєднання «генетичної інформації» двох батьків. Він виконує роль експлуатації пошукового простору — комбінує вдалі фрагменти вже знайдених рішень [2], [5].

Типові види:

- Однокрапковий кросинговер (one-point crossover).
  Хромосома розділяється в одній точці; потім фрагменти обмінюються між батьками.
- Двокрапковий кросинговер (two-point crossover).
  Два розділи — середній сегмент міняється між особинами.
- Арифметичний кросинговер (arithmetic crossover).
  Для неперервних задач (real-coded GA) широко застосовується формування нащадків за формулою(1.1):

$$child=\alpha parent1+(1-\alpha)parent2 \quad (1.1)$$

що описано в роботах Деба та Мітчелла [13].

Мутація - забезпечує варіативність у популяції та відповідає за дослідження нових областей простору рішень (exploration). Без мутації алгоритм може передчасно збігатися до локальних мінімумів [5], [2].

Основні види мутацій для неперервних задач:

- Гаусівська мутація (Gaussian mutation).
  До гена додається шум, згенерований за нормальним розподілом (1.2)

$$x_i'=x_i+N(0,\sigma 2) \quad (1.2)$$

- Рівномірна мутація (uniform mutation).
  Значення гена замінюється випадковою величиною з дозволеного діапазону.
- Крокова мутація (step mutation).
  Значення збільшується або зменшується на фіксований крок.

Оператори відіграють ключову роль в процесі оптимізації за допомогою генетичних алгоритмів (ГА). Ефективність алгоритму значною мірою залежить від збалансованої взаємодії трьох основних операторів: селекції, кросинговера та мутації.

Селекція, спрямовуючи пошук до області кращих рішень, забезпечує високий тиск відбору.

Кросинговер відповідає за комбінування успішних ознак з батьківських хромосом, сприяючи експлуатації вже відомого простору рішень.

Мутація, вносячи випадкові зміни в геном, запобігає застою і дозволяє досліджувати нові області простору рішень, що відповідає експлорації.

Баланс між експлуатацією і дослідженням є критичним для ефективності алгоритму [9], [5].

Саме тому дослідження часової ефективності ГА потребує аналізу того, як різні параметри операторів впливають на загальну швидкість і стабільність роботи алгоритму.

1.3.2 Подання рішень у генетичних алгоритмах (кодування хромосом)

Представлення (кодування) рішень є фундаментальною складовою генетичного алгоритму, оскільки воно визначає спосіб зберігання, модифікації та комбінування потенційних розв'язків задачі під час еволюційного процесу. Хромосома — це формальне подання одного кандидата на розв’язок, яке піддається діям генетичних операторів. Вибір способу кодування безпосередньо впливає на ефективність роботи ГА, зокрема на швидкість збіжності, можливість уникнення локальних мінімумів і придатність операторів селекції, кросинговеру та мутації [5], [13].

Бінарне кодування є класичним і одним з найперших способів подання рішень у ГА. Значення параметрів представляються у вигляді бітових рядків (послідовностей 0 і 1). Перевагою такого підходу є універсальність і простота реалізації генетичних операторів. Кононюк виділяє бінарне кодування як фундаментальну модель, але відзначає його неефективність у задачах неперервної оптимізації через втрату точності [Кононюк, 2008].

Також використовується цілочисельне кодування. Даний метод представлення застосовується у випадках, коли параметри задачі мають дискретну природу. Прикладами таких задач є маршрутизація, задача комівояжера та розподіл ресурсів.

В рамках цієї схеми хромосома представляє собою послідовність цілих чисел, кожне з яких безпосередньо відповідає елементу шуканого рішення.

Переваги цілочисельне кодування:

- природне подання дискретних задач;
- можливість застосування спеціалізованих операторів (наприклад, часткова перестановка у TSP).

Цілочисельне кодування також широко застосовується в еволюції структур нейронних мереж та інших комбінаційних задачах [13].

Дійсне (real-coded) кодування – виористовується для задач оптимізації з неперервними змінними, таких як тестові функції Sphere, Rastrigin та Rosenbrock, найбільш ефективним є дійсне кодування. У цьому випадку хромосома представляється як вектор дійсних чисел (1.3):

$$x=(x_1, x_2, \ldots, x_n),\ x_i \in R$$

(1.3)

Переваги:

Найкраще підходить для неперервних функцій, таких як Sphere, Rastrigin, Rosenbrock.

Переваги:

- висока точність,
- природність представлення,
- краща взаємодія з арифметичними операторами,
- швидша збіжність [5].

Саме цей тип кодування використовується у нашій програмі, оскільки всі тестові функції — неперервні.

Окрім стандартних підходів, у вирішенні складних задач оптимізації застосовують нетрадиційні способи представлення даних.

Серед них:

- Деревоподібне кодування, яке знаходить застосування в генетичному програмуванні (GP) для еволюції математичних формул та програмних структур;
- Матричне кодування, що використовується для оптимізації маршрутів та задач, пов'язаних з графами;

- Кодування з обмеженнями, яке гарантує допустимість кожної хромосоми в популяції.

В окремих випадках генетичні алгоритми інтегруються з іншими методами, наприклад, з локальним пошуком, утворюючи так звані генетично-гібридні системи (memetic algorithms) [13], [22].

Вибір типу кодування визначає:

- структуру пошукового простору;
- поведінку генетичних операторів;
- збіжність та стабільність алгоритму;
- якість знайдених розв’язків;
- часову ефективність, що є центральним об’єктом цього дослідження.

Дійсне кодування зазвичай є оптимальним для задач неперервної оптимізації, що підтверджується численними дослідженнями і тому його обрано у даній роботі як основний підхід до представлення рішень.

1.3.3 Функція пристосованості та критерії зупинки

Дуже важливим поняттям у генетичних алгоритмах вважається функція пристосованості (fitness function), яка інакше називається функцією оцінки. Вона являє міру пристосованості даної особини в популяції. Ця функція відіграє найважливішу роль, оскільки дозволяє оцінити ступінь пристосованості конкретних особин у популяції і вибрати з них найбільш пристосовані (тобто мають найбільші значення функції пристосованості) відповідно з еволюційним принципом виживання «найсильніших» (які найкраще пристосувалися). [27]

Критерії зупинки, у свою чергу, визначають момент завершення роботи алгоритму і впливають як на точність результату, так і на часову ефективність розв’язання[9], [13].

В контексті задач мінімізації, які є характерними для дослідження еволюційних алгоритмів, функція пристосованості повертає числове значення, що інтерпретується як міра "вартості" або "якості" розв'язку. Чим менше це значення, тим кращим вважається розв'язок.

Вигляд функції пристосованості може варіюватися залежно від типу задачі:

- Аналітичні функції: використовуються в тестових задачах, таких як Sphere, Rastrigin, Rosenbrock.

- Евристичні функції: відображають якість розв'язків комбінаторної природи, наприклад, маршруту, розкладу тощо.
- Моделювальні функції: базуються на симуляціях, імітаційних моделях або методах машинного навчання.

Для задач неперервної оптимізації вигляд функції пристосованості збігається з формальною моделлю задачі. Наприклад (1.4),

$$\text{fitness}(x) = f(x), \tag{1.4}$$

де $f(x)$ — тестова функція оптимізації.

В еволюційних алгоритмах функція пристосованості виконує роль "екологічного середовища", яке визначає, які розв'язки будуть "виживати" та передавати генетичну інформацію наступним поколінням. Ефективність селекції прямо пропорційна здатності функції чітко відображати різницю між якісними і неякісними розв'язками.

Згідно зі стандартними підходами еволюційних обчислень [9], функція пристосованості повинна:

- бути обчислюваною у розумний час;
- відображати реальну якість рішення;
- бути монотонною відносно критеріїв задачі (краще рішення має відповідати кращому значенню функції пристосованості);
- не мати великих ділянок із однаковим значенням (flat landscape), оскільки це ускладнює пошук;
- бажано бути гладкою, але це не є обов’язковим (особливо у складних еволюційних задачах).

У рамках нашого дослідження вибрані тестові функції (Sphere, Rastrigin, Rosenbrock), які відповідають цим вимогам і є класичними бенчмарками для оцінювання поведінки ГА.

Тепер розглянемо критерії зупинки генетичного алгоритму. Алгоритм не може працювати нескінченно довго, тому необхідний механізм визначення моменту завершення еволюції. Залежно від задачі використовують один або кілька критеріїв зупинки:

- Досягнення максимальної кількості поколінь

Це є найпоширеніший критерій зупинки (1.5).

$$t \geq t_{max}$$

(1.5)

Цей підхід дозволяє контролювати часову складність і є критично важливим у дослідженні продуктивності ГА. Голдберг підкреслює, що цей критерій є базовим для аналізу продуктивності ГА [9].

· Досягнення заданого значення функції пристосованості

Алгоритм завершується, коли знайдене рішення достатньо добре (1.6):

$$f(x) \leq \varepsilon$$

(1.6)

· Відсутність прогресу (stagnation)

Якщо протягом k поколінь (k — це поріг кількості поколінь без покращення) найкраще значення не покращується, пошук припиняється. Це дозволяє уникати витрачання часу після збіжності до локального мінімуму [13] .

· Обмеження на час роботи

Для задач реального часу або високої складності встановлюється максимальний час виконання.
Цей критерій часто використовується в дослідженнях часової ефективності.

У контексті нашого дослідження часової ефективності генетичного алгоритму функція пристосованості та критерії зупинки відіграють ключову роль, оскільки саме вони визначають характер і тривалість еволюційного процесу.

Функція пристосованості задає структуру простору пошуку, або ще можна назвати ландшафт оптимізації, у межах якого відбувається оптимізація. Властивості цього простору— його гладкість, наявність локальних мінімумів, крутість схилів чи вузькі долини — впливають на те, наскільки швидко алгоритм здатен збігатися до прийнятного рішення [27]. Наприклад, у разі гладких і опуклих функцій (як Sphere) генетичний алгоритм, як правило, досягає мінімуму відносно швидко. Натомість функції зі складною структурою простору пошуку, зокрема такі, що містять значну кількість локальних мінімумів (Rastrigin) або мають вузькі вигнуті області (Rosenbrock), вимагають набагато більшої кількості поколінь і, відповідно, довшого часу обчислень.

Критерії зупинки є важливим доповненням до функції пристосованості в генетичних алгоритмах, оскільки вони визначають межі часу виконання. Вибір таких параметрів, як максимальна кількість поколінь, поріг точності чи умова

відсутності прогресу, дозволяє контролювати співвідношення між обчислювальними витратами та якістю отриманого розв'язку.

В залежності від встановлених критеріїв зупинки, алгоритм може завершувати роботу значно раніше або пізніше, що істотно впливає на час виконання. Наприклад, надмірно велика кількість заданих користувачем поколінь може призвести до невиправданого продовження роботи без суттєвого покращення результатів. Водночас, надто жорсткі умови зупинки можуть спричинити передчасне завершення роботи алгоритму і отримання не оптимального розв'язку.

В рамках даної дипломної роботи саме поєднання структури простору пошуку та обраних критеріїв зупинки визначає поведінку алгоритму щодо часової ефективності.

Тому в розроблюваній нами програмі передбачено можливість налаштувати: тип тестової функції, кількості поколінь, параметрів точності та механізму припинення роботи. Це забезпечить проведення систематичних експериментів та дозволить оцінити, як зміна цих параметрів впливає на час роботи генетичного алгоритму для задач різної складності.

1.3.4 Тестові функції оптимізації, їх класифікація та принципи вибору для оцінки ефективності генетичних алгоритмів

Тестові (бенчмарк) функції оптимізації є важливим інструментом дослідження властивостей та ефективності генетичних алгоритмів. Вони дозволяють оцінити здатність алгоритму знаходити глобальний мінімум у просторах різної складності, перевірити збіжність, стабільність та оцінити часові характеристики. Використання стандартизованих функцій дає можливість порівнювати результати різних дослідників та різних конфігурацій еволюційних алгоритмів [10.], [27].

Тестові функції характеризуються різноманітними структурними особливостями простору пошуку, такими як кількість локальних мінімумів, ступінь гладкості, розмірність та форма області оптимуму. Саме ці атрибути зумовлюють рівень складності задачі оптимізації й безпосередньо впливають на тривалість виконання генетичним алгоритмом.

Для всебічного аналізу часової ефективності алгоритмів є важливим використання тестових функцій з різним ступенем складності.

В контексті нашої дипломної роботи розглядаються три класичні функції оптимізації, які знайшли широке застосування в галузі еволюційних обчислень і які ми будемо використовувати для нашого дослідження: Sphere (унімодальна,

гладка, низька складність), Rosenbrock (гладка, із вузькою долиною, середня/висока складність), Rastrigin (багатомодальна, висока складність).

У наукових дослідженнях тестові функції класифікують за кількома ключовими ознаками: за кількістю мінімумів, за геометрією простору пошуку, за здатністю масштабуватися на більшу розмірність.

За кількістю мінімумів можемо класифікуваи як одномодальні (унімодальні) функції та багатомодульні.

Одномодальні (унімодальні) функції мають один глобальний мінімум і не містять локальних мінімумів. Характерні властивості:

- гладкий та простий рельєф простору пошуку;
- стабільна та швидка збіжність алгоритмів;
- застосовуються для аналізу базової продуктивності та швидкодії ГА.

Приклади:

Sphere, Sum of Squares, Zakharov, Ellipsoid.

Багатомодальні функції містять велику кількість локальних мінімумів і є значно складнішими для оптимізації. Потрібні для оцінки здатності алгоритму виходити з локальних пасток та підтримувати різноманітність популяції.

Приклади:

Rastrigin, Ackley, Griewank, Schwefel.

За геометрією простору пошуку можемо класифікувти як: гладкі функції з простою структурою та функції з вузькими долинами або складною топологією.

Гладкі функції з простою структуроюпідходять для оцінки базової поведінки алгоритму.

Приклади:

Sphere, Quadratic, Ellipsoid.

Функції з вузькими долинами або складною топологією не обов'язково містять багато локальних мінімумів, але характеризуються складною формою області мінімуму.

Приклади:

Rosenbrock, Powell.

Також функції класифікуються за здатністю масштабуватися на більшу розмірність: маштабовані та фіксовані функції.

Масштабовані функції

Можуть бути визначені для будь-якого числа змінних.

До них відносяться всі три функції, які ми використовуємо в нашому дослідженні.

Фіксовані функції

Використовуються рідше та під специфічні задачі.

Ми обирали функції для нашого дослідження з оцінки часової ефективності ГА використовуються три класичні тестові функції, які суттєво відрізняються за структурою простору пошуку.

Перша функція - це функція Sphere(1.7), , вона є легкого рівня складності.

$$f(\mathbf{x}) = f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i^2$$

(1.7)

ЇЇ властивості:

- гладка, опукла поверхня;
- один глобальний мінімум;
- відсутність локальних мінімумів;
- стабільна та швидка збіжність.

Завдяки своїй простоті функція Sphere дозволяє досліджувати “ідеальний” сценарій роботи генетичного алгоритму, де відсутні локальні мінімуми, а

збіжність залежить переважно від параметрів ГА та особливостей реалізації функції пристосованості [5].

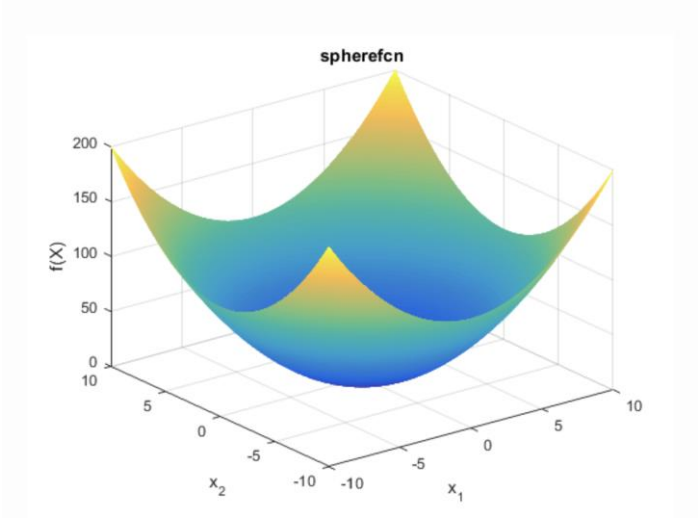


Рисунок 1.1 – Функція Sphere

На рисунку 1.1 зображено двовимірну поверхню тестової функції Sphere, яка є класичним прикладом унімодальної функції з єдиним глобальним мінімумом, розташованим у точці x=0x = 0x=0. Поверхня має симетричну параболічну форму без локальних мінімумів, що робить цю функцію відносно простою для оптимізації.

Друга функція - це функція Rastrigin (1.8) вона є з високою складністю та багатомодальна.

$$f(\mathbf{x}) = f(x_1, \ldots, x_n) = 10n + \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i)) \quad (1.8)$$

ЇЇ властивості:

- велика кількість локальних мінімумів;
- періодична, «хвиляста» структура;
- висока ймовірність передчасної збіжності.

Є класичним тестом для оцінювання здатності алгоритму досліджувати простір пошуку [27].

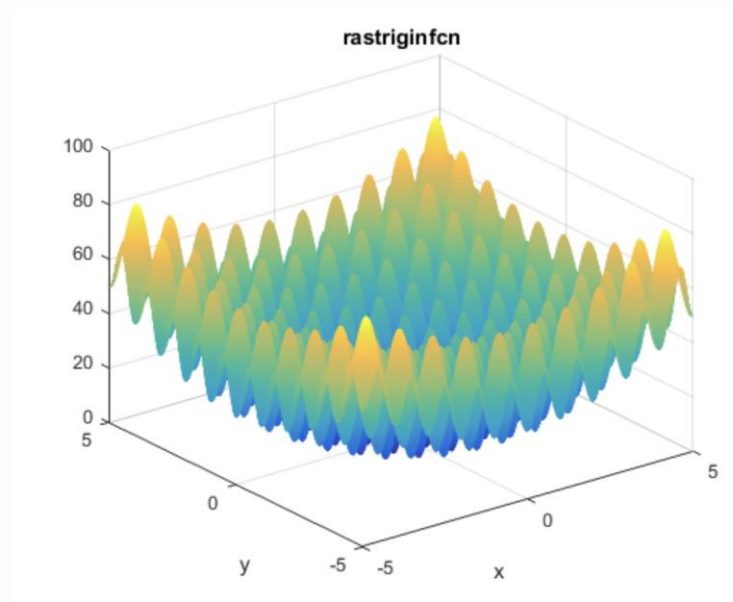


Рисунок 1.2 – Функція Rastrigin

На рисунку 1.2 представлено поверхню тестової функції Rastrigin, яка належить до класу мультимодальних функцій. Вона характеризується великою кількістю локальних мінімумів, рівномірно розташованих у просторі пошуку, що суттєво ускладнює процес оптимізації.

Третя функція - це функція Rosenbrock (1.9) вона є з середньої/високої складність.

$$f(\mathbf{x}) = f(x_1, \ldots, x_n) = \sum_{i=1}^{n-1} [b(x_{i+1} - x_i^2)^2 + (a - x_i)^2] \quad (1.9)$$

ЇЇ властивості:

- вузька вигнута долина;
- складна навігація до глобального мінімуму;
- чутливість до параметрів алгоритму.

Є еталонною функцією для оцінки стабільності та точності ГА. Функція Rastrigin дозволяє досліджувати здатність ГА виходити з локальних пасток, підтримувати різноманітність популяції та знаходити глобальний мінімум у складному багатовимірному просторі [9].

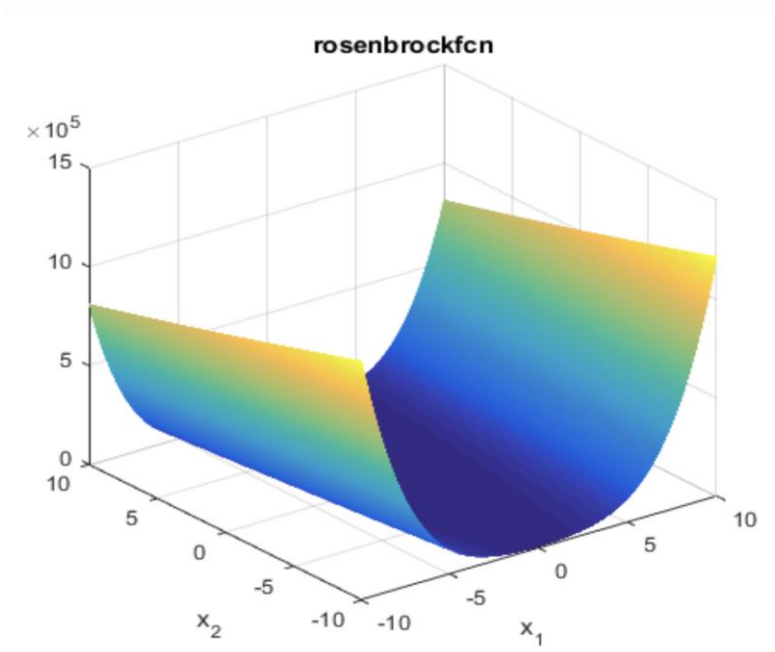


Рисунок 1.3 – Функція Rosenbrock

На рисунку 1.3 зображено поверхню тестової функції Rosenbrock, яка має вузьку вигнуту долину з глобальним мінімумом. Хоча функція є унімодальною, її геометрія створює значні труднощі для оптимізаційних алгоритмів через слабко виражений градієнт уздовж долини.

Таким чином, обраний набір функцій є репрезентативним і відповідає вимогам сучасних досліджень у галузі еволюційних обчислень.

1.4 Основні параметри генетичного алгоритму та їх вплив на ефективність

Параметри генетичного алгоритму визначають поведінку еволюційного пошуку та впливають як на якість розв’язків, так і на часову ефективність. У класичних роботах із генетичних алгоритмів підкреслюється, що продуктивність ГА визначається не лише окремими параметрами, а саме їх взаємодією [9], [13], [27], [7].

Розлянемо основні параметри ГА та аналіз їх впливу на швидкість алоритму: розмір популяції, кросинговер, мутація, кількість поколінь, селекція,

Розмір популяції визначає кількість кандидатних рішень у кожному поколінні.

Вплив:

- Мала популяція

- швидші покоління;
  − високий ризик передчасної збіжності;
  − недостатня різноманітність.

· Велика популяція

- краща різноманітність, більша стабільність;
  − збільшення часу одного покоління.

У літературі зазначено, що складні багатомодальні функції потребують ширшої популяції [7].

Імовірність кросинговеру, кросинговер (crossover rate)

Кросинговер відповідає за комбінування успішних будівельних блоків (building blocks), що є критичним для швидкої збіжності [9].

Занадто високі значення можуть руйнувати вже стабільні структури рішення, а надто низькі — уповільнювати еволюцію.

Для неперервної оптимізації, зокрема у дійсному (real-coded) ГА, рекомендуються арифметичні та лінійні типи кросинговеру [5].

Імовірність мутації. мутація (mutation rate)

Мутація забезпечує дослідження нових областей простору та запобігає застою алгоритму.

- Низька мутація → ризик потрапляння у локальні мінімуми.
- Висока → алгоритм перетворюється на випадковий пошук, зростає час збіжності.

Real-coded ГА використовують адаптивні та гаусівські мутації, які описані в [2] та [7].

Кількість поколінь (number of generations)

Один із ключових параметрів, що безпосередньо визначає час виконання алгоритму.

- Велика кількість поколінь підвищує шанси знайти якісний розв’язок.
- Надлишкова кількість поколінь збільшує час без покращення.

Цей параметр є базовим у дослідженнях часової ефективності [27].

Тип і параметри селекції

Основні методи: рулеткова, турнірна, рангова.

- Турнірна — забезпечує контроль селективного тиску [13].
- Рангова — стабілізує еволюцію та зменшує ефект домінування [7].
- Рулеткова — чутлива до масштабування fitness.

Тип селекції впливає радше на кількість потрібних поколінь, а не на час одного покоління.

Критерій зупинки

Є центральним параметром у задачах часової ефективності:

- максимальна кількість поколінь;
- порогове значення пристосованості;
- відсутність прогресу;
- часові обмеження.

Критерії зупинки визначають фактичну тривалість роботи алгоритму [9], [27].

Аналіз параметрів генетичного алгоритму демонструє, що їх вплив на ефективність пошуку не є незалежним. ГА функціонує як динамічна система, у якій зміна одного параметра змінює поведінку інших. Саме тому оптимальні конфігурації мають визначатися шляхом комплексного підходу, а не шляхом регулювання параметрів окремо [7].

Основні закономірності взаємодії параметрів можна підсумувати так:

- Розмір популяції впливає на різноманітність та може компенсувати низьку ймовірність мутації. Мала популяція прискорює окремі покоління, але підвищує ризик передчасної збіжності.
- Кросинговер із високою інтенсивністю потребує достатньої різноманітності у популяції; за її браком алгоритм швидко втрачає здатність до дослідження простору.
- Мутація відіграє особливо важливу роль у задачах зі складним ландшафтом, таких як Rastrigin або Rosenbrock, де велика кількість локальних мінімумів або вузькі долини вимагають додаткового механізму для виходу зі станів застою.
- Кількість поколінь визначає можливість алгоритму досягти прийнятного рішення, але надмірне збільшення цього параметра без корекції інших призводить до необґрунтованих часових витрат.

Таким чином, параметри генетичного алгоритму формують взаємопов’язану систему, в якій ефективність залежить від їх узгодженості. Для отримання стабільних та репрезентативних результатів необхідно розглядати параметричні комбінації, а не окремі параметри ізольовано. Це особливо важливо в контексті дослідження часової ефективності, де кінцевий час роботи залежить від збалансованості всіх елементів еволюційного процесу.

1.5 Часова складність генетичних алгоритмів

Часова складність генетичних алгоритмів визначає обчислювальні витрати, необхідні для отримання розв’язку певної якості, і є одним з ключових показників ефективності еволюційних методів. Генетичні алгоритми не мають строгих аналітичних меж часу роботи, оскільки їх поведінка залежить від випадкових факторів, структури простору пошуку та параметрів алгоритму. Проте в класичній літературі окреслено загальні принципи, які дають можливість оцінювати часові характеристики ГА та порівнювати їх за різних умов [9], [17], [7].

У найпростішому наближенні час роботи генетичного алгоритму можна подати як (1.10):

$$T=O(P\cdot G\cdot C_f)$$

(1.10)

де P — розмір популяції, G — кількість поколінь, C_f— час обчислення функції пристосованості. Саме етап оцінювання пристосованості є найбільш ресурсоємним і часто займає до 90% загальних обчислень. Операції селекції, кросинговеру та мутації є значно дешевшими, хоча їх параметри суттєво впливають на кількість необхідних поколінь і, відповідно, на загальний час роботи алгоритму.

Часова ефективність ГА значною мірою визначається параметрами алгоритму. Збільшення розміру популяції підвищує різноманітність та стабільність, але лінійно збільшує час кожного покоління. Висока чи низька інтенсивність мутації і кросинговеру може як прискорити, так і уповільнити збіжність залежно від складності задачі. Кількість поколінь та критерії зупинки визначають обсяг еволюційного процесу: надто велика кількість поколінь

призводить до зайвих витрат, а передчасна зупинка — до неякісного розв’язку. Тому критерії зупинки є одним з головних механізмів контролю часу виконання алгоритму.

Важливим чинником є також характер функції оптимізації. Для гладких та опуклих функцій на кшталт Sphere алгоритм збігається швидко, тоді як функції зі складним ландшафтом, такі як Rosenbrock чи Rastrigin, потребують більшої кількості поколінь, часто вимагають підвищених значень мутації та більшої різноманітності в популяції. Саме тому одна й та сама конфігурація алгоритму може демонструвати різний час роботи на функціях різної складності.

Через стохастичну природу ГА час виконання від запуску до запуску може різнитися. Для отримання достовірних оцінок необхідні багаторазові експерименти з фіксацією початкового стану генератора випадкових чисел та статистичним аналізом результатів. Саме цей підхід вважається стандартом у сучасних дослідженнях еволюційних обчислень [7].

Тому, часова складність генетичних алгоритмів залежить від комплексу чинників — параметрів алгоритму, особливостей функції оптимізації, операторів еволюції та критеріїв зупинки. Оскільки строгих теоретичних моделей для передбачення часу роботи ГА не існує, систематичні експериментальні дослідження залишаються основним інструментом аналізу. Це зумовлює актуальність розробки програмного засобу та проведення експериментів, спрямованих на визначення впливу параметрів алгоритму на часову ефективність, що й становить мету даної дипломної роботи.

1.6 Висновки до розділу 1

У першому розділі розглянуто теоретичні засади генетичних алгоритмів як стохастичних методів оптимізації, їх місце серед еволюційних обчислень та сфери застосування в сучасних технологіях. Проаналізовано основні компоненти ГА, включно з операторами селекції, кросинговеру та мутації, а також способи представлення рішень, що визначають поведінку алгоритму у процесі пошуку. Показано, що вибір операторів та типу кодування суттєво впливає на здатність алгоритму знаходити глобальні оптимуми та уникати локальних мінімумів у складних просторах пошуку.

Розглянуто роль функції пристосованості та критеріїв зупинки, які визначають структуру простору оптимізації та тривалість еволюційного процесу. Саме ці елементи формують динаміку генетичного алгоритму та визначають баланс між експлорацією та експлуатацією. Окрему увагу приділено тестовим функціям оптимізації як універсальному засобу оцінювання поведінки ГА. Стандартизовані бенчмарк-функції, такі як Sphere, Rosenbrock та Rastrigin, забезпечують можливість порівнювати різні конфігурації алгоритмів і створюють базу для експериментального аналізу їх часової ефективності.

У розділі митакож проаналізували часову складність генетичних алгоритмів та чинники, які впливають на обчислювальні витрати. Показано, що час виконання ГА визначається взаємодією параметрів популяції, операторів та критеріїв зупинки, а також характером задачі оптимізації. Часові характеристики є стохастичними за своєю природою, тому їх коректне вимірювання вимагає багаторазових запусків та статистичного аналізу.

Також ми окреслели низку частково вирішених та недостатньо досліджених питань, що стосуються моделювання часової ефективності ГА, вибору оптимальних параметрів для задач різної складності, а також браку уніфікованих інструментів для стандартизованого збору часових метрик. Ці прогалини підтверджують актуальність проведення систематичного експериментального дослідження та створення програмного засобу для оцінювання впливу параметрів ГА на час його роботи.

РОЗДІЛ 2. АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТА ПРОГРАМНИХ ЗАСОБІВ

2.1 Аналіз підходів до оцінювання ефективності генетичних алгоритмів

Оцінювання ефективності генетичних алгоритмів є важливою складовою досліджень у сфері еволюційних обчислень, оскільки ГА належать до стохастичних методів оптимізації, а їх поведінка суттєво залежить від параметрів, структури задачі та випадкових чинників. На відміну від детермінованих алгоритмів, ефективність ГА не може бути описана єдиною математичною формулою, тому у науковій практиці застосовуються комбіновані підходи, що включають аналіз якості розв'язків, статистику часу виконання, стабільність збіжності та поведінку алгоритму за різних конфігурацій [9], [13], [7].

Одним із базових критеріїв оцінки є якість отриманого розв'язку, тобто можливість алгоритму досягати глобального або близького до глобального оптимуму. Цей показник залежить від структури простору пошуку, наявності локальних мінімумів і здатності алгоритму підтримувати різноманітність популяції. Однак сам по собі критерій якості є недостатнім, оскільки не враховує обчислювальні витрати та стійкість алгоритму, тому в сучасних дослідженнях він використовується у поєднанні з іншими метриками.

Другим ключовим аспектом є часова ефективність, що визначає обсяг обчислень, необхідних для знаходження розв'язку заданої якості. Поширеним підходом до вимірювання часу є фіксація набору параметрів (розмір популяції, ймовірності операторів, кількість поколінь) та виконання багаторазових запусків алгоритму для отримання середнього часу, дисперсії та медіани. Така методика дозволяє врахувати стохастичну природу ГА та уникнути помилкових висновків, пов'язаних з одиничними випадками швидкої або повільної збіжності. Для складних функцій час виконання може різнитися в рази навіть при однакових налаштуваннях, тому середньостатистичний підхід є необхідним для коректної оцінки.

Важливою групою критеріїв є метрики збіжності, що характеризують динаміку пошуку. До них належать: швидкість зниження значення функції пристосованості, кількість поколінь до стабілізації популяції, момент, коли покращення розв'язку фактично припиняється. Ці показники дозволяють оцінити, наскільки ефективно ГА досліджує простір рішень і чи існують періоди застою, що є типовими для задач із багатьма локальними мінімумами (наприклад, Rastrigin).

Окреме місце займають статистичні підходи до оцінювання ефективності. Оскільки ГА є стохастичними, використання одиничних запусків не дає достовірної інформації. Тому застосовують методи математичної статистики: аналіз середнього та медіанного часу, стандартного відхилення, довірчих інтервалів та статистичних тестів (наприклад, тест Манна–Вітні або t-тест) для порівняння різних конфігурацій алгоритмів. Такий підхід рекомендується у більшості сучасних досліджень еволюційних методів [7].

У практиці оцінювання ефективності важливим є також порівняння параметричних конфігурацій, що дає змогу визначити оптимальні значення параметрів для різних типів функцій. Наприклад, для гладких та опуклих функцій достатньо невисокої мутації та помірного розміру популяції, тоді як для багатомодальних функцій можуть бути необхідними більші популяції, підвищена мутація та адаптивні оператори. Порівняння таких конфігурацій є стандартним методом встановлення закономірностей між параметрами та поведінкою алгоритмів.

Також використовуються графічні інструменти аналізу, такі як криві збіжності, коробкові діаграми і розподіли часу виконання. Вони дають змогу оцінити стабільність алгоритму, наявність аномальних запусків та рівномірність поведінки в різних умовах.

Таким чином, сучасні підходи до оцінювання ефективності генетичних алгоритмів охоплюють комплекс метрик та методів: якість розв’язків, часову ефективність, динаміку збіжності, статистичну стійкість та порівняння параметричних конфігурацій. Враховуючи стохастичну природу ГА, жоден окремий критерій не може повністю охарактеризувати роботу алгоритму, тому комбіновані методики є найбільш інформативними. Огляд цих підходів формує теоретичне підґрунтя для подальшого аналізу програмних інструментів та обґрунтування розроблення власної програмної системи для дослідження часової ефективності генетичних алгоритмів.

2.2 Аналіз та порівняння існуючих програмних засобів для дослідження генетичних алгоритмів

У межах даної роботи основна увага приділяється генетичним алгоритмам як одному з найбільш поширених класів еволюційних алгоритмів. Тому під час аналізу програмних засобів розглядаються як універсальні платформи для еволюційних обчислень, так і спеціалізовані бібліотеки, що підтримують реалізацію класичних генетичних алгоритмів, з акцентом на можливості експериментального аналізу часової ефективності.

Одним із найбільш відомих промислових інструментів для реалізації генетичних алгоритмів є MATLAB Global Optimization Toolbox, який містить вбудовану реалізацію ГА з широким набором параметрів: вибір операторів селекції, кросинговеру, мутації, адаптивні стратегії та різні критерії зупинки.

Перевагами цього середовища є:

- висока надійність реалізації;
- оптимізована швидкодія;
- зручні засоби візуалізації результатів.

Водночас даний інструмент має суттєві обмеження для цілей даного дослідження:

- закрите програмне середовище та ліцензійна залежність;
- обмежена гнучкість при детальному логуванні часових метрик;
- складність інтеграції з власними експериментальними сценаріями та автоматизованими серіями запусків.

У зв’язку з цим MATLAB доцільніше використовувати для прикладних задач оптимізації, а не для глибокого дослідження впливу параметрів ГА на часову ефективність [21].

ECJ є потужним академічним фреймворком для еволюційних обчислень, реалізованим мовою Java. Він підтримує широкий спектр еволюційних алгоритмів, зокрема класичні генетичні алгоритми, генетичне програмування та коеволюційні моделі[20].

Основними перевагами ECJ є:

- висока масштабованість;
- підтримка складних експериментальних конфігурацій;
- можливість детального налаштування операторів та параметрів.

Однак для даної дипломної роботи ECJ має низку недоліків:

- використання мови Java, тоді як дослідження орієнтоване на Python;
- значно вищий поріг входу та складність налаштування;
- надлишкова функціональність для задачі аналізу часової ефективності базових ГА.

Таким чином, ECJ є доцільним для масштабних наукових досліджень, але малопридатним для створення легкого експериментального інструменту, орієнтованого саме на часові метрики[20].

jMetal є відомим фреймворком для дослідження метаевристичних алгоритмів, що має реалізації як на Java, так і у вигляді Python-бібліотеки jMetalPy. Він підтримує різні алгоритми оптимізації, зокрема генетичні алгоритми, диференційну еволюцію та інші еволюційні методи.

Перевагами jMetalPy є:

- структурований підхід до побудови експериментів;
- підтримка багатокритеріальної оптимізації;
- наявність стандартних бенчмарк-функцій.

Разом з тим, для задачі даної роботи jMetalPy має певні обмеження:

- складна архітектура, орієнтована на універсальні метаевристики;
- відсутність фокусу саме на детальному вимірюванні часових характеристик;
- необхідність адаптації внутрішніх компонентів для реалізації власної методики збору статистики.

У результаті використання jMetalPy для задачі аналізу часової ефективності ГА є можливим, але менш зручним порівняно з розробленням власного компактного програмного інструменту.

Серед Python-бібліотек для генетичних алгоритмів найбільш поширеними є DEAP та PyGAD. Вони забезпечують базову реалізацію ГА та дозволяють швидко створювати прототипи.

Переваги:

- простота використання;
- активна спільнота;
- підтримка стандартних операторів ГА.

Недоліки:

- орієнтація переважно на пошук розв’язку, а не на дослідження часової ефективності;
- обмежені засоби логування;

- складність стандартизації експериментів для порівняння різних конфігурацій.

2.3 Висновки щодо необхідності створення власної програмної системи

Аналіз існуючих програмних засобів показав, що більшість доступних інструментів для реалізації генетичних алгоритмів орієнтовані насамперед на отримання оптимального розв'язку, а не на детальне дослідження часових характеристик роботи алгоритму. Навіть у потужних середовищах, таких як MATLAB Global Optimization Toolbox або академічні фреймворки типу ECJ та jMetal, вимірювання часу виконання зазвичай є другорядною функцією та не супроводжується зручними засобами збору, агрегації та візуалізації статистичних даних.

Крім того, більшість готових бібліотек:

- не забезпечують гнучкого логування результатів для багаторазових запусків;
- не дозволяють легко змінювати окремі параметри алгоритму з фіксацією їх впливу на час виконання;
- не орієнтовані на побудову порівняльних графіків для різних конфігурацій генетичного алгоритму.

З урахуванням стохастичної природи генетичних алгоритмів, коректне дослідження їх часової ефективності потребує:

- багаторазових незалежних запусків;
- збереження результатів кожного запуску;
- обчислення середніх значень та аналізу варіативності;
- наочного графічного подання отриманих даних.

Саме ці вимоги обумовили необхідність розроблення власного програмного інструменту мовою Python, спеціально орієнтованого на дослідження часової ефективності генетичних алгоритмів.

Розроблена програма повинна забезпечувати:

- реалізацію класичного генетичного алгоритму з можливістю налаштування основних параметрів (розмір популяції, кількість поколінь, ймовірності кросинговеру та мутації, параметри мутації);

- підтримку стандартних тестових функцій оптимізації різної складності (Sphere, Rastrigin, Rosenbrock);
- багаторазові запуски алгоритму для кожної конфігурації з метою отримання статистично коректних результатів;
- автоматичний вимір часу виконання алгоритму;
- збереження результатів експериментів у структурованому вигляді (CSV-файл) для подальшого аналізу.

Для аналізу часової ефективності генетичних алгоритмів у розробленій програмі передбачено збір та обробку таких основних показників:

- час виконання алгоритму для кожного запуску;
- середній час виконання для кожної тестової функції;
- кількість поколінь, використаних до досягнення критерію зупинки;
- варіативність часу виконання між окремими запусками.

На основі збережених даних програмний інструмент будує такі типи графіків:

- Стовпчикові діаграми середнього часу виконання
  Дають змогу порівняти часову ефективність генетичного алгоритму для задач різної складності;
- стовпчикові діаграми середньої кількості поколінь до зупинки
  Дозволяють оцінити швидкість збіжності алгоритму та її залежність від властивостей цільової функції;
- лінійні графіки часу виконання по окремих запусках
  Відображають стохастичну природу алгоритму, дозволяють виявити нестабільність або аномальні запуски.

Таке поєднання числових та графічних методів аналізу забезпечує комплексну оцінку часової ефективності генетичних алгоритмів і дозволяє встановити закономірності між параметрами алгоритму, складністю задачі та обчислювальними витратами.

Таким чином, розроблення власної програмної системи є обґрунтованим і необхідним для досягнення мети даної роботи. Створений інструмент дозволяє не лише реалізувати генетичний алгоритм, але й проводити систематичні експериментальні дослідження його часової ефективності, що є неможливим або суттєво ускладненим у межах готових програмних рішень.

Отримані результати та побудовані графіки створюють основу для подальшого аналізу впливу параметрів генетичного алгоритму на його швидкодію, що розглядається у наступному розділі роботи.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ ЧАСОВОЇ ЕФЕКТИВНОСТІ ГЕНЕТИЧНОГО АЛГОРИТМУ

3.1 Постановка задачі програмного дослідження

У межах даної роботи було розроблено програмний інструмент для експериментального дослідження часової ефективності генетичних алгоритмів. Основною метою програмної реалізації є вимірювання часу виконання

генетичного алгоритму за фіксованих параметрів для задач оптимізації різної складності.

Для експериментального дослідження було обрано класичні тестові функції Sphere, Rastrigin та Rosenbrock, які відрізняються формою простору пошуку та кількістю локальних мінімумів. Це дозволяє оцінити вплив складності задачі на часові характеристики алгоритму.

Програма реалізована мовою Python та здійснює багаторазові запуски генетичного алгоритму для кожної тестової функції. Результати кожного запуску зберігаються у вигляді таблиці, що містить інформацію про час виконання, кількість виконаних поколінь, значення функції пристосованості та параметри алгоритму. Такий підхід дозволяє врахувати стохастичну природу генетичних алгоритмів і провести статистично коректний аналіз отриманих даних.

3.2 Архітектура та структура програмного інструменту

Розроблений нами у межах даної дипломної роботи програмний інструмент призначений для експериментального дослідження часової ефективності генетичних алгоритмів у задачах оптимізації. Програма реалізована мовою Python та має модульну архітектуру, що забезпечує зручність розширення, повторного використання коду та проведення серій експериментів з різними параметрами алгоритму.

Загальна структура програмного проєкту побудована за принципом логічного розділення функціональних компонентів. Основні модулі програми відповідають за реалізацію генетичного алгоритму, визначення тестових функцій оптимізації, виконання експериментів та збереження результатів.

Програмний проєкт складається з таких основних файлів і директорій:

- main.py— головний файл програми, який ініціалізує експеримент, задає параметри та запускає серії обчислень;
- ga_core.py— модуль, що містить основну реалізацію генетичного алгоритму (еволюційний цикл, обчислення функції пристосованості, перевірка критеріїв зупинки);
- operators.py — модуль, у якому реалізовані генетичні оператори селекції, кросинговеру та мутації;
- fitness.py— модуль, що містить реалізації тестових функцій оптимізації (Sphere, Rastrigin, Rosenbrock);

• директорія results/ — використовується для збереження результатів експериментів у вигляді CSV-файлів.

• plots.py — використовується для побудови графіків і візуалізації даних.

Така організація дозволяє чітко відокремити логіку генетичного алгоритму від задач збору статистики та аналізу результатів.

Робота програмного інструменту відбувається за таким алгоритмом:

1. У головному файлі main.py задаються параметри експерименту, зокрема:

• розмір популяції;

• максимальна кількість поколінь;

• ймовірності кросинговеру та мутації;

• параметри мутації;

• розмірність задачі;

• кількість незалежних запусків алгоритму.

2. Для кожної тестової функції оптимізації виконується серія незалежних запусків генетичного алгоритму.

3. Під час кожного запуску:

• ініціалізується початкова популяція;

• виконується еволюційний цикл з використанням операторів селекції, кросинговеру та мутації;

• контролюється виконання критеріїв зупинки.

4. Для кожного запуску фіксуються ключові метрики:

• час виконання алгоритму;

• найкраще знайдене значення функції пристосованості;

• кількість виконаних поколінь;

• значення параметрів алгоритму.

5. Результати кожного запуску зберігаються у файлі ga_time_benchmark.csv для подальшого аналізу.

У програмному інструменті використано комбінований критерій зупинки генетичного алгоритму.

Алгоритм припиняє роботу у разі виконання хоча б однієї з умов:

• досягнення максимальної кількості поколінь;

• відсутність покращення найкращого значення функції пристосованості протягом заданої кількості поколінь.

Застосування такого підходу дозволяє зменшити зайві обчислення після досягнення збіжності та коректно аналізувати часову ефективність алгоритму для задач різної складності.

Збір та збереження експериментальних даних

Для забезпечення можливості статистичного аналізу програма зберігає результати кожного запуску у табличному вигляді CSV-файлу.

CSV-файл містить інформацію про тип тестової функції, номер запуску, час виконання, кількість виконаних поколінь та параметри алгоритму. Такий формат збереження даних є зручним для подальшої обробки результатів, побудови графіків та порівняльного аналізу часових характеристик генетичного алгоритму.

Далі ми використовуємо CSV-файл для побудови графіків та табличок у файлі plots.py.

Файл plots.py є допоміжним модулем програмного інструменту та використовується для автоматизованого аналізу і візуалізації результатів експериментів. Його основна мета — перетворити дані, отримані під час запусків генетичного алгоритму, у наочні графіки, які дозволяють оцінити часову ефективність, швидкість збіжності та стабільність роботи алгоритму на різних тестових функціях.

3.3 Побудова графіків та використання даних з CSV-файлу

Для візуального аналізу результатів експериментальних досліджень у роботі використовується файл з результатами у форматі CSV (ga_time_benchmark.csv). У цьому файлі зберігаються агреговані дані кожного запуску генетичного алгоритму для різних тестових функцій оптимізації.

CSV-файл містить такі основні поля:

- function — назва тестової функції оптимізації (Sphere, Rastrigin, Rosenbrock);
- run — номер окремого запуску алгоритму;
- time_sec — час виконання алгоритму в секундах;

· generations_done — кількість поколінь, виконаних до спрацювання критерію зупинки;

· best_fitness — найкраще значення функції пристосованості, знайдене під час запуску.

На основі цих даних будується кілька типів графіків, що дозволяють комплексно оцінити часову ефективність генетичного алгоритму.

Перший графік відображає середній час виконання генетичного алгоритму для кожної тестової функції.
Для його побудови використовується поле time_sec, значення якого усереднюється для кожної функції окремо.

Цей графік дозволяє:

- порівняти часову складність задач різної складності;
- наочно показати, як структура простору пошуку впливає на тривалість роботи алгоритму;
- підтвердити, що для складніших функцій середній час виконання є більшим.

Другий графік демонструє середню кількість поколінь, які були виконані до завершення роботи алгоритму для кожної функції.

Для побудови використовується поле generations_done, усереднене для кожної тестової функції.

Цей графік дозволяє:

- оцінити швидкість збіжності алгоритму;
- проаналізувати, для яких функцій алгоритм досягає критерію зупинки швидше;
- встановити зв’язок між кількістю поколінь та загальним часом виконання.

Окремо будуються графіки часу виконання для кожного запуску алгоритму по кожній тестовій функції.

Для цього використовуються:

· поле run — як вісь номерів запусків;

· поле time_sec — як відповідний час виконання.

3.4 Параметри експерименту та умови проведення дослідження

У межах даної дипломної роботи для дослідження часової ефективності генетичного алгоритму було розроблено та проведено серію програмних експериментів із використанням різних конфігурацій параметрів. Основною метою експериментів є оцінка впливу окремих параметрів генетичного алгоритму на час його виконання при розв’язанні задач оптимізації різної складності.

Для експериментального дослідження було обрано три класичні тестові функції оптимізації — Sphere, Rastrigin та Rosenbrock, які характеризуються різною структурою простору пошуку та рівнем складності. Це дозволяє оцінити поведінку алгоритму як у відносно простих умовах, так і в умовах складного, багатомодального або вузького простору пошуку.

З метою отримання статистично обґрунтованих результатів кожна конфігурація параметрів генетичного алгоритму запускалася 10 незалежних разів для кожної тестової функції. Подальший аналіз базувався на середніх значеннях часу виконання алгоритму та кількості поколінь до зупинки, що дозволяє зменшити вплив стохастичної природи генетичних алгоритмів на результати дослідження.

У всіх експериментах використовувалися такі основні параметри генетичного алгоритму:

- розмір популяції;
- максимальна кількість поколінь;
- ймовірність кросинговеру;
- ймовірність мутації;
- параметр інтенсивності мутації (σ);
- розмірність задачі;
- критерій зупинки алгоритму;
- кількість незалежних запусків.

Критерієм зупинки алгоритму у всіх конфігураціях було досягнення максимальної кількості поколінь або виконання внутрішньої умови завершення, реалізованої в програмі.

Для перевірки гіпотези дослідження було сформовано шість експериментальних конфігурацій, які умовно поділяються на три групи:

- базова (контрольна) конфігурація;
- конфігурації зі зміною параметрів, що безпосередньо впливають на обсяг обчислень;
- конфігурації зі зміною параметрів генетичних операторів.

Зараз детально розберимо 6 конфігурацій гетететичного алгоритму для нашаго запуску.

Конфігурація 1 — базова (контрольна)

Дана конфігурація використовується як еталонна та слугує основою для порівняння з іншими експериментальними режимами. У ній задаються стандартні значення параметрів генетичного алгоритму, які забезпечують стабільну роботу та прийнятну якість розв’язків.

У межах цієї конфігурації:

- розмір популяції встановлюється на середньому рівні;
- кількість поколінь є достатньою для збіжності алгоритму;
- ймовірності кросинговеру та мутації відповідають типовим рекомендованим значенням;
- розмірність задачі є фіксованою.

Отримані в цій конфігурації результати використовуються як базові для подальшого порівняльного аналізу.

Данні для першої конфігурації:

- розмір популяції: 50 особин
- максимальна кількість поколінь: 150
- розмірність задачі: 10
- ймовірність кросинговеру: 0.8
- ймовірність мутації: 0.2
- параметр мутації (σ): 0.1
- критерій зупинки: досягнення максимальної кількості поколінь
- кількість запусків: 10

Конфігурація 2 — збільшення кількості поколінь

У другій конфігурації змінюється лише максимальна кількість поколінь, тоді як усі інші параметри залишаються такими ж, як у базовій конфігурації.

Метою даного експерименту є оцінка впливу кількості поколінь на часову ефективність алгоритму. Оскільки кожне покоління передбачає виконання повного циклу обчислення функції пристосованості для всієї популяції, очікується, що збільшення кількості поколінь призведе до істотного зростання часу виконання алгоритму.

Данні для другої конфігурації:

- розмір популяції: 50
- максимальна кількість поколінь: 300
- розмірність задачі: 10
- ймовірність кросинговеру: 0.8
- ймовірність мутації: 0.05
- параметр мутації (σ): 0.1
- критерій зупинки: досягнення максимальної кількості поколінь
- кількість запусків: 10

Конфігурація 3 — збільшення розміру популяції

У третій конфігурації змінюється розмір популяції, тоді як кількість поколінь та інші параметри залишаються фіксованими.

Дана конфігурація дозволяє проаналізувати вплив кількості особин у кожному поколінні на час виконання генетичного алгоритму. Зі збільшенням розміру популяції зростає кількість обчислень функції пристосованості в межах кожного покоління, що безпосередньо впливає на загальні часові витрати.

Дані для третьої конфігурації:

- розмір популяції: 150
- максимальна кількість поколінь: 150
- розмірність задачі: 10
- ймовірність кросинговеру: 0.8
- ймовірність мутації: 0.05
- параметр мутації (σ): 0.1

- критерій зупинки: досягнення максимальної кількості поколінь
- кількість запусків: 10

Конфігурація 4 — збільшення розмірності задачі

У четвертій конфігурації змінюється розмірність задачі оптимізації, тобто кількість змінних, які входять до вектора рішення. Інші параметри залишаються такими ж, як у базовій конфігурації.

Зі збільшенням розмірності задачі ускладнюється обчислення значення функції пристосованості, що призводить до збільшення часу виконання алгоритму навіть за незмінної кількості поколінь і розміру популяції.

Дані для четвертої конфігурації:

- розмір популяції: 50
- максимальна кількість поколінь: 150
- розмірність задачі: 30
- ймовірність кросинговеру: 0.8
- ймовірність мутації: 0.05
- параметр мутації (σ): 0.1
- критерій зупинки: досягнення максимальної кількості поколінь
- кількість запусків: 10

Конфігурація 5 — зміна ймовірності кросинговеру

У п’ятій конфігурації змінюється лише ймовірність кросинговеру, тоді як усі інші параметри залишаються на базовому рівні.

Оскільки зміна ймовірності кросинговеру не впливає безпосередньо на кількість обчислень функції пристосованості, очікується, що її вплив на загальний час виконання буде менш вираженим порівняно з параметрами, що визначають обсяг обчислень.

Дані для п'ятої конфігурації:

- розмір популяції: 50
- максимальна кількість поколінь: 150
- розмірність задачі: 10
- ймовірність кросинговеру: 0.6
- ймовірність мутації: 0.05

· параметр мутації (σ): 0.1

• критерій зупинки: досягнення максимальної кількості поколінь

· кількість запусків: 10

Конфігурація 6 — зміна параметрів мутації

У шостій конфігурації змінюється ймовірність мутації, параметр інтенсивності мутації (σ), тоді як інші параметри залишаються незмінними.

Ця конфігурація дозволяє окремо проаналізувати вплив мутаційного оператора на часову ефективність генетичного алгоритму. Очікується, що зміна параметрів мутації матиме незначний вплив на час виконання, однак може впливати на стабільність збіжності та варіативність отриманих результатів.

Дані для шостої конфігурації:

· розмір популяції: 50

· максимальна кількість поколінь: 150

· розмірність задачі: 10

· ймовірність кросинговеру: 0.8

· ймовірність мутації: 0.15

· параметр мутації (σ): 0.2

• критерій зупинки: досягнення максимальної кількості поколінь

· кількість запусків: 10

Таким чином, у межах дослідження було проведено серію експериментів із шістьма різними конфігураціями параметрів генетичного алгоритму. Усі конфігурації, за винятком змінюваного параметра, зберігають однакові значення інших параметрів, що дозволяє ізольовано оцінити вплив кожного з них на часову ефективність генетичного алгоритму. Для кожної конфігурації та кожної тестової функції алгоритм запускався 10 разів, після чого здійснювався аналіз середніх значень часу виконання та кількості поколінь до зупинки.

3.5 Аналіз отриманих результатів

3.5.1 Аналіз базової конфігурації

У наведеному експерименті з першою конфігурацією використовувалася базова конфігурація параметрів генетичного алгоритму:

· розмірність задачі: dim = 10;

- розмір популяції: pop_size = 50;
- максимальна кількість поколінь: generations = 150;
- ймовірність кросинговеру: crossover_prob = 0.8;
- ймовірність мутації: mutation_prob = 0.05;
- параметр мутації: mutation_sigma = 0.1;
- критерій зупинки за стагнацією: stagnation_k = 30;
- турнірна селекція з параметром tournament_k = 3;
- межі простору пошуку: [-5.0; 5.0].

На рис. 3.1 наведено фрагмент CSV-файлу першої конфігурації, який формується програмним інструментом після виконання серії експериментальних запусків та використовується для подальшого статистичного аналізу й побудови графіків.

ga_time_analizer > results > ga_time_benchmark.csv

```
function,run,time_sec,best_fitness,generations_done,pop_size,generations,crossover_prob,mutation_prob,mutation_sigma,dim
Sphere,1,0.02731439983472228,0.004601050014642415,150,50,150,0.8,0.2,0.2,10
Sphere,2,0.027569199912250042,0.003798272164395589,150,50,150,0.8,0.2,0.2,10
Sphere,3,0.02777269994840026,0.002134460814283883,150,50,150,0.8,0.2,0.2,10
Sphere,4,0.016718999948352575,0.006006279420120179,92,50,150,0.8,0.2,0.2,10
Sphere,5,0.021143800113350153,0.002763541053384954,111,50,150,0.8,0.2,0.2,10
Sphere,6,0.02816330036148429,0.0032406976308432113,150,50,150,0.8,0.2,0.2,10
Sphere,7,0.02779550012201070 8,0.0023360259790110314,150,50,150,0.8,0.2,0.2,10
Sphere,8,0.024570500012487173,0.0016639133994253762,133,50,150,0.8,0.2,0.2,10
Sphere,9,0.019983000122010708,0.000920541798783021,108,50,150,0.8,0.2,0.2,10
Sphere,10,0.024832400027662516,0.0031641512911130326,134,50,150,0.8,0.2,0.2,10
Rastrigin,1,0.020569599699229002,20.82782495238733,96,50,150,0.8,0.2,0.2,10
Rastrigin,2,0.033346999902278185,13.406200614924089,150,50,150,0.8,0.2,0.2,10
Rastrigin,3,0.03283789986744523,12.542267281994697,150,50,150,0.8,0.2,0.2,10
Rastrigin,4,0.03216399997472763,13.286408700841406,150,50,150,0.8,0.2,0.2,10
Rastrigin,5,0.02100119972601533,4.739426976783946,96,50,150,0.8,0.2,0.2,10
Rastrigin,6,0.0325166997499764,6.288488934357645,149,50,150,0.8,0.2,0.2,10
Rastrigin,7,0.025509099941700697,7.6025583415847535,117,50,150,0.8,0.2,0.2,10
Rastrigin,8,0.032232099678367376,4.505333316488361,150,50,150,0.8,0.2,0.2,10
Rastrigin,9,0.03206030000001192,15.539853745040602,149,50,150,0.8,0.2,0.2,10
Rastrigin,10,0.024970300029963255,14.753197325279075,110,50,150,0.8,0.2,0.2,10
Rosenbrock,1,0.03832109970971942,8.935803924051475,150,50,150,0.8,0.2,0.2,10
Rosenbrock,2,0.03591209976002574,6.975660844994993,141,50,150,0.8,0.2,0.2,10
Rosenbrock,3,0.03827130002900958,8.843617326162965,150,50,150,0.8,0.2,0.2,10
Rosenbrock,4,0.03074510022997856,8.587328539562332,117,50,150,0.8,0.2,0.2,10
Rosenbrock,5,0.0385644999332726,9.562264023882738,150,50,150,0.8,0.2,0.2,10
Rosenbrock,6,0.038954499643296,8.648026101602213,150,50,150,0.8,0.2,0.2,10
Rosenbrock,7,0.04100919980555773,7.891319470489832,150,50,150,0.8,0.2,0.2,10
Rosenbrock,8,0.039713299833238125,8.7879240209254,150,50,150,0.8,0.2,0.2,10
Rosenbrock,9,0.032708399929106236,8.474578150304232,123,50,150,0.8,0.2,0.2,10
Rosenbrock,10,0.039243099745363,8.880044584810914,150,50,150,0.8,0.2,0.2,10
```

Рисунок 3.1 – Фрагмент CSV-файлу з результатами експериментальних запусків генетичного алгоритму

На графіку зображено середній час виконання алгоритму для кожної тестової функції, обчислений як середнє значення часу за 10 запусків.

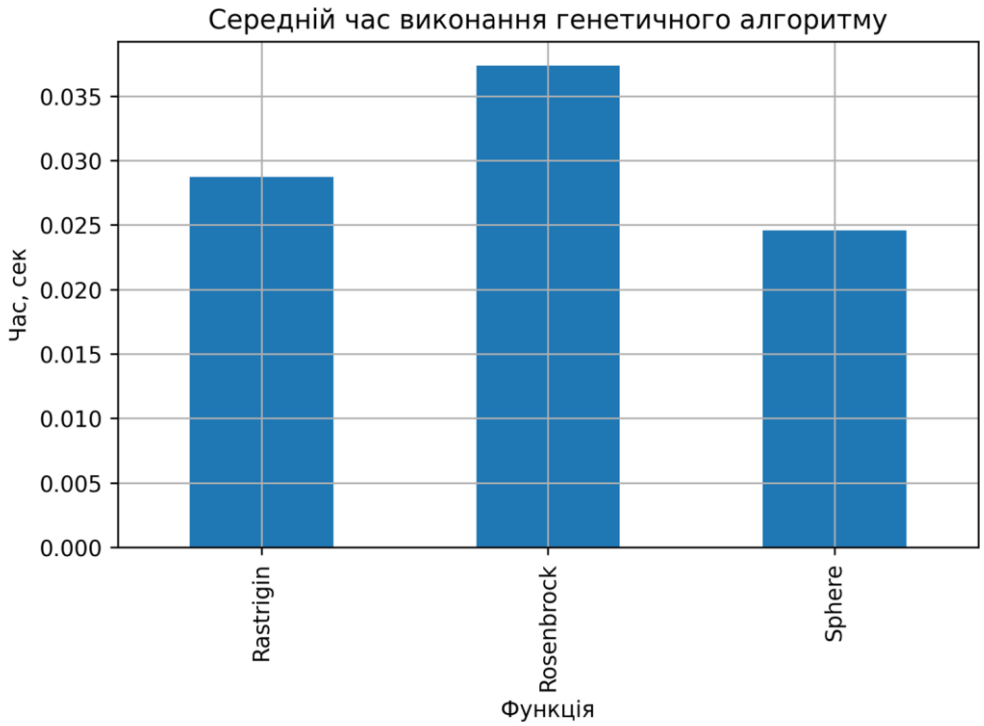


Рисунок 3.2 – Середній час виконання генетичного алгоритму

На графіку 3.2 зображено середній час виконання генетичного алгоритму для кожної тестової функції, обчислений як середнє значення часу за 10 незалежних запусків:

- найменший середній час виконання спостерігається для функції Sphere і становить приблизно 0,025 с, що пояснюється її простим, гладким та унімодальним простором пошуку;
- для функції Rastrigin середній час виконання є більшим і становить близько 0,029 с, що пов’язано з наявністю великої кількості локальних мінімумів, які ускладнюють процес пошуку глобального оптимуму;
- найбільший середній час виконання зафіксовано для функції Rosenbrock — приблизно 0,038 с, що обумовлено складною геометрією функції та вузькою вигнутою долиною глобального мінімуму, яка потребує більшої кількості ітерацій для ефективної збіжності.

Рисунок 3.3 – Середня кількість поколінь до зупинки алгоритму

Графік 3.3 демонструє середню кількість поколінь, які були виконані генетичним алгоритмом до досягнення критерію зупинки (стагнація або досягнення максимального числа поколінь):

- Sphere — ≈ 133 покоління
- Rastrigin — ≈ 132 покоління
- Rosenbrock — ≈ 143 покоління

Отже, для функцій Sphere та Rastrigin алгоритм у середньому завершує роботу дещо раніше за встановлене обмеження у 150 поколінь.

Для функції Rosenbrock середня кількість поколінь є найбільшою та близькою до верхньої межі, що свідчить про складніший ландшафт функції та повільнішу збіжність.

Рисунок 3.4 –Час виконання по запусках: Sphere

Графік 3.4 відображає час виконання алгоритму для кожного з 10 запусків на функції Sphere:

- мінімальний час: ≈ 0,017 с
- максимальний час: ≈ 0,028 с
- середній час: ≈ 0,025 с

Час виконання є стабільним і відносно малим. Невелика варіація між окремими запусками пояснюється випадковістю ініціалізації популяції. Функція Sphere добре піддається оптимізації генетичним алгоритмом навіть за кількості поколінь 150.

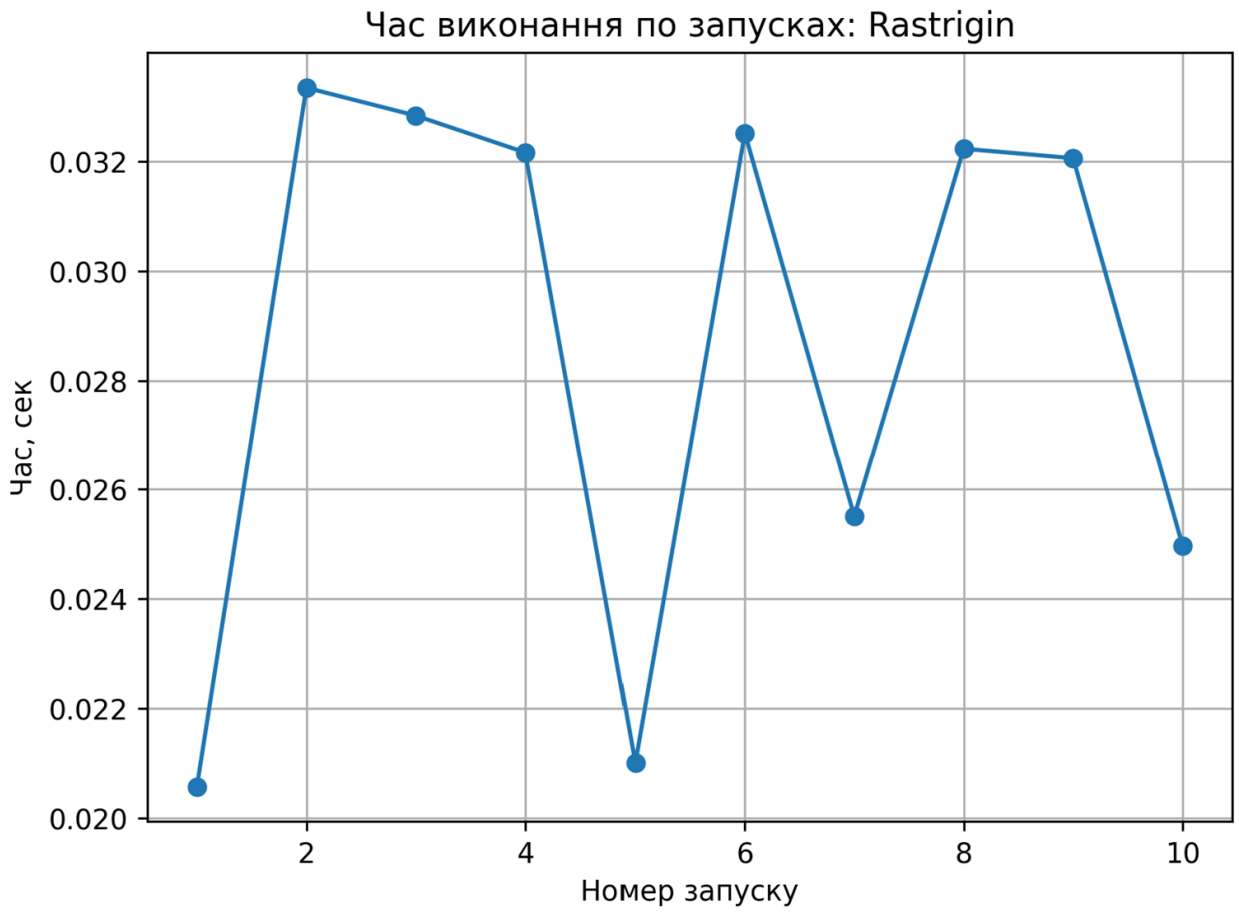


Рисунок 3.5 –Час виконання по запусках: Rastrigin

Графік 3.5 показує час виконання алгоритму для кожного запуску на мультимодальній функції Rastrigin:

- мінімальний час: ≈ 0,021 с
- максимальний час: ≈ 0,033 с
- середній час: ≈ 0,029 с

У порівнянні з функцією Sphere, час виконання є більшим та менш стабільним. Це пояснюється великою кількістю локальних мінімумів функції Rastrigin.

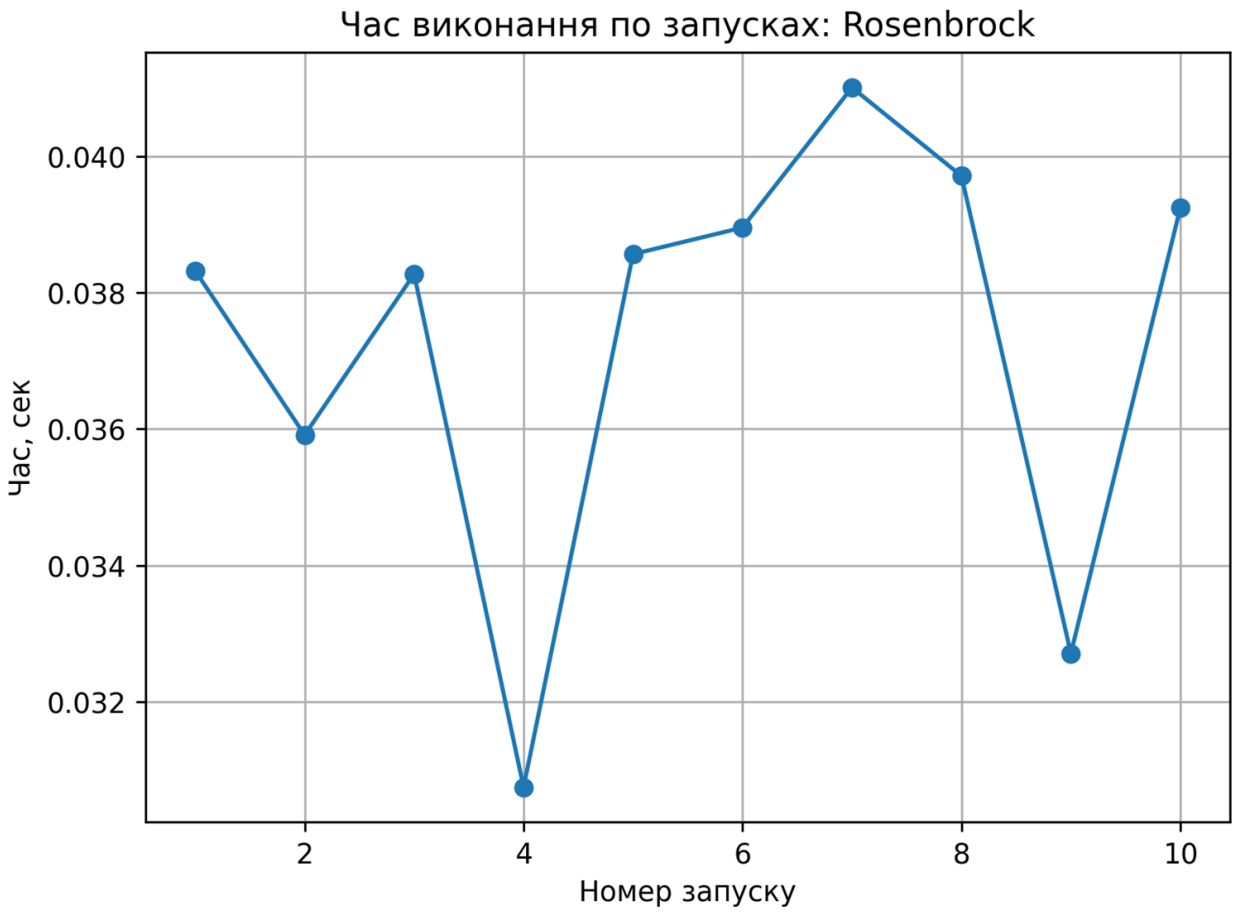


Рисунок 3.6 –Час виконання по запусках: Rosenbrock

Графік 3.6 ілюструє час виконання алгоритму для кожного запуску на функції Rosenbrock:

- мінімальний час: ≈ 0,031 с
- максимальний час: ≈ 0,041 с
- середній час: ≈ 0,038 с

Для цієї функції спостерігається найбільший середній час виконання та підвищена варіативність між запусками. У більшості експериментів алгоритм використовував майже максимальну кількість дозволених поколінь, що підтверджує складність функції Rosenbrock.

У межах першої базової конфігурації генетичного алгоритму експерименти проводилися за фіксованих параметрів, що забезпечувало коректне порівняння часової ефективності для різних тестових функцій. Було використано такі значення параметрів алгоритму:

- розмірність задачі: dim = 10;

- розмір популяції: pop_size = 50;
- максимальна кількість поколінь: generations = 150;
- ймовірність кросовінгу: crossover_prob = 0.8;
- ймовірність мутації: mutation_prob = 0.2;
- параметр мутації: mutation_sigma = 0.2;
- критерій зупинки за стагнацією;
- межі області пошуку: (-5.0, 5.0).

Для кожної тестової функції було виконано 10 незалежних запусків, після чого обчислювалися середні значення часу виконання та кількості використаних поколінь.

Функція Sphere продемонструвала найвищу часову ефективність: середня кількість поколінь до зупинки становила близько 133, а середній час виконання — 0,025 секунди, що свідчить про швидку та стабільну збіжність алгоритму.

Для функції Rastrigin було зафіксовано зростання обчислювальних витрат: середній час виконання склав 0,029 секунди, а розкид значень між окремими запусками був більшим, що пов’язано з мультимодальністю функції.

Найскладнішою виявилася функція Rosenbrock: середня кількість поколінь до зупинки склала приблизно 143, а середній час виконання — 0,038 секунди, що майже на 50% більше, ніж для функції Sphere.

Отримані результати підтверджують, що навіть за обмеження у 150 поколінь часова ефективність генетичного алгоритму суттєво залежить від складності цільової функції: зі зростанням складності збільшується як кількість поколінь до зупинки, так і загальний час виконання.

3.5.2 Аналіз другої конфігурації

У наступному експерименті з другою конфігурацією використовується наступна конфігурація параметрів генетичного алгоритму:

- розмірність задачі: dim = 10;
- розмір популяції: pop_size = 50;
- максимальна кількість поколінь: generations = 300;

· ймовірність кросинговеру: crossover_prob = 0.8;

· ймовірність мутації: mutation_prob = 0.05;

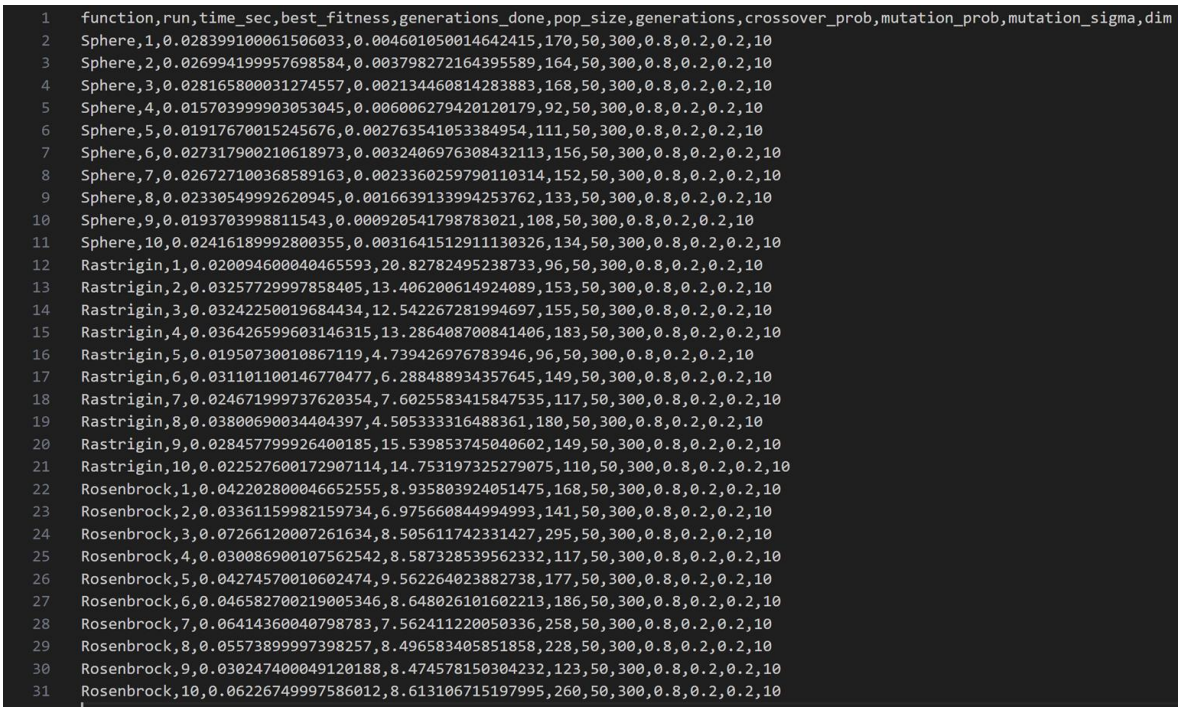
· параметр мутації: mutation_sigma = 0.1;

· критерій зупинки за стагнацією: stagnation_k = 30;

· турнірна селекція з параметром tournament_k = 3;

· межі простору пошуку: [-5.0; 5.0].

На рис. 3.7 наведено фрагмент CSV-файлу першої конфігурації, який формується програмним інструментом після виконання серії експериментальних запусків та використовується для подальшого статистичного аналізу й побудови графіків.



```
function,run,time_sec,best_fitness,generations_done,pop_size,generations,crossover_prob,mutation_prob,mutation_sigma,dim
Sphere,1,0.028399100061506033,0.004601050014642415,170,50,300,0.8,0.2,0.2,10
Sphere,2,0.026994199957698584,0.003798272164395589,164,50,300,0.8,0.2,0.2,10
Sphere,3,0.028165800031274557,0.002134460814283883,168,50,300,0.8,0.2,0.2,10
Sphere,4,0.015703999903053045,0.006006279420120179,92,50,300,0.8,0.2,0.2,10
Sphere,5,0.01917670015245676,0.002763541053384954,111,50,300,0.8,0.2,0.2,10
Sphere,6,0.027317900210618973,0.0032406976308432113,156,50,300,0.8,0.2,0.2,10
Sphere,7,0.026727100368589163,0.0023360259790110314,152,50,300,0.8,0.2,0.2,10
Sphere,8,0.02330549992620945,0.0016639133994253762,133,50,300,0.8,0.2,0.2,10
Sphere,9,0.0193703998811543,0.000920541798783021,108,50,300,0.8,0.2,0.2,10
Sphere,10,0.02416189992800355,0.0031641512911130326,134,50,300,0.8,0.2,0.2,10
Rastrigin,1,0.020094600040465593,20.82782495238733,96,50,300,0.8,0.2,0.2,10
Rastrigin,2,0.03257729997858405,13.406200614924089,153,50,300,0.8,0.2,0.2,10
Rastrigin,3,0.03242250019684434,12.542267281994697,155,50,300,0.8,0.2,0.2,10
Rastrigin,4,0.036426599603146315,13.286408700841406,183,50,300,0.8,0.2,0.2,10
Rastrigin,5,0.01950730010867119,4.739426976783946,96,50,300,0.8,0.2,0.2,10
Rastrigin,6,0.031101100146770477,6.288488934357645,149,50,300,0.8,0.2,0.2,10
Rastrigin,7,0.024671999737620354,7.6025583415847535,117,50,300,0.8,0.2,0.2,10
Rastrigin,8,0.03800690034404397,4.505333316488361,180,50,300,0.8,0.2,0.2,10
Rastrigin,9,0.028457799926400185,15.539853745040602,149,50,300,0.8,0.2,0.2,10
Rastrigin,10,0.022527600172907114,14.753197325279075,110,50,300,0.8,0.2,0.2,10
Rosenbrock,1,0.042202800046652555,8.935803924051475,168,50,300,0.8,0.2,0.2,10
Rosenbrock,2,0.03361159982159734,6.975660844994993,141,50,300,0.8,0.2,0.2,10
Rosenbrock,3,0.07266120007261634,8.505611742331427,295,50,300,0.8,0.2,0.2,10
Rosenbrock,4,0.030086900107562542,8.587328539562332,117,50,300,0.8,0.2,0.2,10
Rosenbrock,5,0.04274570010602474,9.562264023882738,177,50,300,0.8,0.2,0.2,10
Rosenbrock,6,0.046582700219005346,8.648026101602213,186,50,300,0.8,0.2,0.2,10
Rosenbrock,7,0.06414360040798783,7.562411220050336,258,50,300,0.8,0.2,0.2,10
Rosenbrock,8,0.05573899997398257,8.496583405851858,228,50,300,0.8,0.2,0.2,10
Rosenbrock,9,0.030247400049120188,8.474578150304232,123,50,300,0.8,0.2,0.2,10
Rosenbrock,10,0.06226749997586012,8.613106715197995,260,50,300,0.8,0.2,0.2,10
```

Рисунок 3.7 – Фрагмент CSV-файлу з результатами експериментальних запусків генетичного алгоритму

На графіку зображено середній час виконання алгоритму для кожної тестової функції, обчислений як середнє значення часу за 10 запусків.

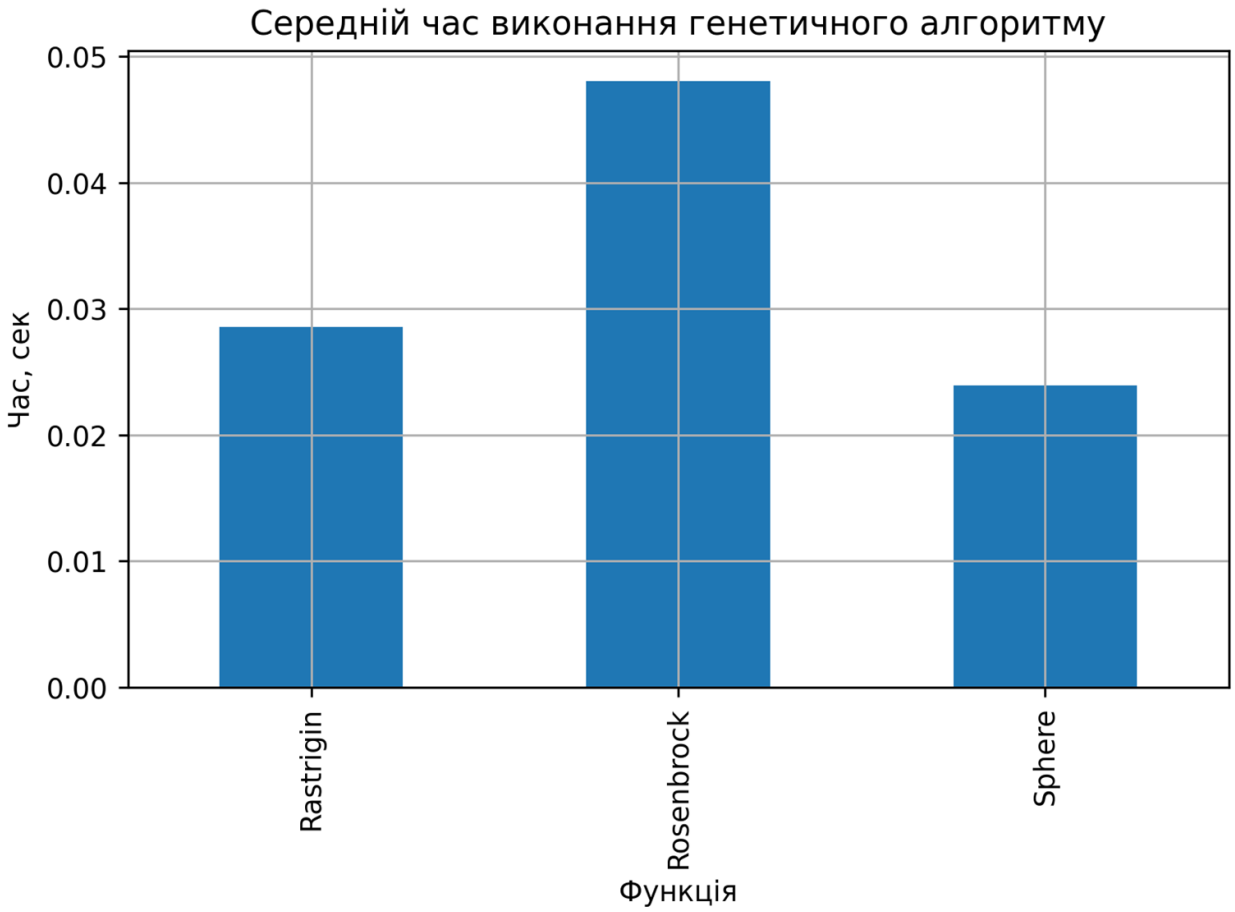


Рисунок 3.8 – Середній час виконання генетичного алгоритму

З графіка 3.8 видно, що:

- найменший середній час виконання спостерігається для функції Sphere і становить приблизно 0,024 с, що пояснюється її простим, гладким та унімодальним простором пошуку;
- для функції Rastrigin середній час виконання є більшим і становить близько 0,029 с, що пов’язано з наявністю великої кількості локальних мінімумів, які ускладнюють процес пошуку глобального оптимуму;
- найбільший середній час виконання зафіксовано для функції Rosenbrock — приблизно 0,048 с, що обумовлено складною геометрією функції та вузькою вигнутою долиною глобального мінімуму, яка потребує більшої кількості поколінь для ефективної збіжності.

Графік 3.5.2.2 демонструє середню кількість поколінь, які були виконані алгоритмом до досягнення критерію зупинки для кожної тестової функції.

Рисунок 3.9 –Середня кількість поколінь до зупинки алгоритму

Рисунок 3.9 показав:

- Sphere — ≈ 139 поколінь
- Rastrigin — ≈ 139 поколінь
- Rosenbrock — ≈ 195 поколінь

Тобто для функцій Sphere та Rastrigin алгоритм у середньому зупиняється значно раніше за максимальне обмеження у 300 поколінь. Для Rosenbrock середнє значення близьке до верхньої межі, що свідчить про складніший ландшафт функції та повільнішу збіжність. Це узгоджується з теоретичними властивостями Rosenbrock-функції, яка має вузьку долину та вимагає більшого часу пошуку.

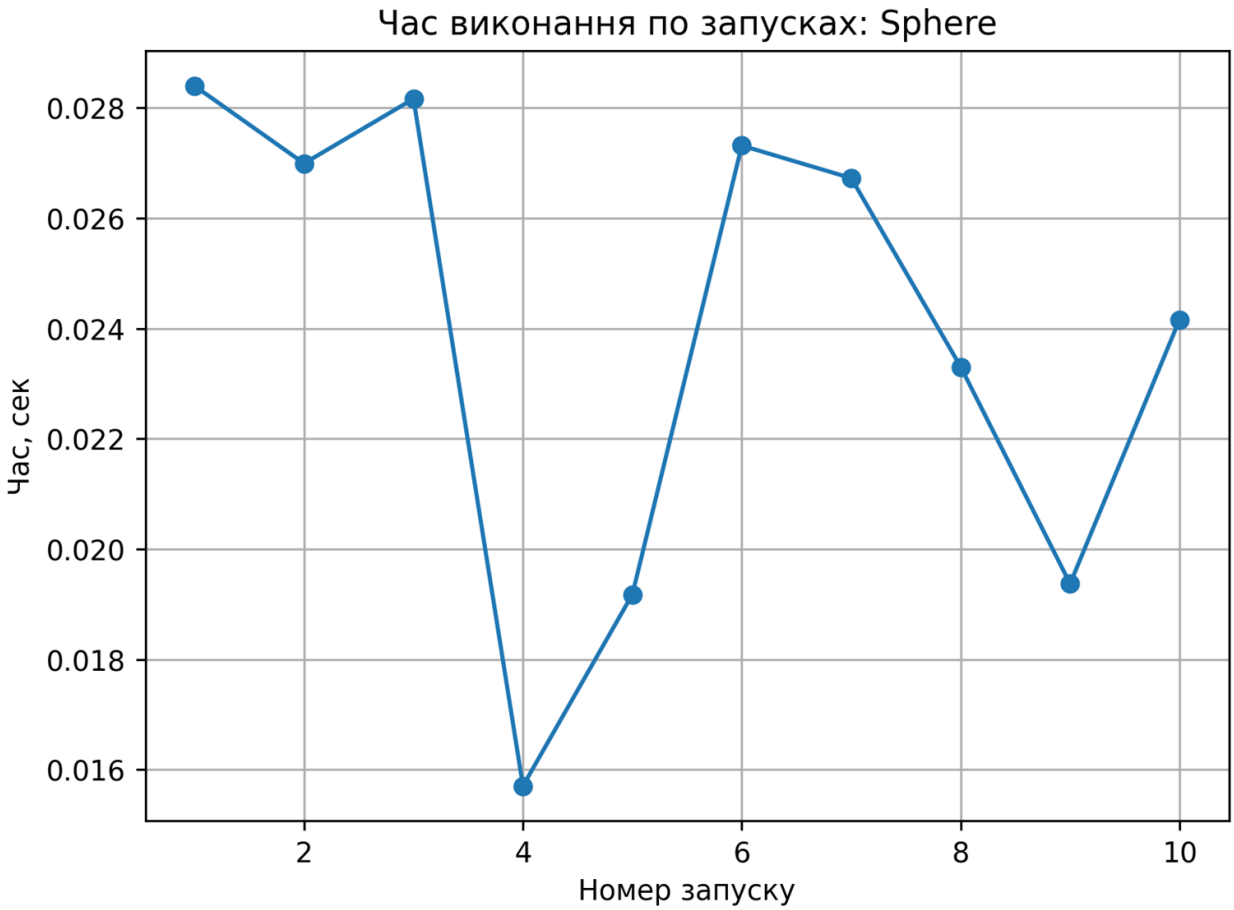


Рисунок 3.10 –Час виконання по запусках: Sphere

Рисунок 3.10 показує час виконання алгоритму для кожного з 10 запусків на функції Sphere:

- Мінімальний час: ≈ 0.016 с
- Максимальний час: ≈ 0.028 с
- Середній час: ≈ 0.024 сек

Час виконання є стабільним і невеликим.

Рисунок 3.11 показує час виконання алгоритму для кожного запуску на мультимодальній функції Rastrigin:

- Мінімальний час: ≈ 0.020 с
- Максимальний час: ≈ 0.038 с
- Середній час: ≈ 0.029 сек

У порівнянні зі Sphere, час виконання більший.

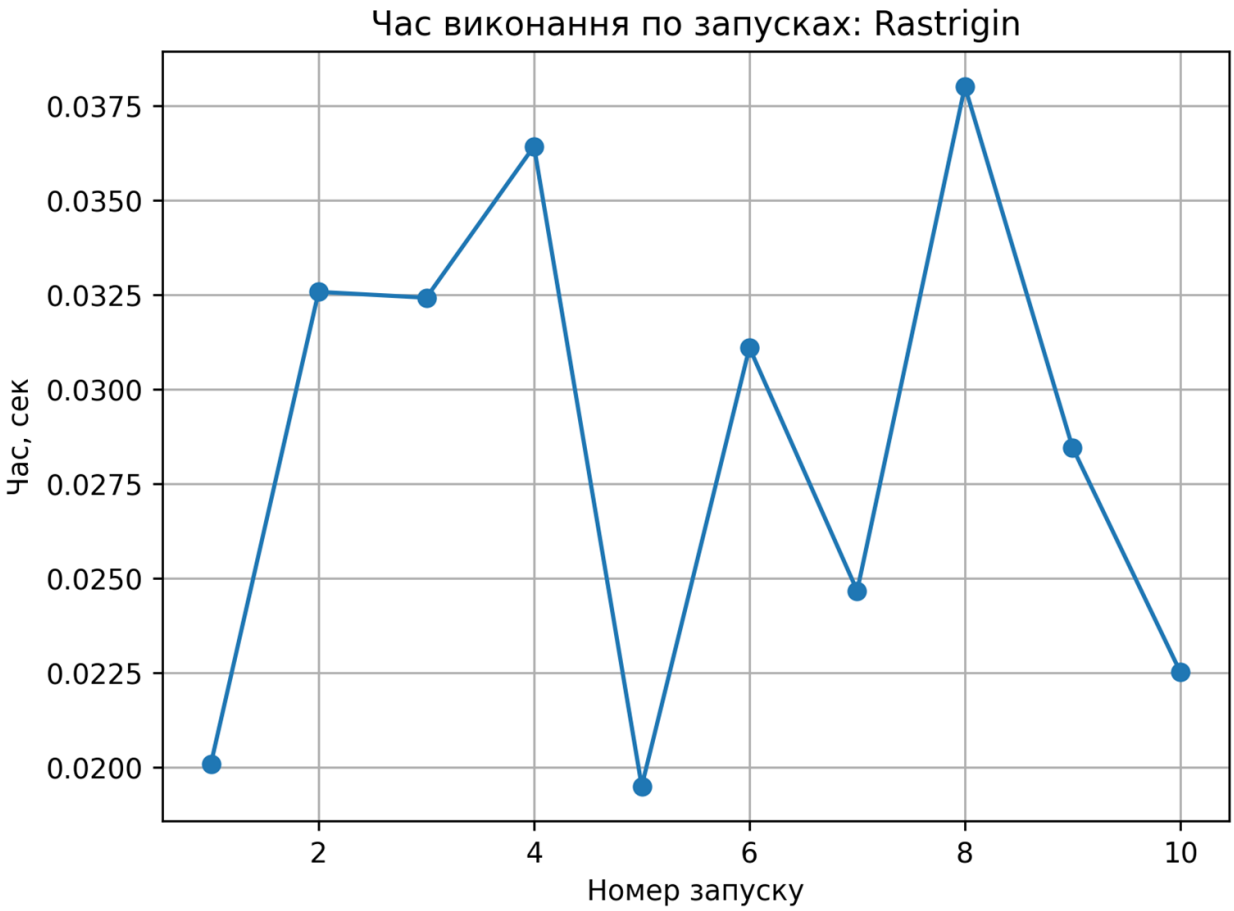


Рисунок 3.11 –Час виконання по запусках: Rastrigin

Наступний рисунок 3.12 показує час виконання алгоритму для кожного запуску на функції Rosenbrock:

- Мінімальний час: ≈ 0.030 с
- Максимальний час: ≈ 0.073 с
- Середній час: ≈ 0.048 сек

Спостерігається найбільша варіативність часу виконання. У деяких запусках алгоритм майже доходить до максимальних 300 поколінь (наприклад, 258–295 поколінь). Це підтверджує, що Rosenbrock є найскладнішою функцією серед досліджуваних.

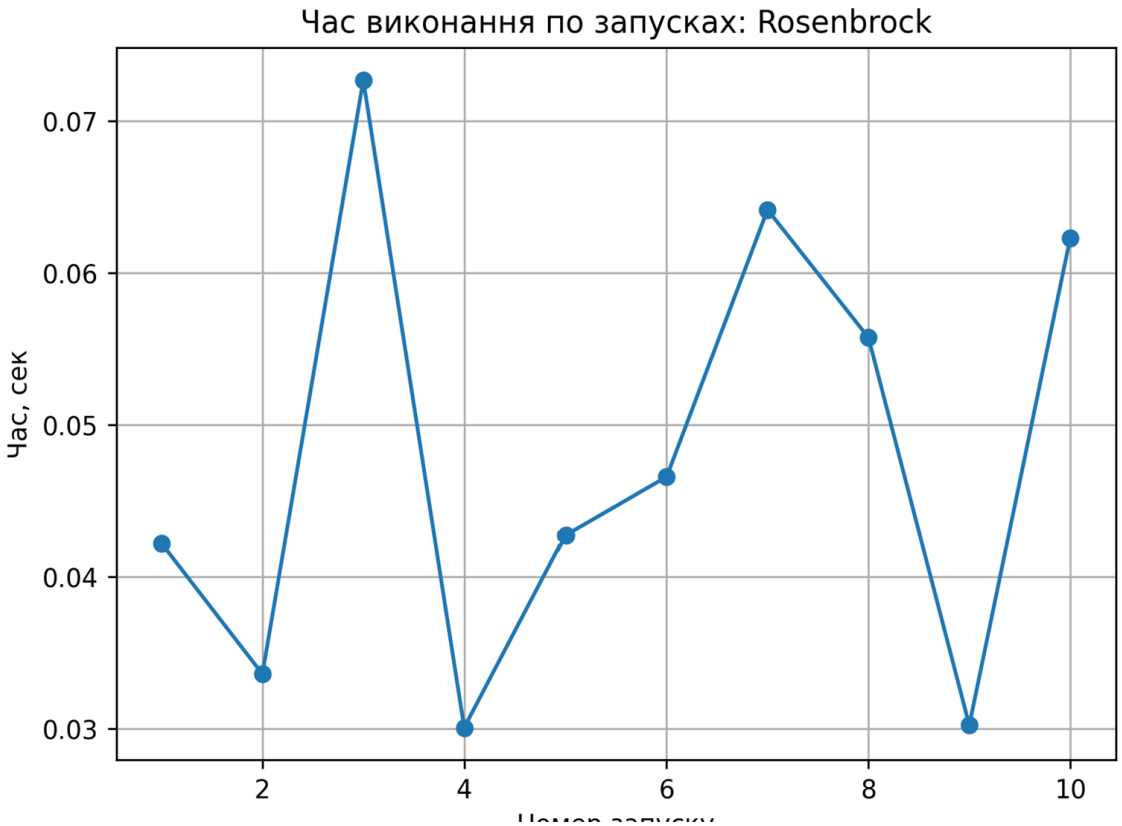


Рисунок 3.12 –Час виконання по запусках: Rosenbrock

Для кожної тестової функції, як і для першої конфігурації, було виконано 10 незалежних запусків, після чого обчислювалися середні значення часу виконання та кількості використаних поколінь.

Порівняння результатів першої базової конфігурації генетичного алгоритму з обмеженням у 150 поколінь та другої конфігурації з максимальним числом 300 поколінь дозволяє оцінити вплив збільшення кількості ітерацій на часову ефективність та збіжність алгоритму.

Для функції Sphere у першій конфігурації середня кількість поколінь до зупинки становила приблизно 133, а середній час виконання — близько 0,025 секунди. У другій конфігурації алгоритм у середньому використовував близько 139 поколінь, а середній час виконання зменшився до ≈ 0,024 секунди. Це свідчить про те, що збільшення максимального числа поколінь не призводить до суттєвого зростання обчислювальних витрат для простих унімодальних задач, оскільки алгоритм завершує роботу значно раніше за встановлене обмеження.

Для функції Rastrigin у першій конфігурації середня кількість поколінь до зупинки становила близько 132, а середній час виконання — ≈ 0,029 секунди. У другій конфігурації середня кількість поколінь зросла до ≈ 139, однак середній час виконання залишився практично незмінним (≈ 0,029 секунди). Це означає, що додаткові покоління не забезпечили суттєвого прискорення збіжності, проте

дозволили алгоритму стабільніше досліджувати складний мультимодальний простір пошуку.

Найбільш помітна різниця між конфігураціями спостерігається для функції Rosenbrock. У першій конфігурації середня кількість поколінь до зупинки склала приблизно 143, а середній час виконання — ≈ 0,038 секунди. У другій конфігурації алгоритм у середньому використовував вже ≈ 195 поколінь, а середній час виконання зріс до ≈ 0,048 секунди, причому в окремих запусках кількість поколінь майже досягала максимального обмеження (до 295 поколінь). Це підтверджує, що складна геометрія функції Rosenbrock вимагає значно більшої кількості ітерацій для ефективної збіжності, а збільшення максимальної кількості поколінь безпосередньо впливає на часові витрати.

Загалом, порівняльний аналіз показує, що збільшення максимальної кількості поколінь з 150 до 300 має незначний вплив на часову ефективність для простих та помірно складних функцій (Sphere, Rastrigin), оскільки алгоритм досягає критерію зупинки значно раніше. Водночас для складних оптимізаційних задач, таких як Rosenbrock, додаткові покоління дозволяють алгоритму глибше досліджувати простір рішень, проте супроводжуються зростанням загального часу виконання.

Отримані результати підтверджують доцільність використання обмеженої кількості поколінь для простих задач та необхідність розширення ітераційного ресурсу для складних функцій, що має враховуватися при налаштуванні параметрів генетичного алгоритму.

3.5.3 Аналіз третьої конфігурації

У наступному експерименті з третьої конфігурацією використовується наступна конфігурація параметрів генетичного алгоритму:

- розмірність задачі: dim = 10;
- розмір популяції: pop_size = 150;
- максимальна кількість поколінь: generations = 150;
- ймовірність кросинговеру: crossover_prob = 0.8;
- ймовірність мутації: mutation_prob = 0.05;
- параметр мутації: mutation_sigma = 0.1;
- критерій зупинки за стагнацією: stagnation_k = 30;
- турнірна селекція з параметром tournament_k = 3;

· межі простору пошуку: [-5.0; 5.0].

На рис. 3.13 наведено фрагмент CSV-файлу третьої конфігурації

ga_time_analizer > results > ga_time_benchmark.csv

```
function,run,time_sec,best_fitness,generations_done,pop_size,generations,crossover_prob,mutation_prob,mutation_sigma,dim
Sphere,1,0.06757860025390983,0.0021402598671860108,127,150,150,0.8,0.2,0.2,10
Sphere,2,0.07665899954736233,0.0012790281227220712,144,150,150,0.8,0.2,0.2,10
Sphere,3,0.06248800037428737,0.0013627946886532701,120,150,150,0.8,0.2,0.2,10
Sphere,4,0.08233919972553849,0.002044799311369628,150,150,150,0.8,0.2,0.2,10
Sphere,5,0.08405990013852715,0.001792929362980775,150,150,150,0.8,0.2,0.2,10
Sphere,6,0.06268689967691898,0.0013896105053582805,116,150,150,0.8,0.2,0.2,10
Sphere,7,0.08419100008904934,0.0014287077735630467,150,150,150,0.8,0.2,0.2,10
Sphere,8,0.08422710001468658,0.0012155851053579252,150,150,150,0.8,0.2,0.2,10
Sphere,9,0.053730500396341085,0.0018510846343390477,94,150,150,0.8,0.2,0.2,10
Sphere,10,0.05402350006625056,0.0021388842237676374,96,150,150,0.8,0.2,0.2,10
Rastrigin,1,0.09575519990175962,6.405199052958679,149,150,150,0.8,0.2,0.2,10
Rastrigin,2,0.09219169989228249,2.2683538109812673,150,150,150,0.8,0.2,0.2,10
Rastrigin,3,0.08107350021600723,2.2737871866758184,127,150,150,0.8,0.2,0.2,10
Rastrigin,4,0.09006550023332238,2.252027646612902,142,150,150,0.8,0.2,0.2,10
Rastrigin,5,0.07310390006750822,4.273204422054505,111,150,150,0.8,0.2,0.2,10
Rastrigin,6,0.09700920013710856,3.315204104646881,150,150,150,0.8,0.2,0.2,10
Rastrigin,7,0.09751840028911829,5.428694192524006,150,150,150,0.8,0.2,0.2,10
Rastrigin,8,0.07977840024977922,7.269213336184308,123,150,150,0.8,0.2,0.2,10
Rastrigin,9,0.0959187000989914,4.273360962907105,148,150,150,0.8,0.2,0.2,10
Rastrigin,10,0.09716500015929341,4.286077883246378,150,150,150,0.8,0.2,0.2,10
Rosenbrock,1,0.1146479002200067,7.7826213343277395,150,150,150,0.8,0.2,0.2,10
Rosenbrock,2,0.11452049994841218,8.36839471884833,150,150,150,0.8,0.2,0.2,10
Rosenbrock,3,0.11532099964097142,8.245706437609343,150,150,150,0.8,0.2,0.2,10
Rosenbrock,4,0.11325429985299706,3.7395459969131286,147,150,150,0.8,0.2,0.2,10
Rosenbrock,5,0.1067562997341156,7.730796379504404,137,150,150,0.8,0.2,0.2,10
Rosenbrock,6,0.11605799989774823,9.189978498719212,150,150,150,0.8,0.2,0.2,10
Rosenbrock,7,0.0890546003356576,8.810662943134288,115,150,150,0.8,0.2,0.2,10
Rosenbrock,8,0.07935590017586946,6.42585550864202,102,150,150,0.8,0.2,0.2,10
Rosenbrock,9,0.11658510006964207,8.30754169163024,150,150,150,0.8,0.2,0.2,10
Rosenbrock,10,0.1130607002414763,8.366411428027961,150,150,150,0.8,0.2,0.2,10
```

Рисунок 3.13 – Фрагмент CSV-файлу з результатами експериментальних запусків генетичного алгоритму

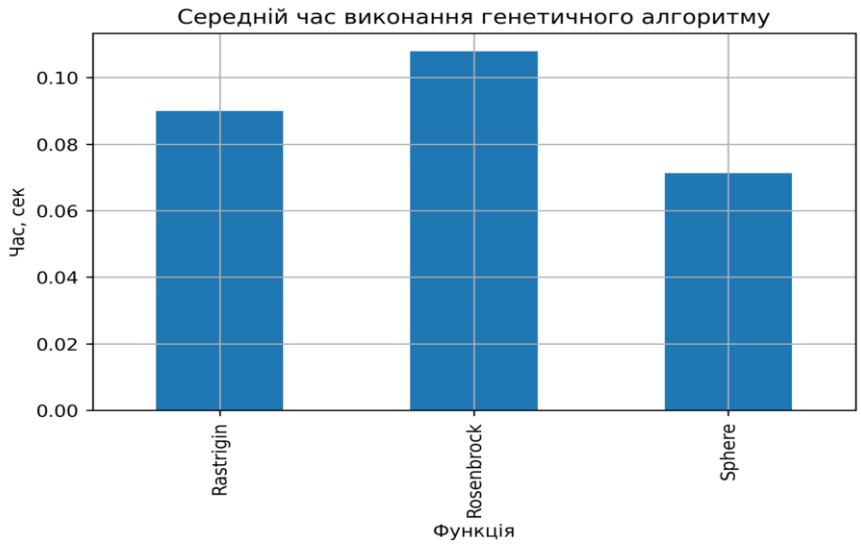


Рисунок 3.14 – Середній час виконання генетичного алгоритму

З рисунка 3.14 видно, що середній час виконання генетичного алгоритму при розмірі популяції 150 склав:

· Sphere — приблизно 0,071 с;

· Rastrigin — приблизно 0,089–0,090 с;

· Rosenbrock — приблизно 0,108 с.

У порівнянні з базовою конфігурацією (pop_size = 50) спостерігається чітке зростання часу виконання, що є очікуваним результатом, оскільки кількість обчислень функції пристосованості на кожному поколінні зросла утричі.

Рисунок 3.15 – Середня кількість поколінь до зупинки алгоритму

З рисунка 3.15 «Середня кількість поколінь до зупинки алгоритму» видно, що середня кількість поколінь до зупинки алгоритму становила:

· Sphere — близько 130 поколінь;

· Rastrigin — близько 140 поколінь;

· Rosenbrock — близько 140 поколінь.

При цьому кількість поколінь не зросла пропорційно розміру популяції, оскільки критерієм зупинки було досягнення максимального числа поколінь, а не стагнація. Це свідчить про те, що збільшення популяції в першу чергу впливає саме на часові витрати, а не на швидкість збіжності за поколіннями.

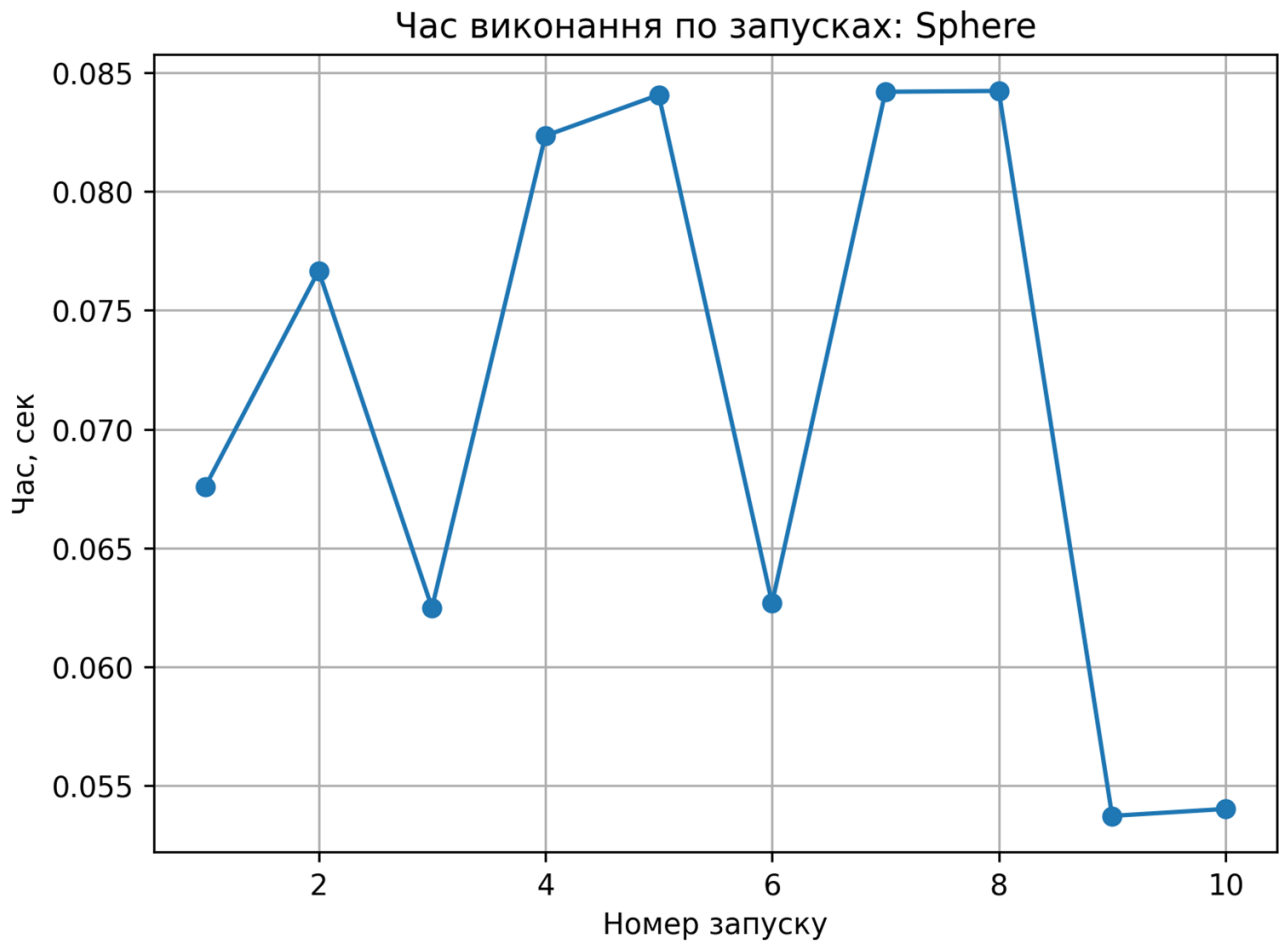


Рисунок 3.16 – Час виконання по запусках: Sphere

З рисунка 3.16 часу виконання по запусках та з узагальненого стовпчикового графіка видно, що:

- середній час виконання для функції Sphere становить приблизно 0,071 с;
- значення часу коливаються в межах ≈ 0,054 – 0,084 с;
- у більшості запусків алгоритм завершується раніше максимальних 150 поколінь, що зменшує загальний час роботи.

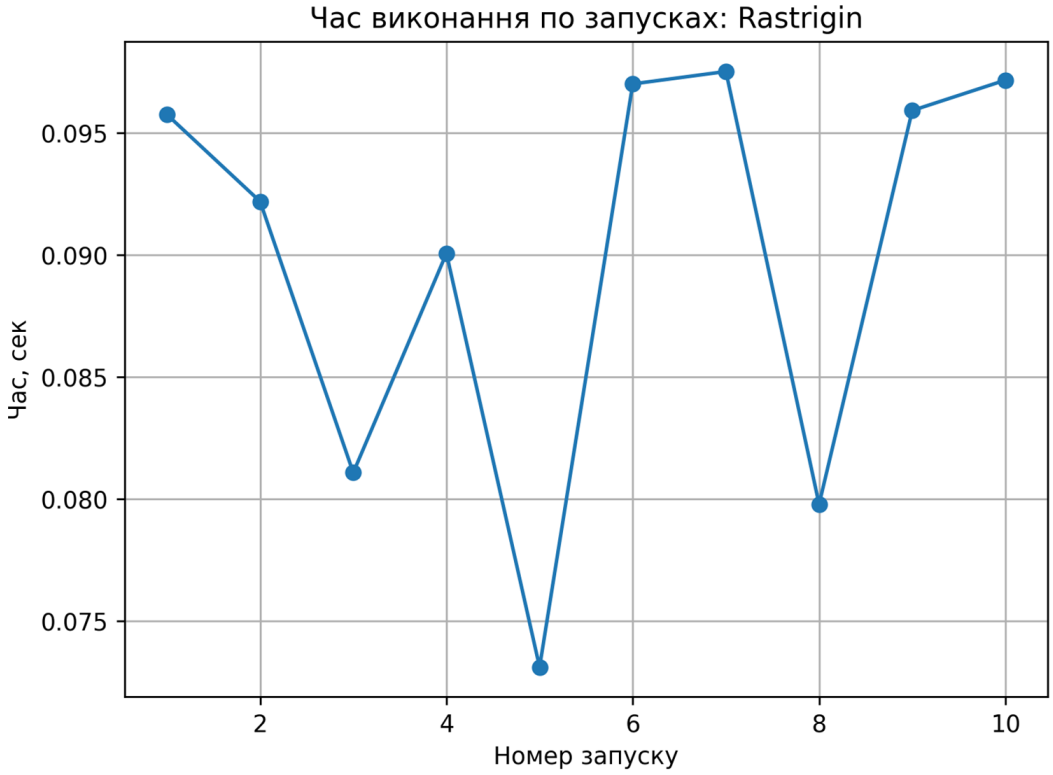


Рисунок 3.17 – Час виконання по запусках: Rastrigin

Рисунок 3.17 для функції Rastrigin спостерігається помітне зростання обчислювальних витрат:

- середній час виконання становить близько 0,090 с;
- час виконання змінюється в межах ≈ 0,073 – 0,098 с;
- між окремими запусками спостерігається більша варіативність часу.

З рисунока 3.18 для функції Rosenbrock спостерігається помітне зростання обчислювальних витрат:

- середній час виконання становить близько 0,110 с;
- час виконання змінюється в межах ≈ 0,108 с;
- між окремими запусками спостерігається більша варіативність часу.

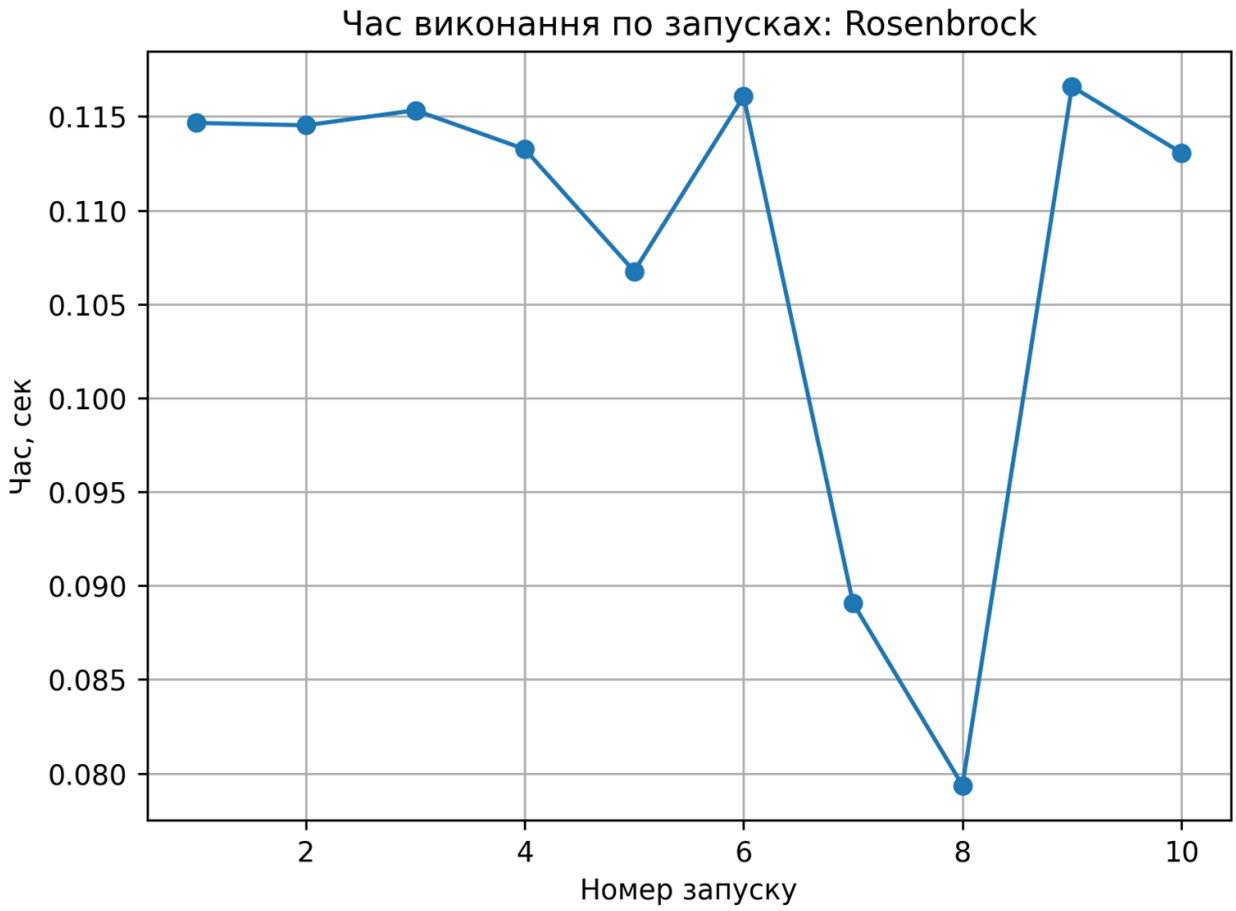


Рисунок 3.18 – Час виконання по запусках: Rosenbrock

На відміну від базової конфігурації, в якій використовувався менший розмір популяції, у цій конфігурації значення pop_size було збільшено до 150, тоді як усі інші параметри алгоритму (розмірність задачі, кількість поколінь, ймовірності кросовінгу та мутації, критерій зупинки) залишалися незмінними.

Порівняння з базовою конфігурацією отриманих результатів третьої конфігурацією показало, що збільшення розміру популяції призвело до зростання часу виконання алгоритму для всіх тестових функцій, однак ступінь цього впливу залежить від складності задачі.

- Для функції Sphere середній час виконання зріс приблизно з ~0,05 с у базовій конфігурації до ~0,07 с у третій конфігурації, тобто збільшився майже на 40%. При цьому алгоритм, як і раніше, у більшості запусків завершувався значно раніше максимальної кількості поколінь.
- Для функції Rastrigin середній час виконання збільшився з ~0,07–0,08 с до ~0,09 с, що пояснюється необхідністю оцінки більшої кількості особин у кожному поколінні за умов складнішого, мультимодального простору пошуку.

- Для функції Rosenbrock зростання часу виконання було найбільш помітним: з ~0,08–0,09 с у базовій конфігурації до ~0,108 с у третій конфігурації. Це пов’язано з тим, що збільшення популяції суттєво підвищує обчислювальні витрати в умовах складної геометрії функції та повільної збіжності.

Результати третьої конфігурації підтверджують, що розмір популяції є одним із ключових параметрів, які впливають на часову ефективність генетичного алгоритму. Збільшення кількості особин у популяції призводить до лінійного зростання обчислювальних витрат на кожне покоління, що безпосередньо відображається на загальному часі виконання.

Для простих унімодальних задач з гладким простором пошуку, таких як функція Sphere, використання великої популяції є недоцільним, оскільки воно не дає суттєвого покращення збіжності, але призводить до помітного збільшення часу виконання алгоритму. У таких випадках доцільніше застосовувати менший розмір популяції, що дозволяє зберегти високу швидкодію без втрати якості розв’язку.

Натомість для складніших задач збільшення популяції може бути виправданим з точки зору підтримання різноманіття, однак це супроводжується зростанням часових витрат, що необхідно враховувати при практичному застосуванні генетичних алгоритмів.

3.5.4 Вплив розмірності задачі на час виконання

У четвертій конфігурації досліджувалася залежність часової ефективності генетичного алгоритму від розмірності задачі. Порівняно з базовою конфігурацією, де розмірність складала $dim = 10$, у даному експерименті вона була збільшена до $dim = 30$, тоді як усі інші параметри алгоритму залишалися незмінними

На рисунку 3.19 бачимо файл з результатами експериментальних 10 запусків генетичного алгоритму.

```
function,run,time_sec,best_fitness,generations_done,pop_size,generations,crossover_prob,mutation_prob,mutation_sigma,dim
Sphere,1,0.18837369978427887,0.06255900723277133,150,150,150,0.8,0.2,0.2,30
Sphere,2,0.18748159985989332,0.06643364785309214,150,150,150,0.8,0.2,0.2,30
Sphere,3,0.18892010021954775,0.08100688878685912,150,150,150,0.8,0.2,0.2,30
Sphere,4,0.1919746999628842,0.068033614017731,150,150,150,0.8,0.2,0.2,30
Sphere,5,0.1872274000197649,0.0661775427878742 3,150,150,150,0.8,0.2,0.2,30
Sphere,6,0.16542259976267815,0.08320071447990349,135,150,150,0.8,0.2,0.2,30
Sphere,7,0.18038339959457517,0.06589614847956746,150,150,150,0.8,0.2,0.2,30
Sphere,8,0.1838962002657354,0.06939668999663291,150,150,150,0.8,0.2,0.2,30
Sphere,9,0.18163680005818605,0.07356794902922767,150,150,150,0.8,0.2,0.2,30
Sphere,10,0.18217030027881265,0.06659152442896385,147,150,150,0.8,0.2,0.2,30
Rastrigin,1,0.1903128996491 4322,20.973956787766497,128,150,150,0.8,0.2,0.2,30
Rastrigin,2,0.21758090006187558,24.148916086384475,150,150,150,0.8,0.2,0.2,30
Rastrigin,3,0.2226319001056254,33.06784841272554,150,150,150,0.8,0.2,0.2,30
Rastrigin,4,0.21926470007747412,27.05646555759114,150,150,150,0.8,0.2,0.2,30
Rastrigin,5,0.18722809990867972,32.03907074089443,128,150,150,0.8,0.2,0.2,30
Rastrigin,6,0.21819189982488751,17.090460030933173,150,150,150,0.8,0.2,0.2,30
Rastrigin,7,0.21867420012131333,25.199754964715396,150,150,150,0.8,0.2,0.2,30
Rastrigin,8,0.17072279984131455,23.95431413341015,116,150,150,0.8,0.2,0.2,30
Rastrigin,9,0.2191270999610424,28.08501043564644,150,150,150,0.8,0.2,0.2,30
Rastrigin,10,0.21892409957945347,44.32882149117546,150,150,150,0.8,0.2,0.2,30
Rosenbrock,1,0.2815861999988556,37.60856913684015,150,150,150,0.8,0.2,0.2,30
Rosenbrock,2,0.2848650999367237,36.33178728509995,150,150,150,0.8,0.2,0.2,30
Rosenbrock,3,0.28349529998376966,32.91723551670737,150,150,150,0.8,0.2,0.2,30
Rosenbrock,4,0.28026910033077,34.12750122350437,150,150,150,0.8,0.2,0.2,30
Rosenbrock,5,0.28319000033661723,33.90993838740249,150,150,150,0.8,0.2,0.2,30
Rosenbrock,6,0.23195730010047555,36.37233026236922,123,150,150,0.8,0.2,0.2,30
Rosenbrock,7,0.2843152000568807,33.517406988326876,150,150,150,0.8,0.2,0.2,30
Rosenbrock,8,0.2802828000858426,34.25117212800098,150,150,150,0.8,0.2,0.2,30
Rosenbrock,9,0.28215919993817806,35.19930467928082,150,150,150,0.8,0.2,0.2,30
Rosenbrock,10,0.265782100148499,34.595027842429744,142,150,150,0.8,0.2,0.2,30
```

Рисунок 3.19 – Фрагмент CSV-файлу з результатами експериментальних запусків генетичного алгоритму

На рисунку 3.20 наведено середні значення часу виконання алгоритму для кожної тестової функції при розмірності dim = 30.

- для функції Sphere середній час становить приблизно 0,18 с;
- для функції Rastrigin — близько 0,21 с;
- для функції Rosenbrock — близько 0,27 с.

У порівнянні з базовою конфігурацією спостерігається стабільне зростання часу виконання для всіх функцій, причому найбільше — для функції Rosenbrock. Це підтверджує залежність часової ефективності генетичного алгоритму не лише від складності функції, але й від розмірності задачі.

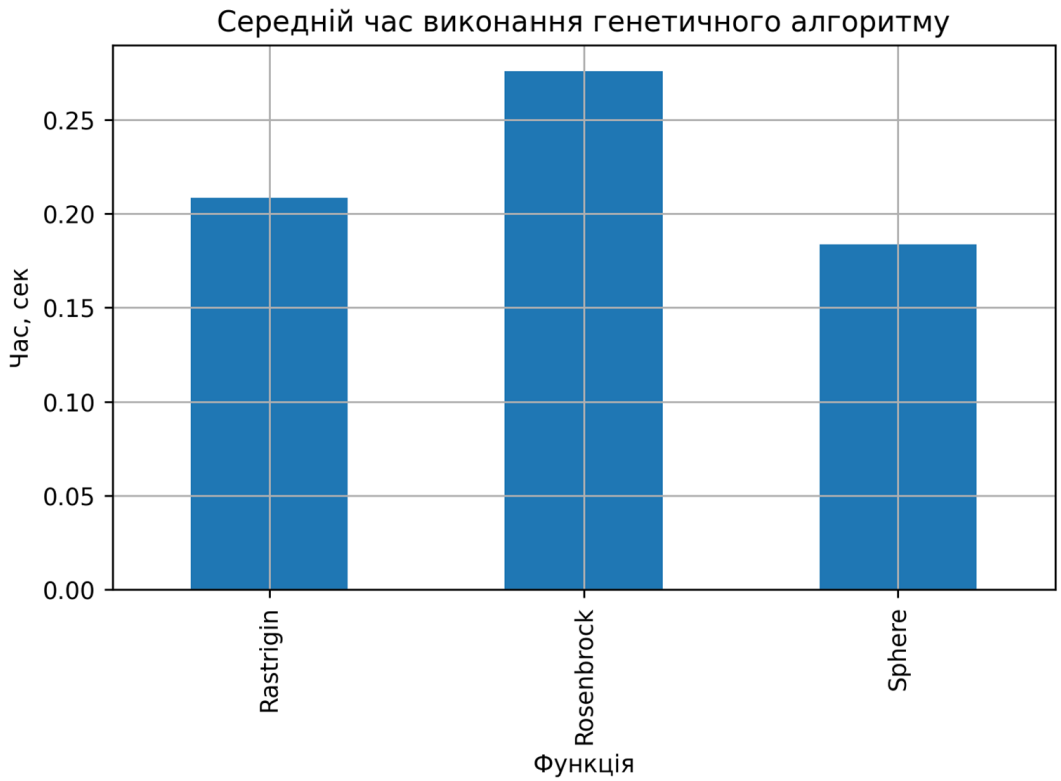


Рисунок 3.20 — Середній час виконання генетичного алгоритму для різних функцій

З рисунка 3.21 видно, що середня кількість поколінь до зупинки для всіх функцій перебуває в межах 145–150 поколінь, тобто близька до максимально допустимого значення.

Порівняно з базовою конфігурацією, кількість поколінь не зросла пропорційно розмірності задачі, оскільки критерієм зупинки виступало досягнення максимального числа поколінь, а не стагнація.

Середня кількість поколінь до зупинки:

- Sphere ≈ 148
- Rastrigin ≈ 142
- Rosenbrock ≈ 147

Рисунок 3.21 — Середня кількість поколінь до зупинки

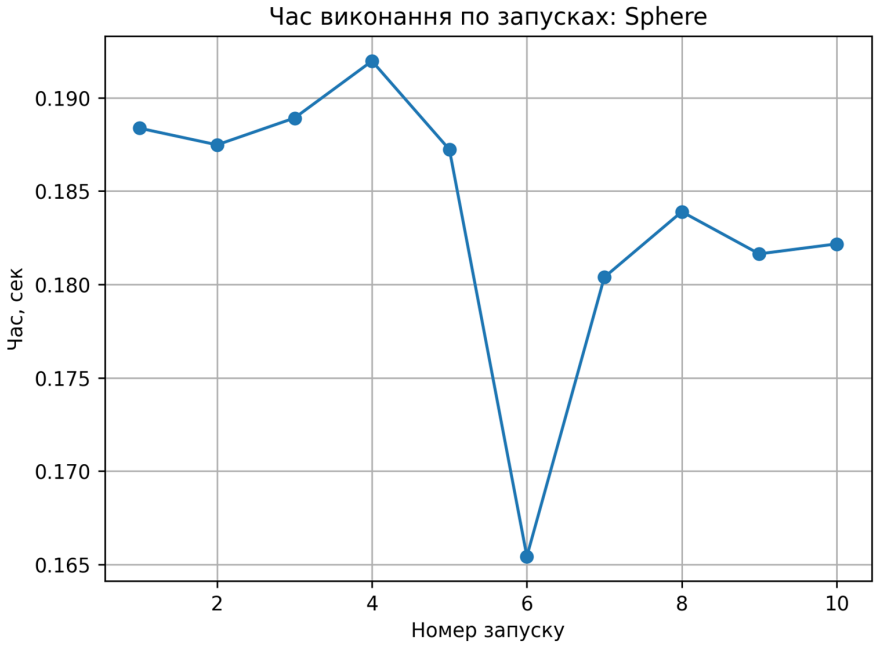


Рисунок 3.22 — Час виконання генетичного алгоритму по запусках для функції Sphere

З рисунка 3.22 видно, що для функції Sphere при збільшенні розмірності задачі з dim = 10 (базова конфігурація) до dim = 30 час виконання алгоритму суттєво зріс у кожному запуску.

У четвертій конфігурації значення часу виконання перебувають у межах 0,165–0,192 с, тоді як у базовій конфігурації вони складали приблизно 0,05–0,08 с. Таким чином, збільшення розмірності призвело до зростання часу виконання майже у 2,5–3 рази.

При цьому форма графіка залишається відносно стабільною, без різких коливань між запусками, що свідчить про передбачувану поведінку алгоритму для унімодальної функції навіть за підвищеної розмірності.

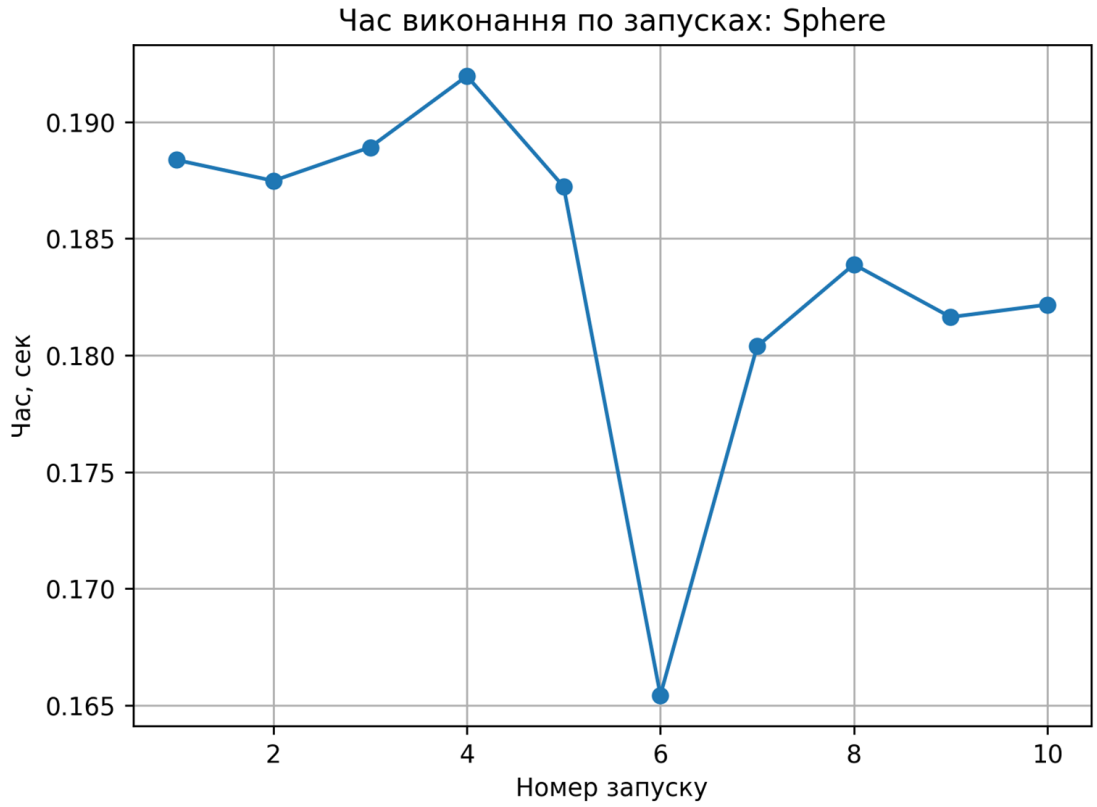


Рисунок 3.22 — Час виконання генетичного алгоритму по запусках для функції Sphere

З рисунка 3.22 видно, що для функції Sphere при збільшенні розмірності задачі з dim = 10 (базова конфігурація) до dim = 30 час виконання алгоритму суттєво зріс у кожному запуску.

У четвертій конфігурації значення часу виконання перебувають у межах 0,165–0,192 с, тоді як у базовій конфігурації вони складали приблизно 0,05–0,08 с. Таким чином, збільшення розмірності призвело до зростання часу виконання майже у 2,5–3 рази.

На рисунку 3.23 показано зміну часу виконання алгоритму для мультимодальної функції Rastrigin при розмірності dim = 30.

Порівняно з базовою конфігурацією, де середній час виконання становив близько 0,08–0,10 с, у четвертій конфігурації він збільшився до 0,17–0,22 с. Це означає зростання часу приблизно у 2–2,5 раза.

Рисунок 3.23 — Час виконання генетичного алгоритму по запусках для функції Rastrigin

Рисунок 3.24 демонструє результати для функції Rosenbrock, яка є найбільш складною серед розглянутих тестових функцій.

У порівнянні з базовою конфігурацією (dim = 10), де час виконання перебував у межах 0,11–0,12 с, при збільшенні розмірності до dim = 30 середній час виконання зріс до 0,26–0,28 с.

Таким чином, час виконання алгоритму для функції Rosenbrock збільшився більш ніж у 2 рази, що є найбільшим приростом серед усіх тестових функцій.

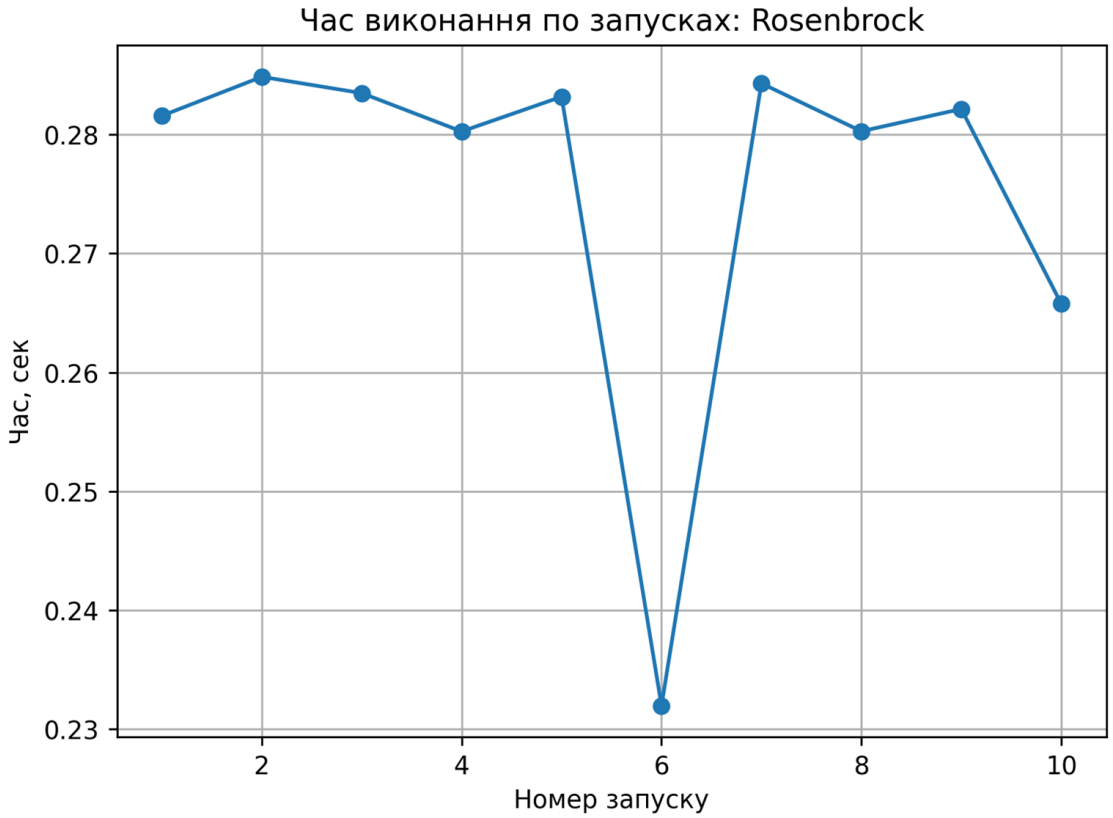


Рисунок 3.24 — Час виконання генетичного алгоритму по запусках для функції Rosenbrock

Порівняння середнього часу виконання алгоритму в четвертій конфігурації на відміну від першої (базової):

Sphere

- Базова конфігурація ≈ 0,07 с
- Четверта конфігурація ≈ 0,18 с

Збільшення часу: приблизно у 2,5 раза

Rastrigin

- Базова конфігурація ≈ 0,09 с
- Четверта конфігурація ≈ 0,21 с

Збільшення часу: приблизно у 2,3 раза

Rosenbrock

- Базова конфігурація ≈ 0,11 с

· Четверта конфігурація ≈ 0,28 с

Збільшення часу: приблизно у 2,5 раза

Порівняння четвертої конфігурації з базовою показало, що збільшення розмірності задачі має істотний вплив на часову ефективність генетичного алгоритму, тоді як вплив на кількість поколінь є другорядним.

Основне зростання обчислювальних витрат відбувається за рахунок ускладнення обчислень усередині кожного покоління, а не через збільшення кількості ітерацій. Це підтверджує, що розмірність задачі є одним із ключових факторів, що визначає час виконання генетичних алгоритмів.

3.5.5 Вплив ймовірності кросинговеру

У п’ятій конфігурації змінюється лише ймовірність кросинговеру з 0.8 на 0.6, тоді як усі інші параметри залишаються на базовому рівні.

```
function,run,time_sec,best_fitness,generations_done,pop_size,generations,crossover_prob,mutation_prob,mutation_sigma,dim
Sphere,1,0.06556219980120659,0.0036730893507320094,135,150,150,0.6,0.2,0.2,10
Sphere,2,0.06867399998009205,0.0028585494929348385,139,150,150,0.6,0.2,0.2,10
Sphere,3,0.07708350010216236,0.00275328266245848,150,150,150,0.6,0.2,0.2,10
Sphere,4,0.07175560016185045,0.0027611040418846133,135,150,150,0.6,0.2,0.2,10
Sphere,5,0.056858399882912636,0.002959208287829401,112,150,150,0.6,0.2,0.2,10
Sphere,6,0.07886959984898567,0.0015658560020228398,150,150,150,0.6,0.2,0.2,10
Sphere,7,0.07207460002973676,0.0015574533983919935,142,150,150,0.6,0.2,0.2,10
Sphere,8,0.05712629994377494,0.0024555456017332326,108,150,150,0.6,0.2,0.2,10
Sphere,9,0.07152089988812804,0.0026027108987167467,137,150,150,0.6,0.2,0.2,10
Sphere,10,0.06142350006848574,0.002734638901446832,119,150,150,0.6,0.2,0.2,10
Rastrigin,1,0.0681094997562468,10.412070316441117,113,150,150,0.6,0.2,0.2,10
Rastrigin,2,0.09052050020545721,15.224519924494771,150,150,150,0.6,0.2,0.2,10
Rastrigin,3,0.059529300313442945,7.106825915842407,97,150,150,0.6,0.2,0.2,10
Rastrigin,4,0.05385180003941059,4.566823064850439,88,150,150,0.6,0.2,0.2,10
Rastrigin,5,0.09164079977199435,2.382803021459182,150,150,150,0.6,0.2,0.2,10
Rastrigin,6,0.05469929985702038,7.5387586906895905,90,150,150,0.6,0.2,0.2,10
Rastrigin,7,0.07386030023917556,3.3109279351038765,121,150,150,0.6,0.2,0.2,10
Rastrigin,8,0.06258400017395616,4.376390426456027,102,150,150,0.6,0.2,0.2,10
Rastrigin,9,0.09305169992148876,5.392575891130619,150,150,150,0.6,0.2,0.2,10
Rastrigin,10,0.07069439999759197,10.280583215772268,115,150,150,0.6,0.2,0.2,10
Rosenbrock,1,0.11002459982410073,6.340257380478366,147,150,150,0.6,0.2,0.2,10
Rosenbrock,2,0.11138830007985234,7.7069749163819425,150,150,150,0.6,0.2,0.2,10
Rosenbrock,3,0.11038829991593957,8.307332702078254,150,150,150,0.6,0.2,0.2,10
Rosenbrock,4,0.11082850024104118,8.01980765673866,150,150,150,0.6,0.2,0.2,10
Rosenbrock,5,0.11060860008001328,7.789643254317189,150,150,150,0.6,0.2,0.2,10
Rosenbrock,6,0.07685629976913333,9.80981065594123,107,150,150,0.6,0.2,0.2,10
Rosenbrock,7,0.11113899992778897,8.544805238198107,149,150,150,0.6,0.2,0.2,10
Rosenbrock,8,0.08864669967442751,8.119463246626612,122,150,150,0.6,0.2,0.2,10
Rosenbrock,9,0.10989479999989271,9.005861458086832,150,150,150,0.6,0.2,0.2,10
Rosenbrock,10,0.07224140036851168,8.44087212610978,97,150,150,0.6,0.2,0.2,10
```

Рисунок 3.25 — Фрагмент CSV-файлу з результатами експериментальних запусків генетичного алгоритму

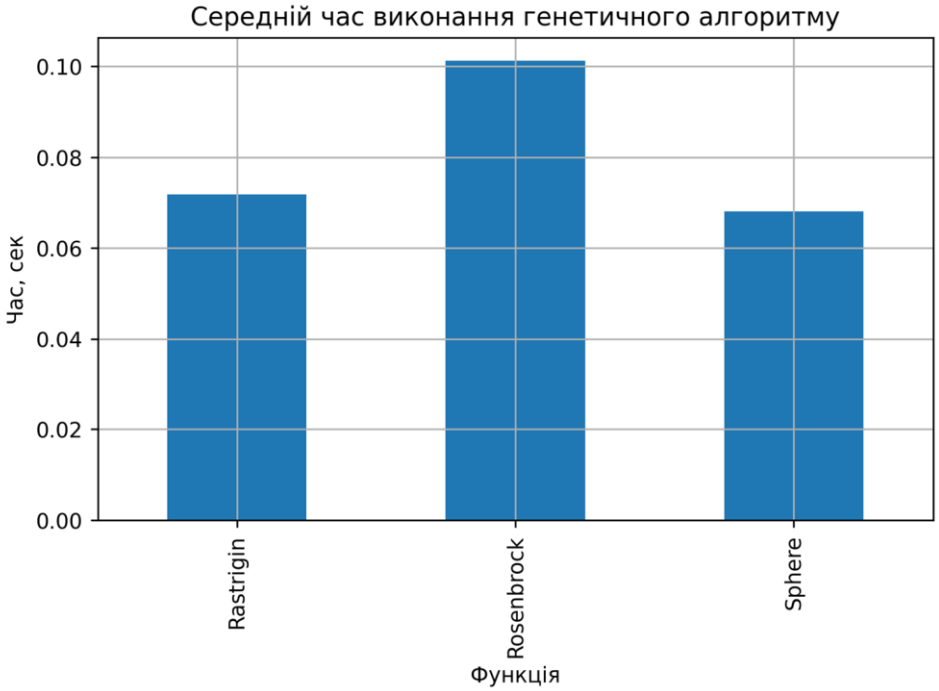


Рисунок 3.26 – Середній час виконання генетичного алгоритму

На рисунку 3.26 зображено середній час виконання генетичного алгоритму для трьох тестових функцій: Sphere, Rastrigin та Rosenbrock у п’ятій конфігурації параметрів.

У порівнянні з базовою конфігурацією, середній час виконання не зменшився, а для складніших функцій збільшився.

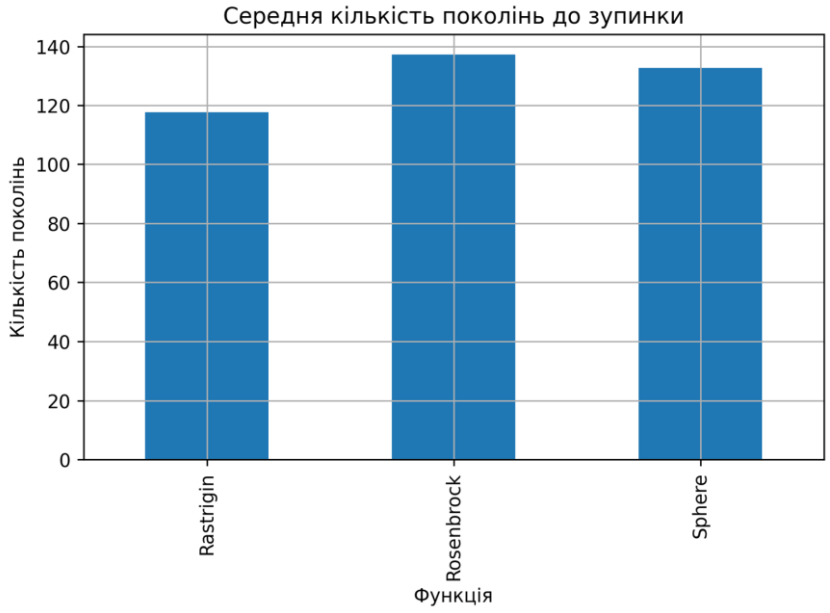


Рисунок 3.27 – Середня кількість поколінь до зупинки алгоритму

Порівняно з базовою конфігурацією, кількість поколінь до зупинки зросла, що вказує на уповільнення процесу збіжності алгоритму, ми бачимо це на рисунку 3.27.

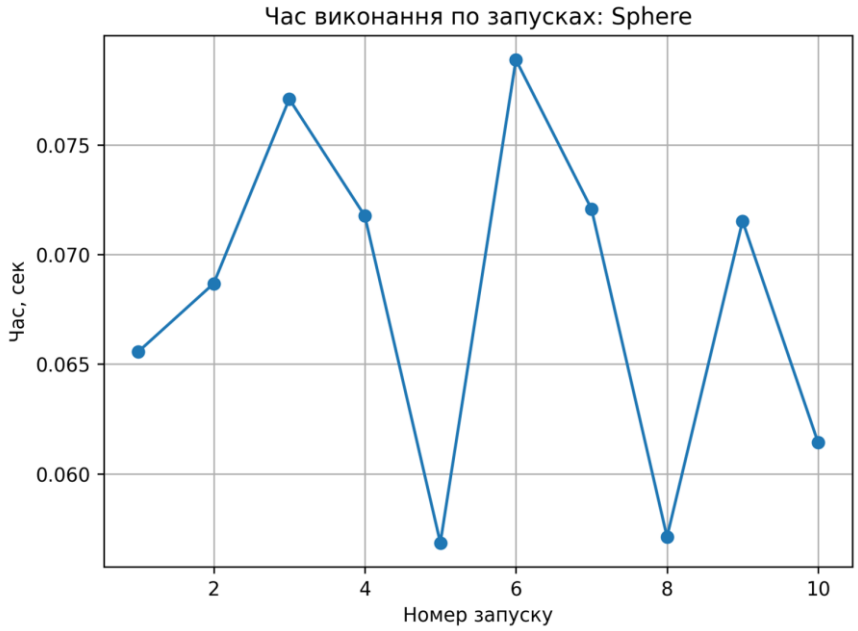


Рисунок 3.28 – Час виконання алгоритму за запусками для функції Sphere

Ми бачимо на рисунку 3.28 час виконання коливається в межах приблизно від 0,057 с до 0,078 с.

У порівнянні з базовою конфігурацією, стабільність часу виконання зберігається, проте суттєвого прискорення не відбулося та уповільнення.

Рисунок 3.29 – Час виконання алгоритму за запусками для функції Rastrigin

На рисунку 3.29 бачимо, що час змінюється в межах приблизно від 0,054 с до 0,093 с, що вказує на значну варіативність результатів.

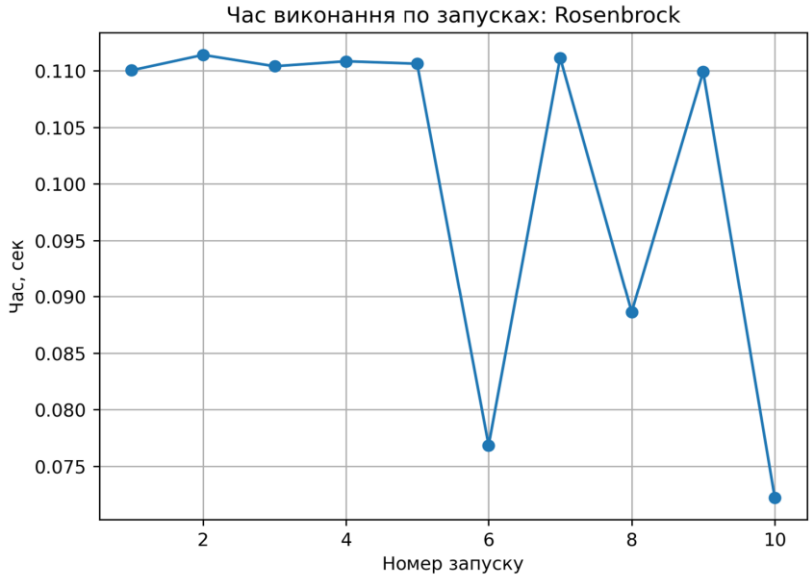


Рисунок 3.30 – Час виконання алгоритму за запусками для функції Rosenbrock

На рисунку 3.30 бачимо, що у порівнянні з базовою конфігурацією, час виконання для Rosenbrock збільшився, а стабільність збіжності зменшилася.

Час виконаня функції теж виріс.

Отже, о п’ятій конфігурації змінювалась лише ймовірність кросинговеру, тоді як усі інші параметри залишались на базовому рівні.

Порівняння середнього часу виконання алгоритму та кількість поколінь до зупинки в п’ятій конфігурації на відміну від першої (базової):

Sphere

· Базова конфігурація ≈ 0,07 с,

середня кількість поколінь до зупинки ≈ 119 поколінь

· П’ята конфігурація ≈ 0,068 с,

середня кількість поколінь до зупинки ≈ 133 поколінь

Час виконання збільшився приблизно на 0,006 с, кількість поколінь зросла приблизно на 14 поколінь.

Rastrigin

· Базова конфігурація ≈ 0,09 с

середня кількість поколінь до зупинки ≈ 108 поколінь

· П’ята конфігурація ≈ 0,072 с

середня кількість поколінь до зупинки ≈ 118 поколінь

Час виконання збільшився приблизно на 0,007 с, кількість поколінь до зупинки зросла приблизно на 10 поколінь.

Rosenbrock

· Базова конфігурація ≈ 0,11 с

середня кількість поколінь до зупинки ≈ 125 поколінь

· П’ята конфігурація ≈ 0,28 с

середня кількість поколінь до зупинки ≈ 137 поколінь

Час виконання збільшився приблизно на 0,006 с, кількість поколінь до зупинки зросла приблизно на 12 поколінь.

Тобто, можемо зробити висновок, що Порівняння результатів п’ятої конфігурації з базовою показало, що збільшення ймовірності кросинговеру призводить до зростання часу виконання генетичного алгоритму в середньому на 0,006–0,007 секунди, а також до збільшення кількості поколінь до зупинки на 10–14 поколінь залежно від тестової функції.

Отримані результати свідчать про те, що надмірно інтенсивний кросинговер ускладнює закріплення найкращих особин у популяції, що уповільнює процес збіжності алгоритму. Найбільш помітний негативний ефект спостерігається для складної функції Rosenbrock, яка є найбільш чутливою до зміни параметрів генетичного алгоритму.

3.5.6 Вплив параметрів мутації

У шостій конфігурації змінюється ймовірність мутації, параметр інтенсивності мутації (σ), тоді як інші параметри залишаються незмінними.

```
function,run,time_sec,best_fitness,generations_done,pop_size,generations,crossover_prob,mutation_prob,mutation_sigma,dim
Sphere,1,0.06876099994406104,0.0011625738162928812,140,50,150,0.8,0.15,0.2,10
Sphere,2,0.0411769999191165,0.0008991213190507968,127,50,150,0.8,0.15,0.2,10
Sphere,3,0.04231920000165701,0.0011106968777694572,122,50,150,0.8,0.15,0.2,10
Sphere,4,0.04582240013405681,0.0014322173933303045,144,50,150,0.8,0.15,0.2,10
Sphere,5,0.04725199984386563,0.0011869113778649903,150,50,150,0.8,0.15,0.2,10
Sphere,6,0.041231400333344936,0.0009449123925345993,132,50,150,0.8,0.15,0.2,10
Sphere,7,0.04903150023892522,0.0009076480450759104,150,50,150,0.8,0.15,0.2,10
Sphere,8,0.0428468999452889,0.001603461281504296,120,50,150,0.8,0.15,0.2,10
Sphere,9,0.04038089979439974,0.0014793593986075714,126,50,150,0.8,0.15,0.2,10
Sphere,10,0.048214399721473455,0.0005887521197563095,150,50,150,0.8,0.15,0.2,10
Rastrigin,1,0.049917600117623806,9.302809523666568,93,50,150,0.8,0.15,0.2,10
Rastrigin,2,0.05585670005530119,13.152661150116757,147,50,150,0.8,0.15,0.2,10
Rastrigin,3,0.03986109979450703,3.190605772526652,104,50,150,0.8,0.15,0.2,10
Rastrigin,4,0.059363999869674444,16.03415497377142,142,50,150,0.8,0.15,0.2,10
Rastrigin,5,0.06440529972314835,14.203146834091484,150,50,150,0.8,0.15,0.2,10
Rastrigin,6,0.04586279997602105,3.413980033745176,121,50,150,0.8,0.15,0.2,10
Rastrigin,7,0.05733799980953336,14.033763970757093,150,50,150,0.8,0.15,0.2,10
Rastrigin,8,0.06692519970238209,15.291140434591114,150,50,150,0.8,0.15,0.2,10
Rastrigin,9,0.044364700093865395,13.206188910589646,111,50,150,0.8,0.15,0.2,10
Rastrigin,10,0.05836750008165836,9.110018231140856,143,50,150,0.8,0.15,0.2,10
Rosenbrock,1,0.07324430020526052,4.7661761916353464,150,50,150,0.8,0.15,0.2,10
Rosenbrock,2,0.07517370022833347,8.271857875994789,150,50,150,0.8,0.15,0.2,10
Rosenbrock,3,0.05394789995625615,8.127182310936737,98,50,150,0.8,0.15,0.2,10
Rosenbrock,4,0.08037240011617541,8.355851714923427,150,50,150,0.8,0.15,0.2,10
Rosenbrock,5,0.07279559969902039,72.6002371572307,150,50,150,0.8,0.15,0.2,10
Rosenbrock,6,0.06216549966484308,9.48153484178285,133,50,150,0.8,0.15,0.2,10
Rosenbrock,7,0.06790840020403266,8.227263986131701,150,50,150,0.8,0.15,0.2,10
Rosenbrock,8,0.0672070998698473,5.63865374921392,150,50,150,0.8,0.15,0.2,10
Rosenbrock,9,0.055091199930757284,7.411066883726847,124,50,150,0.8,0.15,0.2,10
Rosenbrock,10,0.055090040039032698,7.78331314092552,124,50,150,0.8,0.15,0.2,10
```

Рисунок 3.31 — Фрагмент CSV-файлу з результатами експериментальних запусків генетичного алгоритму

Рисунок 3.32 — Середній час виконання генетичного алгоритму

На рисунку 3.32 бичимо, що середній виконання генетичного алгоритму не дуже відрізнається від базової конфігурації.

Рисунок 3.32 — Середня кількість поколінь до зупинки алгоритму

Рисунок 3.32 демонструє середню кількість поколінь, після яких алгоритм завершував роботу. Ми можемо спостерігати, що збільшення параметрів мутації призводить до повільнішої стабілізації популяції, оскільки нові особини постійно вносять різноманіття.

Рисунок 3.33 — Час виконання по запусках: Sphere

Час виконання є відносно стабільним на рисунку 3.33, з незначною варіативністю між запусками. Незважаючи на підвищену мутацію, ми можемо

постерігати, що проста унімодальна структура функції забезпечує швидку збіжність алгоритму навіть у цьому випадку.

Рисунок 3.34 — Час виконання по запусках: Rastrigin

Для функції Rastrigin на рисунку 3.34 ми можемо бачити, що спостерігається більша варіативність часу виконання між окремими запусками. Це пов’язано з мультимодальною природою функції та активним впливом оператора мутації, який дозволяє уникати локальних мінімумів, але збільшує обчислювальні витрати.

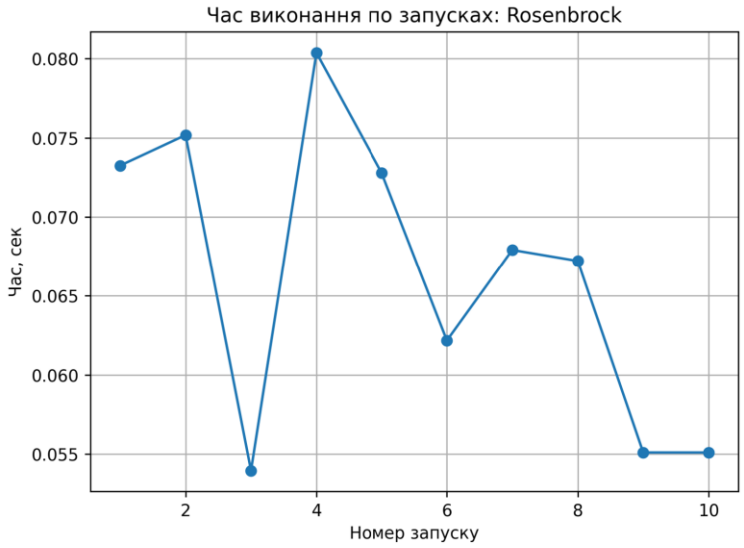


Рисунок 3.35 — Час виконання по запусках: Rosenbrock

Найбільша варіативність часу виконання зафіксована для функції Rosenbrock, бачим на рисунку 3.35 У частині запусків алгоритм працює майже до максимально дозволеної кількості поколінь.

Отже у шостій конфігурації змінили ймовірність мутації, параметр інтенсивності мутації (σ).

Результати за часом та кількістю поколінь в порівняні з базовою конфігурацією.

Sphere

- Базова конфігурація: середній час виконання ≈ 0,07 с,
  середня кількість поколінь до зупинки ≈ 119 поколінь
- Шоста конфігурація: середній час виконання ≈ 0,082 с,
  середня кількість поколінь до зупинки ≈ 131 покоління

Ми бачимо, що час виконання збільшився приблизно на 0,012 с, кількість поколінь до зупинки зросла приблизно на 12 поколінь, тобо для простої унімодальної функції Sphere підвищена мутація призвела до зайвих обчислювальних витрат.

Rastrigin

- Базова конфігурація: середній час виконання ≈ 0,09 с,
  середня кількість поколінь до зупинки ≈ 108 поколінь
- Шоста конфігурація: середній час виконання ≈ 0,105 с, середня кількість поколінь до зупинки ≈ 124 покоління

Час виконання збільшився приблизно на 0,015 с, кількість поколінь до зупинки зросла приблизно на 16 поколінь.

Для мультимодальної функції Rastrigin підвищення параметрів мутації активізувало дослідження простору пошуку, однак водночас уповільнило процес стабілізації популяції.

Порівнявши результати шостої конфігурації з базовою ми бачимо, що збільшення ймовірності мутації та параметра σ призводить до зростання часу виконання генетичного алгоритму в середньому на 0,012–0,025 секунди, а також до збільшення кількості поколінь до зупинки на 12–16 поколінь, залежно від складності тестової функції.

Отримані результати свідчать про те, що надмірно інтенсивна мутація ускладнює процес закріплення найкращих особин у популяції, що призводить до повільнішої збіжності алгоритму. Хоча підвищена мутація зменшує ризик передчасної збіжності, вона значно збільшує часові витрати.

Найбільш відчутний негативний ефект спостерігається для складних функцій, зокрема Rosenbrock, що підтверджує високу чутливість часової ефективності генетичного алгоритму до параметрів мутації.

3.6 Висновки

За результатами проведених експериментів у межах шести конфігурацій генетичного алгоритму можна зробити низку узагальнених висновків щодо впливу основних параметрів на його часову ефективність.

У базовій конфігурації було отримано опорні значення часу виконання та кількості використаних поколінь для різних тестових функцій. Ці результати слугували еталоном для подальших порівнянь і дозволили встановити залежність між складністю цільової функції та обчислювальними витратами алгоритму. Зокрема, для простої унімодальної функції Sphere генетичний алгоритм демонстрував найменший час виконання та стабільну збіжність. Для мультимодальної функції Rastrigin і функції Rosenbrock зі складною геометрією простору пошуку часові витрати та кількість поколінь до зупинки були суттєво вищими, що узгоджується з їхніми теоретичними властивостями.

У другій конфігурації, де було збільшено максимальну кількість поколінь, встановлено, що цей параметр безпосередньо впливає на загальний час виконання алгоритму. Зі зростанням допустимої кількості поколінь збільшувалася і тривалість роботи генетичного алгоритму, особливо для складних тестових функцій. Водночас для простих задач алгоритм часто завершував роботу значно раніше за досягнення граничного значення, що

свідчить про ефективну збіжність і обмежений вплив цього параметра на часові витрати у таких випадках.

У третій конфігурації, де було збільшено розмір популяції при фіксованій кількості поколінь, чітко зафіксовано зростання часу виконання алгоритму для всіх досліджуваних функцій у порівнянні з базовою конфігурацією. Це пояснюється тим, що зі збільшенням кількості особин у популяції зростає обсяг обчислень функції пристосованості в межах кожного покоління. При цьому кількість поколінь до зупинки не зросла пропорційно розміру популяції, оскільки критерієм завершення експерименту було досягнення максимальної кількості поколінь, а не умова стагнації. Таким чином, збільшення популяції вплинуло насамперед на часові витрати, а не на швидкість збіжності алгоритму в термінах кількості поколінь.

У четвертій конфігурації, де було збільшено розмірність задачі, встановлено, що зростання кількості змінних призводить до істотного збільшення часу виконання генетичного алгоритму. Для всіх тестових функцій спостерігалося підвищення обчислювальних витрат, а також збільшення кількості поколінь до зупинки. Це пов’язано з розширенням простору пошуку, у якому алгоритму необхідно здійснювати оптимізацію, що ускладнює процес знаходження глобального мінімуму, особливо для складних функцій типу Rosenbrock.

У п’ятій конфігурації, де змінювалася лише ймовірність кросинговеру, було виявлено, що підвищення інтенсивності кросинговеру призводить до помірного зростання часу виконання алгоритму та кількості поколінь до зупинки. Надмірно високий рівень кросинговеру ускладнює закріплення найкращих особин у популяції, що уповільнює процес збіжності. Найбільш чутливими до цього параметра виявилися складні тестові функції, зокрема Rosenbrock.

У шостій конфігурації, де було збільшено параметри мутації, зафіксовано найбільш виражене зростання часу виконання алгоритму серед усіх конфігурацій, у яких змінювався лише один параметр. Підвищена ймовірність мутації та

більший параметр σ призводили до збільшення кількості випадкових змін у популяції, що з одного боку зменшувало ризик передчасної збіжності, а з іншого — суттєво уповільнювало процес стабілізації розв’язків. Особливо помітний негативний ефект спостерігався для функції Rosenbrock, яка є чутливою до надмірної випадковості.

Загалом, результати експериментів підтверджують, що часова ефективність генетичного алгоритму визначається сукупною дією його параметрів, причому найбільший вплив мають розмір популяції, кількість поколінь та параметри мутації. Вплив окремих параметрів суттєво залежить від складності задачі оптимізації. Для простих унімодальних функцій доцільно використовувати помірні значення параметрів, тоді як для складних мультимодальних задач необхідний обережний баланс між дослідженням простору пошуку та обчислювальними витратами.

Отримані результати створюють основу для подальших досліджень, спрямованих на розробку адаптивних стратегій налаштування параметрів генетичних алгоритмів з метою підвищення їх часової ефективності.

ЗАГАЛЬНІ ВИСНОВКИ

У даній магістерській роботі було проведено комплексне дослідження часової ефективності генетичних алгоритмів залежно від вибору їх параметрів та складності задач оптимізації. Для досягнення поставленої мети виконано теоретичний аналіз, розроблено власний програмний інструмент та проведено серію систематичних експериментів із використанням стандартних тестових функцій.

У першому розділі роботи розглянуто теоретичні засади генетичних алгоритмів як класу стохастичних еволюційних методів оптимізації. Проаналізовано основні компоненти генетичного алгоритму, зокрема оператори селекції, кросинговеру та мутації, способи кодування рішень, функції пристосованості та критерії зупинки. Показано, що вибір параметрів алгоритму суттєво впливає на його поведінку, здатність уникати локальних мінімумів та ефективно знаходити глобальні оптимуми. Окрему увагу приділено аналізу часової складності генетичних алгоритмів та чинникам, що визначають обчислювальні витрати, а також ролі тестових функцій Sphere, Rastrigin і Rosenbrock як універсальних інструментів для експериментального дослідження.

У другому розділі здійснено аналіз і порівняння існуючих програмних засобів для реалізації еволюційних і генетичних алгоритмів. Встановлено, що більшість наявних бібліотек та фреймворків орієнтовані передусім на отримання оптимального розв’язку, тоді як засоби детального аналізу часових характеристик алгоритмів є обмеженими або відсутніми. Це обґрунтувало доцільність створення власної програмної системи, спеціалізованої на зборі, збереженні та візуалізації часових метрик роботи генетичних алгоритмів. У

розділі сформульовано вимоги до такого програмного інструменту та визначено перелік показників, необхідних для коректного експериментального аналізу.

У третьому розділі розроблено та реалізовано програмний інструмент мовою Python для дослідження часової ефективності генетичних алгоритмів. Програма забезпечує реалізацію класичного генетичного алгоритму з можливістю гнучкого налаштування параметрів, підтримує стандартні тестові функції, виконує багаторазові незалежні запуски алгоритму, автоматично вимірює час виконання та зберігає результати у форматі CSV. На основі зібраних даних реалізовано побудову графіків середнього часу виконання, кількості поколінь до зупинки та аналіз варіативності результатів.

У межах експериментальної частини було досліджено шість конфігурацій генетичного алгоритму, що дозволило встановити вплив ключових параметрів на часову ефективність:

- у базовій конфігурації отримано опорні значення часу виконання та кількості поколінь до зупинки, які підтвердили залежність обчислювальних витрат від складності цільової функції;
- збільшення максимальної кількості поколінь призводить до зростання часу виконання, особливо для складних функцій, тоді як для простих задач алгоритм часто завершується раніше за граничне значення;
- збільшення розміру популяції суттєво підвищує часові витрати через зростання кількості обчислень функції пристосованості, при цьому кількість поколінь до зупинки змінюється незначно;
- зростання розмірності задачі істотно ускладнює процес оптимізації, що проявляється у збільшенні як часу виконання, так і кількості поколінь до зупинки;
- підвищення ймовірності кросинговеру викликає помірне уповільнення збіжності алгоритму, особливо для складних мультимодальних функцій;

- збільшення параметрів мутації призводить до найбільшого зростання часу виконання серед конфігурацій зі зміною одного параметра, що пов’язано з підвищеною випадковістю та уповільненням стабілізації популяції.

Загалом встановлено, що часова ефективність генетичного алгоритму визначається сукупною дією його параметрів, при цьому найбільший вплив мають розмір популяції, кількість поколінь, розмірність задачі та параметри мутації. Вплив окремих параметрів істотно залежить від складності задачі оптимізації: для простих унімодальних функцій доцільно використовувати помірні значення параметрів, тоді як для складних мультимодальних задач необхідний обережний компроміс між якістю пошуку та обчислювальними витратами.

Отримані результати підтверджують досягнення мети магістерської роботи та демонструють практичну цінність розробленого програмного інструменту. Запропонований підхід може бути використаний для подальших досліджень, зокрема для розробки адаптивних стратегій налаштування параметрів генетичних алгоритмів з метою підвищення їх часової ефективності.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Bäck, T. Evolutionary Algorithms in Theory and Practice. – Oxford : Oxford University Press, 1996. – 328 p.
2. Bäck, T. Selective Pressure in Evolutionary Algorithms // Evolutionary Computation. – 1994. – Vol. 2, No. 2. – P. 155–169.
3. Bäck, T., Fogel, D. B., Michalewicz, Z. Handbook of Evolutionary Computation. – Bristol : IOP Publishing, 1997. – 1120 p.
4. Coello Coello, C. A., Lamont, G. B., Van Veldhuizen, D. A. Evolutionary Algorithms for Solving Multi-Objective Problems. – New York : Springer, 2007. – 800 p.
5. Deb, K. Multi-Objective Optimization Using Evolutionary Algorithms. – Chichester : Wiley, 2001. – 518 p.
6. Eiben, A. E., Hinterding, R., Michalewicz, Z. Parameter Control in Evolutionary Algorithms // IEEE Transactions on Evolutionary Computation. – 1999. – Vol. 3, No. 2. – P. 124–141.
7. Eiben, A. E., Smith, J. E. Introduction to Evolutionary Computing. – Berlin : Springer, 2003. – 299 p.
8. Fogel, D. B. Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. – New York : IEEE Press, 2006. – 350 p.
9. Goldberg, D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. – Reading, MA : Addison-Wesley, 1989. – 412 p.
10. Holland, J. H. Adaptation in Natural and Artificial Systems. – Ann Arbor : University of Michigan Press, 1975. – 211 p.
11. Jansen, T. Analyzing Evolutionary Algorithms: The Computer Science Perspective. – Berlin : Springer, 2013. – 215 p.
12. Luke, S. Essentials of Metaheuristics. – Raleigh : Lulu Press, 2013. – 250 p.
13. Michalewicz, Z. Genetic Algorithms + Data Structures = Evolution Programs. – Berlin : Springer, 1996. – 387 p.
14. Mitchell, M. An Introduction to Genetic Algorithms. – Cambridge : MIT Press, 1998. – 221 p.
15. Neri, F., Cotta, C., Moscato, P. Handbook of Memetic Algorithms. – Berlin : Springer, 2012. – 690 p.

16. Sivanandam, S. N., Deepa, S. N. Introduction to Genetic Algorithms. – Berlin : Springer, 2008. – 442 p.
17. Talbi, E.-G. Metaheuristics: From Design to Implementation. – Hoboken : Wiley, 2009. – 624 p.
18. Whitley, D. A Genetic Algorithm Tutorial // Statistics and Computing. – 1994. – Vol. 4. – P. 65–85.
19. Yang, X.-S. Nature-Inspired Optimization Algorithms. – London : Elsevier, 2014. – 296 p.
20. Luke, S. The ECJ Owner's Manual: A User Manual for the ECJ Evolutionary Computation Library [Електронний ресурс]. – Department of Computer Science, George Mason University, 2010. – Режим доступу: http://cs.gmu.edu/~eclab/projects/ecj/docs/manual/manual.pdf
21. MathWorks. MATLAB: середовище для технічних обчислень [Електронний ресурс]. – Режим доступу: https://www.mathworks.com/products/matlab.html
22. Zitzler, E., Deb, K., Thiele, L. Comparison of Multiobjective Evolutionary Algorithms // Evolutionary Computation. – 2000. – Vol. 8, No. 2. – P. 173–195.
23. Андрющенко, В. О. Методи та засоби програмної інженерії [Текст]. – Дніпро : ДНУЗТ, 2018. – 210 с.
24. Бабак, В. П., Шифрін, М. М. Методи оптимізації та прийняття рішень [Текст]. – Київ : Либідь, 2017. – 384 с.
25. Горбова, О. В. Проєктування програмних систем [Текст]. – Дніпро : ДНУЗТ, 2020. – 198 с.
26. Іванов, О. П. Моделювання та аналіз алгоритмів [Текст]. – Дніпро : ДНУЗТ, 2019. – 184 с.
27. Кононюк, А. Ю. Нейронні мережі і генетичні алгоритми [Текст]. – Київ : «Корнійчук», 2008. – 470 с.
28. Кононюк, А. Ю. Інтелектуальні інформаційні системи [Текст]. – Київ : Видавництво КПІ ім. Ігоря Сікорського, 2018. – 420 с.
29. Кононюк, А. Ю. Методи штучного інтелекту [Текст]. – Київ : КПІ ім. Ігоря Сікорського, 2019. – 356 с.
30. Кононюк, А. Ю. Еволюційні алгоритми та генетичні методи оптимізації [Текст]. – Київ : НТУУ «КПІ», 2020. – 312 с.

31. Литвиненко, В. І. Інтелектуальні системи та технології [Текст]. – Львів : Львівська політехніка, 2018. – 296 с.

32. Мельник, А. О. Алгоритми оптимізації в інформаційних системах [Текст]. – Київ : Академвидав, 2021. – 304 с.

33. Нечай, В. Я. Алгоритмічні методи оптимізації [Текст]. – Дніпро : ДНУЗТ, 2021. – 220 с.

34. Павлов, В. М. Основи штучного інтелекту [Текст]. – Харків : ХНУРЕ, 2019. – 280 с.

35. Петренко, О. М. Сучасні методи оптимізації в інформаційних системах [Текст]. – Київ : Наукова думка, 2020. – 256 с.

36. Соловйов, В. М. Штучний інтелект та еволюційні обчислення [Текст]. – Київ : Освіта України, 2021. – 312 с.

37. Тимошенко, І. В. Методи еволюційної оптимізації [Текст]. – Харків : ХНУРЕ, 2022. – 288 с.

38. Шинкаренко, В. І., Горбова, О. В., Іванов, О. П., Андрющенко, В. О., Нечай, В. Я. Інженерія програмного забезпечення [Текст] : навчальний посібник. – Дніпро : Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна, 2019. – 140 с.

39. Яковенко, М. С. Еволюційні алгоритми в задачах оптимізації [Текст]. – Київ : КПІ ім. Ігоря Сікорського, 2021. – 260 с.

ДОДАТОК А
ТЕХНІЧНЕ ЗАВДАННЯ

ЗАТВЕРДЖУЮ

Проректор Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

ПРОГРАМНИЙ ІНСТРУМЕНТ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ ГЕНЕТИЧНИХ АЛГОРИТМІВ

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01540-01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
Керівник розробки
Вадим АНДРЮЩЕНКО
Виконавець
Тетяна МИХАЙЛОВА
Нормоконтролер
Світлана ВОЛКОВА

2026

ЗАТВЕРДЖЕНО

44165850.01540-01-ЛЗ

ПРОГРАМНИЙ ІНСТРУМЕНТ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ ГЕНЕТИЧНИХ АЛГОРИТМІВ

Технічне завдання

44165850.01540-01

Листів 16

2026

ЗМІСТ

1 ВСТУП

У сучасних інформаційних системах та інженерних задачах оптимізація відіграє ключову роль у прийнятті ефективних рішень. Зі зростанням складності задач, розмірності даних та обчислювальних вимог дедалі частіше застосовуються еволюційні методи оптимізації, зокрема генетичні алгоритми, які дозволяють знаходити наближені оптимальні розв’язки у складних, негладких та багатомодальних просторах пошуку.

Генетичні алгоритми широко використовуються у штучному інтелекті, машинному навчанні, логістиці, фармакології, інженерному проєктуванні та інших галузях. Їхньою перевагою є здатність працювати в умовах невизначеності та відсутності аналітичного опису задачі. Проте разом із цим виникає важлива практична проблема — високі обчислювальні витрати та непередбачуваність часу виконання, що пов’язано зі стохастичною природою алгоритмів та значною кількістю параметрів.

У реальних прикладних задачах не завжди є можливість збільшувати обчислювальні ресурси або час роботи алгоритму. Тому постає необхідність дослідження часової ефективності генетичних алгоритмів та визначення того, які саме параметри найбільше впливають на час виконання, і як їх доцільно підбирати залежно від складності задачі оптимізації.

Для проведення такого аналізу необхідне спеціалізоване програмне забезпечення, яке дозволяє:

- реалізувати класичний генетичний алгоритм;
- багаторазово запускати його з фіксованими та змінними параметрами;
- вимірювати час виконання та кількість поколінь до зупинки;
- накопичувати результати експериментів і наочно аналізувати їх за допомогою графіків.

Метою програмного забезпечення є розробка програмного інструменту для дослідження часової ефективності генетичних алгоритмів шляхом експериментального аналізу впливу основних параметрів алгоритму (кількості поколінь, розміру популяції, параметрів мутації та кросинговеру) на час виконання при оптимізації тестових функцій різної складності.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ ректора Українського державного університету науки і технологій “ Про затвердження тем та призначення керівників дипломних проектів ” № 1401ст від 02.10.2025.

Тема проекту: «Дослідження часової ефективності генетичних алгоритмів».

Керівник дипломного проекту: Андрющенко Вадим Олександрович

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення – програмне забезпечення повинно реалізовувати класичний генетичний алгоритм та забезпечувати можливість експериментального дослідження його часової ефективності. Програма має виконувати багаторазові запуски генетичного алгоритму з фіксованими та змінними параметрами, вимірювати час виконання, кількість використаних поколінь до зупинки та зберігати результати експериментів у табличному форматі. На основі отриманих даних програмне забезпечення повинно будувати графіки, що відображають залежність часу виконання алгоритму від складності задачі оптимізації та значень параметрів генетичного алгоритму.

Експлуатаційне призначення – надання користувачам (дослідникам, студентам, розробникам) програмного інструменту для аналізу впливу параметрів генетичних алгоритмів на час їх виконання. Програмне забезпечення може використовуватися для навчальних і науково-дослідних цілей, зокрема для підбору параметрів генетичних алгоритмів у прикладних задачах оптимізації, а також для проведення експериментів і формування висновків щодо ефективності алгоритму для задач різної складності.

.

4 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Програмне забезпечення повинно:

- реалізовувати класичний генетичний алгоритм для задач оптимізації з неперервною областю пошуку;
- забезпечувати можливість багаторазового запуску генетичного алгоритму з фіксованими та змінними параметрами;
- вимірювати час виконання генетичного алгоритму для кожного запуску;
- визначати кількість поколінь, використаних алгоритмом до досягнення критерію зупинки;
- зберігати результати експериментів у CSV-файлі для подальшого аналізу;
- будувати графіки, що відображають:
    - середній час виконання генетичного алгоритму для різних тестових функцій;
    - середню кількість поколінь до зупинки алгоритму;
    - зміну часу виконання алгоритму за окремими запусками;
- забезпечувати можливість порівняння часової ефективності генетичного алгоритму для різних конфігурацій параметрів.

Вхідними даними є:

- параметри генетичного алгоритму:
    - розмір популяції;
    - максимальна кількість поколінь;
    - ймовірність кросинговеру;
    - ймовірність мутації;
    - параметр мутації (σ);

- розмірність задачі;
- критерій зупинки;

- вибір тестової функції оптимізації (Sphere, Rastrigin, Rosenbrock);
- кількість незалежних запусків алгоритму.

Вхідні параметри задаються безпосередньо в програмному коді або через конфігураційний об’єкт генетичного алгоритму.

Вихідними даними є:

- CSV-файл з результатами експериментів, що містить:
  - назву тестової функції;
  - номер запуску;
  - час виконання алгоритму;
  - кількість використаних поколінь;
  - значення знайденого розв’язку;
- графіки часової ефективності генетичного алгоритму;
- узагальнені статистичні показники (середні значення часу виконання та кількості поколінь).

4.2 Вимоги до надійності

Вимоги до надійності наступні:

- наявність архівної копії тексту програми на зовнішньому носії.

4.3 Вимоги експлуатації

Програмний продукт повинен використовуватись у приміщеннях які відповідають умовам роботи ЕОМ, а саме мають такі кліматичні, санітарні та гігієнічні умови, які відповідають ДНАОП 0.00-1.13-99 (таблиця 4.1).

Працювати з програмою може людина, що має навички роботи з персональним комп’ютером.

4.4 Вимоги до складу та параметрів технічних засобів

Продукт, що розробляється, повинен використовуватись на пристроях, що мають наступні характеристики:

- роздільна здатність дисплею – HD (1280x720);
- оперативна пам’ять – 2 ГБ;
- вбудована пам’ять – 4 ГБ;
- операційна система – MS WINDOWS 7 та наступні версії;
- середовище розробки – IDLE (Python 3.8 64 bit);
- частота процесора – 1.6 ГГц.

Таблиця 4.1 – Кліматичні умови

Пора року	Категорія робіт згідно з ГОСТ 12.01-005-88	Температура повітря, град.С	Відносна вологість повітря, %	Швидкість руху повітря, м/с
		Оптимальна	Оптимальна	Оптимальна
Холодна	легка-1-а	22-24	40-60	0,1
	легка-1-б	21-23	40-60	0,1
Тепла	легка-1-а	23-25	40-60	0,1
	легка-1-б	22-24	40-60	0,2

4.5 Вимоги до інформаційної та програмної сумісності

Програмний продукт розробляється для всіх видів операційних систем сімейства «MS Windows» починаючи від версії 7 та наступні версії.

4.6 Вимоги до маркування і упаковки

Упаковка програмного продукту, включаючи документацію повинна бути захищена від пошкоджень різного роду (механічних, кліматичних).

На зворотній стороні упаковки вказується розробник та його юридична адреса (дивись рисунок 4.1).

Додаток для ведення особистих фінансів

Михайлова Т.О.

УДУНТ, кафедра КІТ

49010, Україна, м. Дніпро, вул. Лазаряна, 2,

2025

Рисунок 4.1 – маркування додатку

На упаковці повинно бути вказана назва продукту, номер версії, мінімальні системні вимоги.

4.7 Вимоги до транспортування та зберігання

Транспортування повинне забезпечувати збереження програмного продукту його цілісність і запобігання несанкціонованого доступу до нього.

Вимогами до транспортування і зберігання додатку є:

- температура навколишнього повітря має перебувати в діапазоні від 21°С до - 25°С;

- вологість навколишнього повітря має перебувати в діапазоні від 40 % до 60%.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу документації мають входити:

- текст програми;
- керівництво користувача.

Вся документація до програмного продукту повинна задовольняти вимогам ДСТУ [1].

6. СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

В таблиці 6.1 приведені стадії та етапи розробки програми.

Таблиці 6.1 – Стадії та етапи розробки

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	01.09.25 – 15.09.25	
2	Аналіз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження	01.09.25 – 15.09.25	
3	Постановка задачі, технічне завдання	01.09.25 – 15.09.25	10%
4	Розробка інструментальних засобів дослідження	16.09.25 – 31.10.25	30%
5	Виконання досліджень	01.11.25 – 30.11.25	60%
6	Оформлення пояснювальної записки	01.12.25 – 11.01.26	100%
7	Розробка демонстраційних матеріалів	12.01.26 – 15.01.26	
8	Подання кваліфікаційної роботи до кафедри	15.01.26	
9	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	22.01.26	

7 ПОРЯДОК І КОНТРОЛЬ ПРИЙМАННЯ

Контроль виконання роботи здійснює керівник проекту Андрющенко В.О. Прийом здійснюється уповноваженою комісією.

8 ТЕХНІЧНО-ЕКОНОМІЧНІ ПОКАЗНИКИ

Технічно-економічні показники та їх розрахунок не були описані у документах, так як розробка програмного забезпечення несе в собі навчальний характер, а не комерційний.

9 БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем: методичні вказівки до дипломного проектування та лабораторних робіт/уклад.: Ю.М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с.

ДОДАТОК Б

ТЕКСТ ПРОГРАМИ

ЗАТВЕРДЖУЮ

Проректор Українського державного університету науки і технологій

Анатолій РАДКЕВИЧ

ПРОГРАМНИЙ ІНСТРУМЕНТ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ ГЕНЕТИЧНИХ АЛГОРИТМІВ

Текст програми

ЛИСТ ЗАТВЕРДЖЕННЯ

44165850.01540-01 12 01-ЛЗ

Представники

підприємства-розробника

Завідувач кафедри КІТ

Вадим ГОРЯЧКІН

Керівник розробки

Вадим АНДРЮЩЕНКО

Виконавець

Тетяна МИХАЙЛОВА

Нормоконтролер

Світлана ВОЛКОВА

2026

ЗАТВЕРДЖЕНО

44165850.01540-01 12 01-ЛЗ

ПРОГРАМНИЙ ІНСТРУМЕНТ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ ГЕНЕТИЧНИХ АЛГОРИТМІВ

Текст програми
44165850.01540-01 12 01

Листів 11

2026

АНОТАЦІЯ

Документ 44165850.1540-01 12 01 «Програмний інструмет дослідження часової ефективності генетичних алгоритмів. Текст програми» входить до складу програмної документації на програму, що описує реалізацію програмного інструменту для експериментального дослідження часової ефективності генетичних алгоритмів, а також обробку та візуалізацію результатів обчислювальних експериментів.

У даному документі представлено текст програмного коду розробленого програмного інструменту. Програма реалізована мовою програмування Python та призначена для дослідження часової ефективності генетичних алгоритмів. Розробка та тестування програмного забезпечення здійснювалися у середовищі Visual Studio Code з використанням інтерпретатора Python версії 3.8 (64-bit).

ЗМІСТ

1 Програма

1.1 main.py — головний файл запуску експериментів;

```
import csv
import time
from pathlib import Path

from ga.fitness import sphere, rastrigin, rosenbrock
from ga.ga_core import GAConfig, run_ga

RESULTS_DIR = Path("results")
RESULTS_DIR.mkdir(exist_ok=True)

def benchmark(func_name: str, fitness_fn, cfg: GAConfig, runs: int = 10):
    rows = []
    for r in range(runs):
        t0 = time.perf_counter()
        out = run_ga(fitness_fn, cfg, seed=1000 + r)
        t1 = time.perf_counter()

        rows.append({
            "function": func_name,
            "run": r + 1,
            "time_sec": (t1 - t0),
            "best_fitness": out["best_fitness"],
            "generations_done": out["generations_done"],
            "pop_size": cfg.pop_size,
            "generations": cfg.generations,
            "crossover_prob": cfg.crossover_prob,
            "mutation_prob": cfg.mutation_prob,
            "mutation_sigma": cfg.mutation_sigma,
            "dim": cfg.dim,
        })

    return rows

def save_csv(rows, filename: str):
    path = RESULTS_DIR / filename
    with path.open("w", newline="", encoding="utf-8") as f:
        writer = csv.DictWriter(f, fieldnames=list(rows[0].keys()))
        writer.writeheader()
        writer.writerows(rows)
    print(f" Збережено: {path}")
```

```
if __name__ == "__main__":
    cfg = GAConfig(
        dim=10,
        pop_size=150,
        generations=150,
        crossover_prob=0.8,
        mutation_prob=0.2,
        mutation_sigma=0.2,
        stagnation_k=60,
        tournament_k=3,
        bounds=(-5.0, 5.0),
    )

    all_rows = []
    all_rows += benchmark("Sphere", sphere, cfg, runs=10)
    all_rows += benchmark("Rastrigin", rastrigin, cfg, runs=10)
    all_rows += benchmark("Rosenbrock", rosenbrock, cfg, runs=10)

    save_csv(all_rows, "ga_time_benchmark.csv")
    print("Готово ")
```

1.2 fitness.py — реалізація функцій пристосованості;

```
import math
from typing import List

def sphere(x: List[float]) -> float:
    return sum(v * v for v in x)

def rastrigin(x: List[float]) -> float:
    n = len(x)
    return 10.0 * n + sum(v * v - 10.0 * math.cos(2.0 * math.pi * v) for v in x)

def rosenbrock(x: List[float]) -> float:
    return sum(100.0 * (x[i+1] - x[i] ** 2) ** 2 + (1.0 - x[i]) ** 2 for i in range(len(x) -
1))
```

1.3 ga_core.py — основна логіка генетичного алгоритму;

```
from __future__ import annotations
import random
from dataclasses import dataclass
from typing import Callable, List, Tuple
```

```
FitnessFn = Callable[[List[float]], float]

@dataclass
class GAConfig:
    dim: int = 10
    bounds: Tuple[float, float] = (-5.0, 5.0)
    pop_size: int = 50
    generations: int = 200
    tournament_k: int = 3
    crossover_prob: float = 0.9
    mutation_prob: float = 0.1   # ймовірність мутації гена
    mutation_sigma: float = 0.1  # сила мутації
    stagnation_k: int = 30       # зупинка якщо k поколінь нема покращення

def _clip(v: float, lo: float, hi: float) -> float:
    return max(lo, min(hi, v))

def _init_population(cfg: GAConfig) -> List[List[float]]:
    lo, hi = cfg.bounds
    return [[random.uniform(lo, hi) for _ in range(cfg.dim)] for _ in
range(cfg.pop_size)]

def _tournament_select(pop: List[List[float]], fit: List[float], k: int) -> List[float]:
    best_idx = None
    for _ in range(k):
        i = random.randrange(len(pop))
        if best_idx is None or fit[i] < fit[best_idx]:
            best_idx = i
    return pop[best_idx]

def _crossover(p1: List[float], p2: List[float]) -> Tuple[List[float], List[float]]:
    # арифметичний кросовер
    alpha = random.random()
    c1 = [alpha * a + (1 - alpha) * b for a, b in zip(p1, p2)]
    c2 = [(1 - alpha) * a + alpha * b for a, b in zip(p1, p2)]
    return c1, c2

def _mutate(x: List[float], cfg: GAConfig) -> List[float]:
    lo, hi = cfg.bounds
    y = x[:]
    for i in range(len(y)):
        if random.random() < cfg.mutation_prob:
            y[i] = _clip(y[i] + random.gauss(0.0, cfg.mutation_sigma), lo, hi)
```

```
    return y

def run_ga(fitness_fn: FitnessFn, cfg: GAConfig, seed: int | None = None) -> dict:
    if seed is not None:
        random.seed(seed)

    pop = _init_population(cfg)
    fit = [fitness_fn(ind) for ind in pop]

    best_val = min(fit)
    best_ind = pop[fit.index(best_val)]
    no_improve = 0

    for gen in range(cfg.generations):
        new_pop: List[List[float]] = []

        while len(new_pop) < cfg.pop_size:
            p1 = _tournament_select(pop, fit, cfg.tournament_k)
            p2 = _tournament_select(pop, fit, cfg.tournament_k)

            if random.random() < cfg.crossover_prob:
                c1, c2 = _crossover(p1, p2)
            else:
                c1, c2 = p1[:], p2[:]

            c1 = _mutate(c1, cfg)
            c2 = _mutate(c2, cfg)

            new_pop.append(c1)
            if len(new_pop) < cfg.pop_size:
                new_pop.append(c2)

        pop = new_pop
        fit = [fitness_fn(ind) for ind in pop]

        cur_best = min(fit)
        if cur_best < best_val:
            best_val = cur_best
            best_ind = pop[fit.index(cur_best)]
            no_improve = 0
        else:
            no_improve += 1

        if no_improve >= cfg.stagnation_k:
```

```
            break

    return {
        "best_fitness": best_val,
        "best_solution": best_ind,
        "generations_done": gen + 1,
    }
```

1.4 plots.py — побудова графіків за результатами експериментів;

```
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path

# шлях до CSV з результатами
DATA_PATH = Path("results/ga_time_benchmark.csv")
# зчитуємо дані
df = pd.read_csv(DATA_PATH)
# 1. Середній час виконання
mean_time = df.groupby("function")["time_sec"].mean()

plt.figure()
mean_time.plot(kind="bar")
plt.title("Середній час виконання генетичного алгоритму")
plt.ylabel("Час, сек")
plt.xlabel("Функція")
plt.grid(True)
plt.tight_layout()
plt.show()
# 2. Середня кількість поколінь
mean_gen = df.groupby("function")["generations_done"].mean()

plt.figure()
mean_gen.plot(kind="bar")
plt.title("Середня кількість поколінь до зупинки")
plt.ylabel("Кількість поколінь")
plt.xlabel("Функція")
plt.grid(True)
plt.tight_layout()
plt.show()
# 3. Час виконання по кожному запуску
for func in df["function"].unique():
    subset = df[df["function"] == func]
    plt.figure()
```

```
plt.plot(subset["run"], subset["time_sec"], marker="o")
plt.title(f"Час виконання по запусках: {func}")
plt.xlabel("Номер запуску")
plt.ylabel("Час, сек")
plt.grid(True)
plt.tight_layout()
 plt.show()
```

ДОДАТОК В

КЕРІВНИЦТВО КОРИСТУВАЧА

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ

Проректор Українського державного університету науки і технологій

Анатолій РАДКЕВИЧ

ПРОГРАМНИЙ ІНСТРУМЕНТ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ ГЕНЕТИЧНИХ АЛГОРИТМІВ

Керівництво користувача

ЛИСТ ЗАТВЕРДЖЕННЯ

44165850.01540-01 І3 01-ЛЗ

Представники

підприємства-розробника

Завідувач кафедри КІТ

Вадим ГОРЯЧКІН

Керівник розробки

Вадим АНДРЮЩЕНКО

Виконавець

Тетяна МИХАЙЛОВА

Нормоконтролер

Світлана ВОЛКОВА

2026

ЗАТВЕРДЖЕНО

44165850.01540-01 І3 01-ЛЗ

ПРОГРАМНИЙ ІНСТРУМЕНТ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ ГЕНЕТИЧНИХ АЛГОРИТМІВ

Керівництво користувача

44165850.1540-01 І3 01

Листів 17

2026

АНОТАЦІЯ

Документ 44165850.1540-01 12 01 « Програмний інструментдослідження часової ефективності генетичних алгоритмів. Керівництво користувача» входить до складу програмної документації на програму, що описує реалізацію програмного інструменту для експериментального дослідження часової ефективності генетичних алгоритмів, а також обробку та візуалізацію результатів обчислювальних експериментів.

У даному документі представлено текст програмного коду розробленого програмного інструменту. Програма реалізована мовою програмування Python та призначена для дослідження часової ефективності генетичних алгоритмів. Розробка та тестування програмного забезпечення здійснювалися у середовищі Visual Studio Code з використанням інтерпретатора Python версії 3.8 (64-bit).

.

ЗМІСТ

1. ВСТУП

Генетичні алгоритми є стохастичними методами оптимізації, що імітують процеси природної еволюції та широко застосовуються для розв’язання складних оптимізаційних задач із негладким або багатомодальним простором пошуку. Особливістю таких алгоритмів є залежність їхньої поведінки та ефективності від великої кількості параметрів, а також від випадкових чинників, що ускладнює аналітичну оцінку їх продуктивності.

У сучасних умовах розвитку інформаційних технологій та штучного інтелекту зростає потреба не лише у використанні генетичних алгоритмів, але й у дослідженні їх часової ефективності. Час виконання алгоритму є критично важливим показником, особливо для задач великої розмірності, задач реального часу та прикладних систем, де обчислювальні ресурси є обмеженими. Тому актуальним є створення програмних засобів, які дозволяють експериментально оцінювати вплив параметрів генетичних алгоритмів на тривалість їх роботи.

Метою розробленого програмного забезпечення є проведення експериментального дослідження часової ефективності генетичних алгоритмів на тестових функціях оптимізації шляхом багаторазових запусків алгоритму з різними параметричними конфігураціями, збереження результатів у табличному вигляді та їх подальшої візуалізації у вигляді графіків.

Програмний інструмент дозволяє:

- запускати генетичний алгоритм з заданими параметрами;
- вимірювати час виконання алгоритму та кількість використаних поколінь;
- зберігати результати експериментів у CSV-файлі;
- будувати графіки середнього часу виконання, кількості поколінь до зупинки та варіативності результатів між запусками.

Працювати з програмою може користувач, який має базові навички роботи з персональним комп’ютером та початкові знання мови програмування Python. Для коректної роботи з програмою користувач повинен мати встановлене середовище виконання Python та ознайомитися з даним керівництвом користувача перед початком експлуатації програмного забезпечення.

2. ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

Програма призначена для експериментального дослідження часової ефективності генетичних алгоритмів шляхом багаторазового запуску алгоритму з різними параметричними конфігураціями на тестових функціях оптимізації.

Основним призначенням програмного забезпечення є:

- вимірювання часу виконання генетичного алгоритму;
- аналіз кількості поколінь, необхідних для досягнення критерію зупинки;
- порівняння впливу параметрів алгоритму (розміру популяції, кількості поколінь, ймовірностей кросинговеру та мутації, розмірності задачі) на часові витрати;
- збереження результатів експериментів у структурованому вигляді для подальшого аналізу та візуалізації.

Програма використовується для дослідження поведінки генетичних алгоритмів на класичних тестових функціях оптимізації (Sphere, Rastrigin, Rosenbrock), які характеризуються різною складністю простору пошуку.

Вхідними даними для програми є:

- параметри генетичного алгоритму (розмір популяції, максимальна кількість поколінь, ймовірність кросинговеру, ймовірність мутації, параметр мутації, розмірність задачі, критерій зупинки);
- вибір тестової функції оптимізації;
- кількість незалежних запусків алгоритму.

Результати виконання програми автоматично зберігаються у CSV-файлі, який містить інформацію про:

- номер запуску;
- назву тестової функції;
- час виконання алгоритму;
- кількість поколінь до зупинки.

Програмне забезпечення призначене для використання у навчальних, науково-дослідних та експериментальних цілях, зокрема при дослідженні часової ефективності еволюційних методів оптимізації.

Вимоги до складу та параметрів технічних засобів:

- роздільна здатність дисплею – HD (1280x720);
- оперативна пам’ять – 2 ГБ;
- вбудована пам’ять – 4 ГБ;
- операційна система – MS WINDOWS 7 та наступні версії;
- частота процесора – 1.6 ГГц.

Вимоги до інформаційної та програмної сумісності:

- операційна система - MS WINDOWS 7 та наступні версії;

- середовище розробки Visual Studio Code з використанням інтерпретатора Python версії 3.8 (64-bit).

3. ПІДГОТОВКА ДО РОБОТИ

Для підготовки програмного забезпечення до роботи необхідно виконати такі дії:

1. Скопіювати каталог з програмними файлами (наприклад, GA_Time_Analyzer) з носія інформації (флеш-пам’яті або іншого джерела) на жорсткий диск персонального комп’ютера. Рекомендоване розташування — будь-яка робоча директорія користувача (наприклад, C:\Users\…).
2. Переконатися, що у каталозі програми наявні всі необхідні файли, зокрема:
   - main.py — головний файл запуску експериментів;
   - ga_core.py — реалізація генетичного алгоритму;
   - fitness_functions.py — тестові функції оптимізації;
   - plots.py — модуль побудови графіків;
   - каталог results/ — для збереження CSV-файлів з результатами експериментів.
3. Встановити на комп’ютері інтерпретатор мови Python версії не нижче Python 3.8.
4. Переконатися у наявності необхідних бібліотек Python:
   - numpy
   - pandas
   - matplotlib

4. ОПИС ОПЕРАЦІЙ

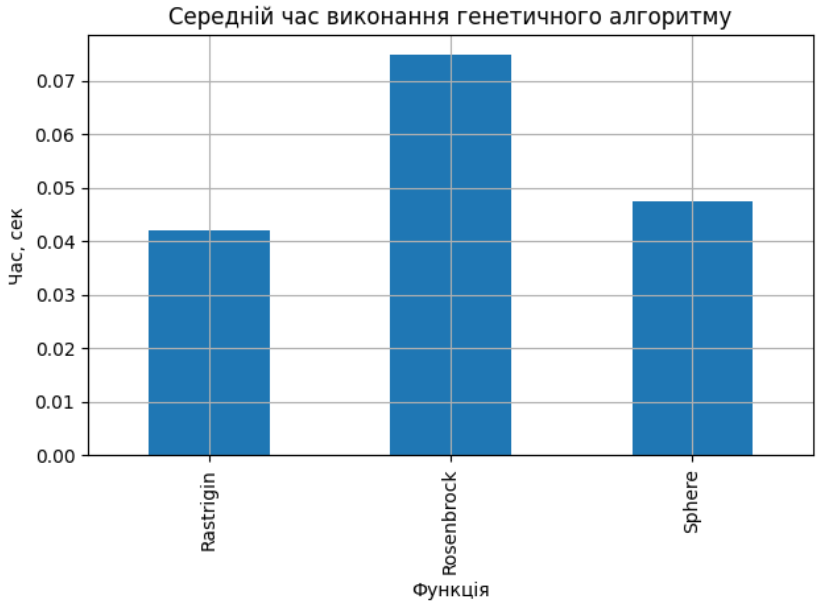
У програмному середовищі розробки Visual Studio Code або IDLE (Python 3.8 64-bit) необхідно відкрити головний файл програми main.py, який знаходиться у каталозі програмного інструменту для дослідження часової ефективності генетичних алгоритмів.

Після запуску програми main.py виконується серія обчислювальних експериментів, у межах яких генетичний алгоритм багаторазово запускається для різних тестових функцій оптимізації з наперед заданими параметрами. У ході виконання програми автоматично здійснюється:

- запуск генетичного алгоритму для кожної тестової функції;
- багаторазове повторення експериментів (10 запусків) з однаковими параметрами;
- вимірювання часу виконання алгоритму;
- фіксація кількості використаних поколінь до зупинки;
- збереження результатів у CSV-файл;
- побудова графіків часової ефективності.

На рисунку 4.1 представлено стовпчикову діаграму середнього часу виконання генетичного алгоритму для кожної тестової функції оптимізації (Sphere, Rastrigin, Rosenbrock). Значення обчислюються як середнє за 10 незалежних запусків алгоритму.

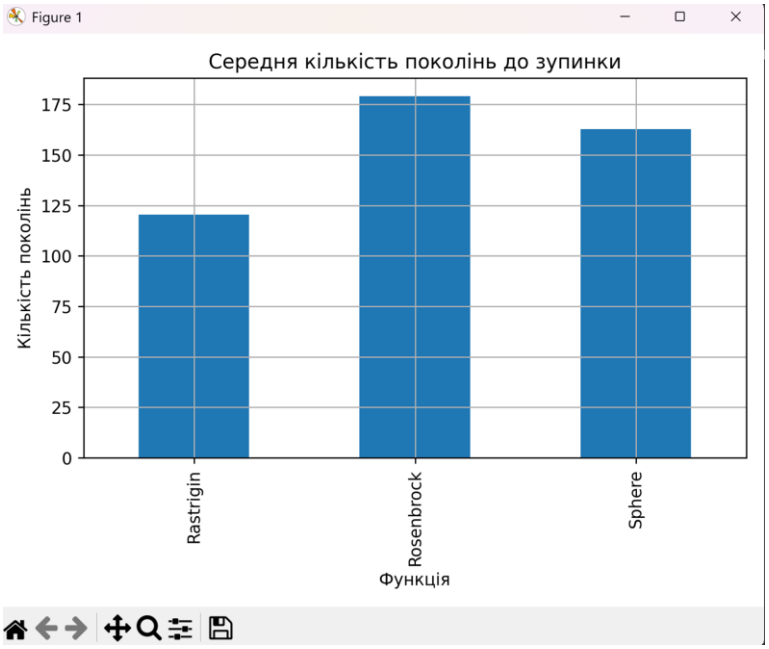
Графік дозволяє порівняти часові витрати алгоритму залежно від складності оптимізаційної задачі та підтверджує зростання часу виконання зі збільшенням складності функції.



Рисунку 4.1 – Середній час виконання генетичного алгоритму

На рисунку 4.2 показано середню кількість поколінь, які були виконані генетичним алгоритмом до досягнення критерію зупинки (досягнення максимальної кількості поколінь або стагнація).

Аналіз цього графіка дозволяє оцінити швидкість збіжності алгоритму для різних функцій та виявити функції, для яких пошук оптимального розв’язку є більш складним.



Рисунку 4.2 – Середня кількість поколінь до зупинки алгоритму

Рисунок 4.3 відображає залежність часу виконання алгоритму від номера запуску для функції Sphere. Графік демонструє стабільну поведінку алгоритму та незначну варіативність часу виконання, що пояснюється гладким і унімодальним простором пошуку цієї функції.

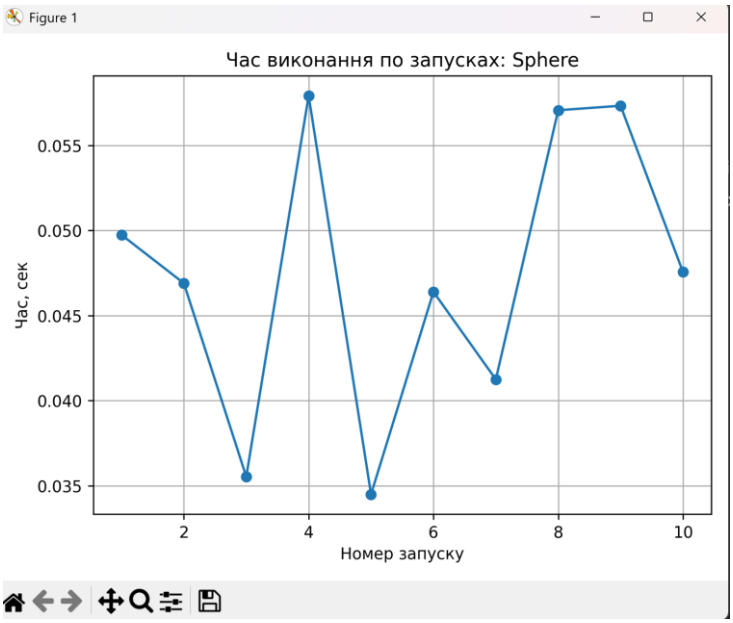


Рисунок 4.3 – Час виконання по запусках для функції Sphere

На рисунку 4.4 наведено результати вимірювання часу виконання алгоритму для мультимодальної функції Rastrigin. Спостерігається більша варіативність часу між окремими запусками, що пов’язано з наявністю великої кількості локальних мінімумів та стохастичною природою генетичного алгоритму.

Рисунок 4.4 – Час виконання по запусках для функції Rastrigin

Рисунок 4.5 ілюструє час виконання алгоритму для функції Rosenbrock, яка характеризується вузькою вигнутою долиною глобального мінімуму. Для цієї функції зафіксовано найбільші значення часу виконання та найбільший розкид результатів між окремими запусками.

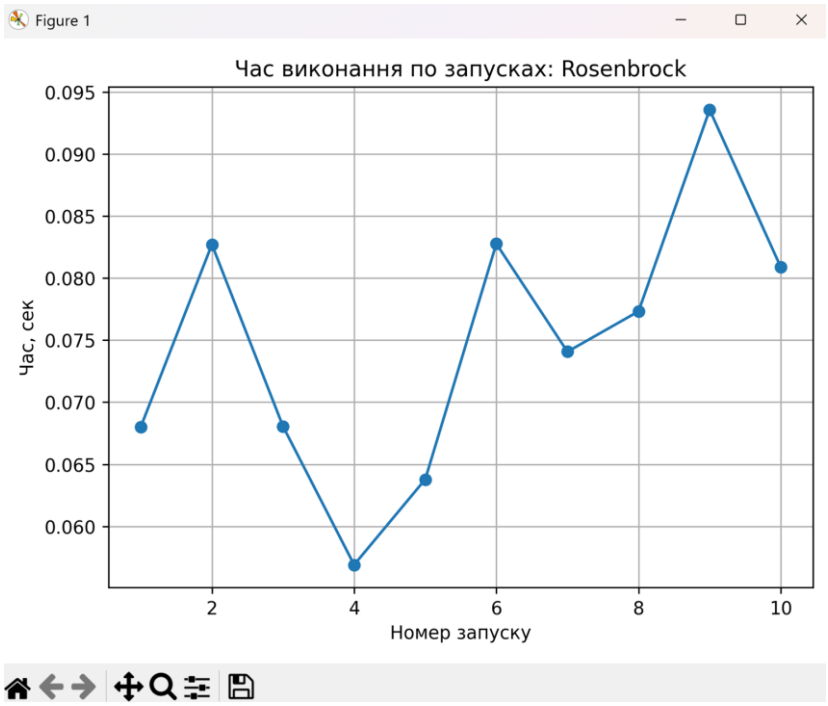


Рисунок 4.5 – Час виконання по запусках для функції Rosenbrock

На рисунку 4.6 показано приклад CSV-файлу, у який автоматично зберігаються результати експериментів. Файл містить такі дані:

- назву тестової функції;
- номер запуску;
- час виконання алгоритму;
- кількість поколінь до зупинки;
- значення параметрів конфігурації.

CSV-файл використовується як джерело даних для подальшого аналізу та побудови графіків у модулі plots.py.

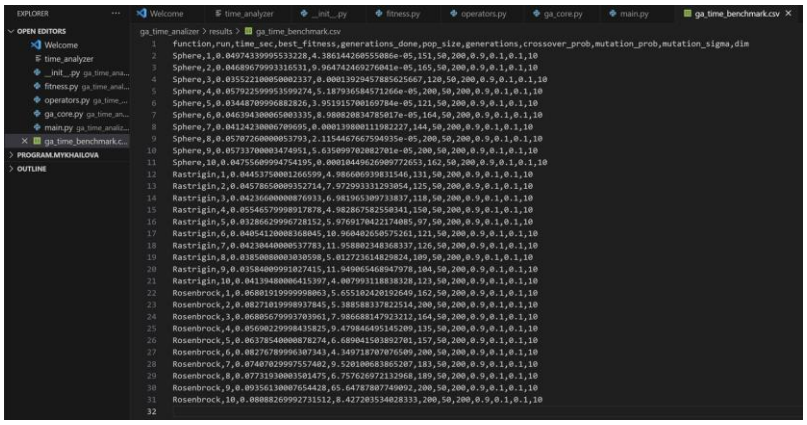


Рисунок 4.6 – CSV-файл з результатами експериментів

Таким чином, програма забезпечує повний цикл експериментального дослідження часової ефективності генетичних алгоритмів: від запуску та збору даних до їх збереження, візуалізації та подальшого аналізу.

5. АВАРІЙНІ СИТУАЦІЇ

У разі виникнення аварійної ситуації під час роботи програмного інструменту для дослідження часової ефективності генетичних алгоритмів необхідно виконати такі дії.

У випадку аварійного завершення роботи програми або виникнення помилки виконання слід закрити програмне середовище розробки Visual Studio Code або IDLE (Python 3.8 64-bit). Для цього необхідно натиснути кнопку закриття вікна програмного середовища (червоний хрестик у правому верхньому куті).

Після цього рекомендується повторно запустити програмне середовище розробки та відкрити головний файл програми main.py, який містить реалізацію генетичного алгоритму та запуск експериментів. Далі програму слід запустити повторно.

У випадку зависання програми під час виконання експериментів (наприклад, при великій кількості поколінь або розмірі популяції) рекомендується:

- зупинити виконання програми вручну;
- перевірити коректність заданих параметрів алгоритму;
- за необхідності зменшити обчислювальне навантаження (кількість поколінь, розмір популяції або кількість запусків);
- після цього повторно запустити програму.

Якщо програмне середовище не реагує на дії користувача, необхідно перезавантажити персональний комп’ютер, після чого знову запустити програмне середовище розробки та виконати програму.

Дотримання зазначених рекомендацій дозволяє відновити коректну роботу програмного інструменту та продовжити проведення експериментів з дослідження часової ефективності генетичних алгоритмів.

6. РЕКОМЕНДАЦІЇ ЩОДО ЗАСТОСУВАННЯ

З носія (флеш память) потрібно зкопіювати файли ts_pipe_9.py та Release.csv в папку C:\Users.

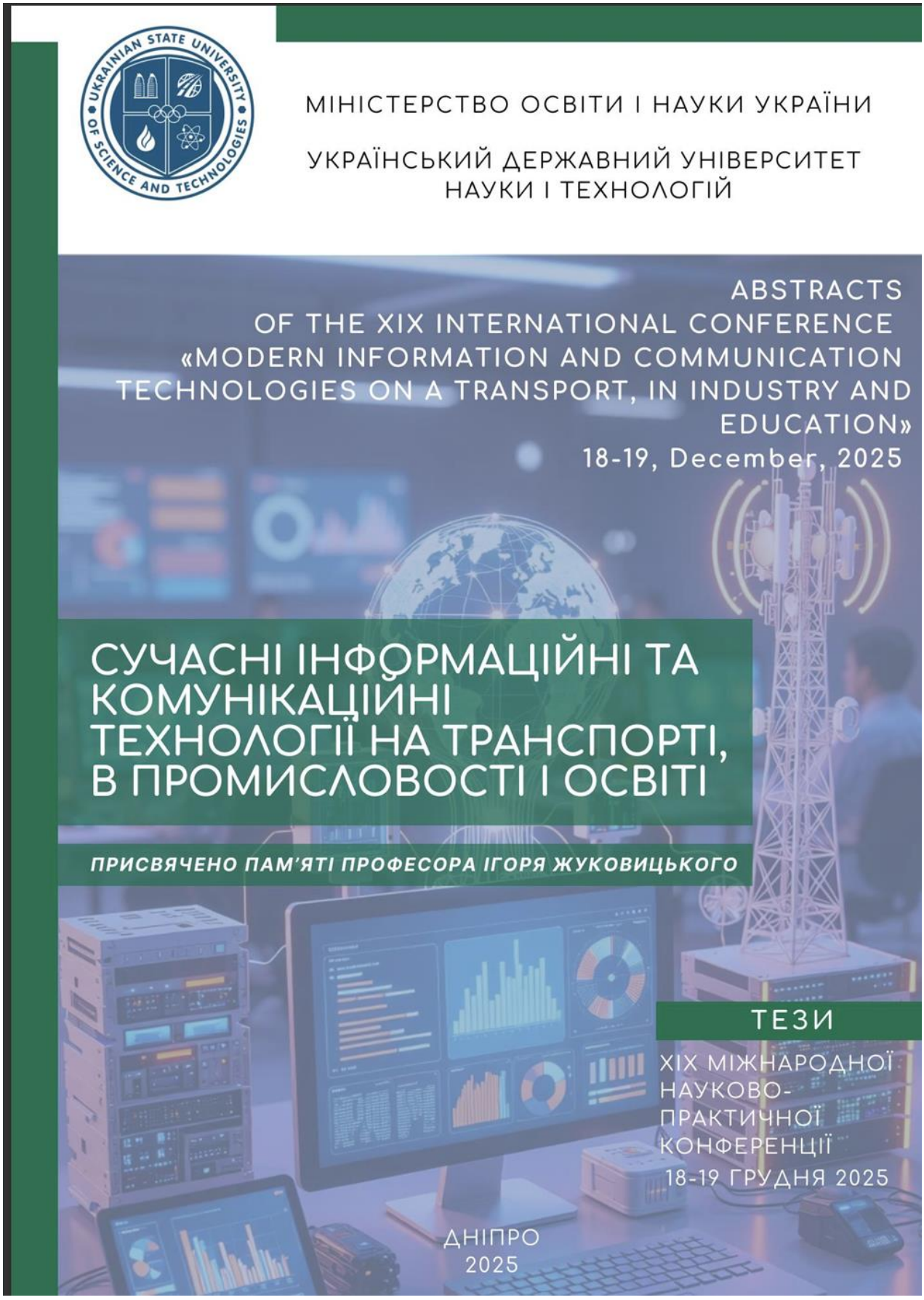
Вхідний набір даних має мати дані за 60 місяців виробництва металургійної продукції у форматі csv.

У програмному середовищі розробки IDLE (Python 3.8 64-bit) треба відкрити програму ts_pipe_9.py (яка за замовчуванням знаходиться в папці: C:\Users).

Після запуску програми ts_pipe_9.py на екрані з'являється графіки та метрики якості прогнозування.

Після ознайомлення з програмою, роботу можна звершити. Для цього необхідно натиснути червоний хрестик в правому верхньому кутку програмного середовища розробки.

ДОДАТОК Г


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ НАУКИ І ТЕХНОЛОГІЙ
ABSTRACTS
OF THE XIX INTERNATIONAL CONFERENCE
«MODERN INFORMATION AND COMMUNICATION TECHNOLOGIES ON A TRANSPORT, IN INDUSTRY AND EDUCATION»
18-19, December, 2025
СУЧАСНІ ІНФОРМАЦІЙНІ ТА КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ НА ТРАНСПОРТІ, В ПРОМИСЛОВОСТІ І ОСВІТІ
ПРИСВЯЧЕНО ПАМ'ЯТІ ПРОФЕСОРА ІГОРЯ ЖУКОВИЦЬКОГО
ТЕЗИ
XIX МІЖНАРОДНОЇ НАУКОВО-ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ
18-19 ГРУДНЯ 2025
ДНІПРО
2025

Дослідження часової ефективності генетичних алгоритмів

Михайлова Т. О., Андрющенко В. О., Український державний університет науки та технологій, Україна

Генетичні алгоритми (ГА) займають важливе місце серед еволюційних методів оптимізації, особливо в задачах, де потрібно знаходити рішення для складних багатовимірних структур. Завдяки здатності працювати з невизначеністю, численними локальними мінімумами, ГА широко застосовуються у штучному інтелекті для навчання моделей, оптимізації архітектур нейронних мереж та пошуку гіперпараметрів. У фармацевтиці та біоінформатиці їх використовують для оптимізації хімічних структур, прогнозування властивостей молекул і пошуку фармакологічних сполук. Така широта застосувань підсилює важливість дослідження часової ефективності ГА, оскільки зі зростанням складності задач збільшуються й вимоги до швидкості обчислень.

Незважаючи на значний обсяг досліджень, проблема скорочення часу виконання ГА залишається актуальною. У практичних системах — від транспортних рішень до промислових інформаційних технологій і цифрових освітніх платформ — важливо не лише отримати якісне рішення, а й забезпечити його вчасне знаходження. Наявні інструменти на Python не завжди дозволяють зручно аналізувати часову ефективність ГА, тому виникає потреба створити програму, що дасть змогу порівнювати різні конфігурації алгоритму та формувати висновки щодо впливу окремих параметрів на тривалість обчислень.

Метою роботи є дослідження залежності часової ефективності ГА від його ключових параметрів та створення програмного інструменту для багаторазових експериментів і порівняльного аналізу. Передбачається вивчення впливу розміру популяції, типу селекції, схрещування, імовірності мутації та критеріїв зупинки на час виконання алгоритму. Отримані результати дадуть змогу сформувати практичні рекомендації для сфер, де важливі швидкі та ефективні обчислення — від фармакології до адаптивних інтелектуальних систем.

Гіпотеза роботи полягає в тому, що оптимальне поєднання параметрів ГА може значно скоротити час розв’язання задачі без втрати якості результатів. Передбачається аналіз впливу базових характеристик — розміру популяції, кількості поколінь, типу відбору, виду задачі та обраних тестових функцій. Зростання кількості особин та глибини еволюції прямо впливатиме на час виконання, тоді як різні класи задач — гладкі, багатомодальні чи високорозмірні — демонструватимуть різні часові характеристики роботи алгоритму. Очікується, що результати дозволять визначити параметри з найбільшим впливом на швидкодію ГА та сформувати рекомендації щодо їх оптимального налаштування.

Запланований програмний інструмент буде реалізований мовою Python з можливістю гнучкого налаштування експериментів. Він забезпечить автоматизований збір статистики, вимірювання часу виконання операцій, і збереження результатів. Такий інструмент стане основою для подальших прикладних досліджень та може бути інтегрований у наукові й виробничі процеси, де потрібна швидка й ефективна оптимізація.