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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система глобальної пошукової оптимізації параметрів технологічних процесів для аналізу ефективності модифікованих еволюційних алгоритмів»

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

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

_____ Владислав БРИЛЬ
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-62
Владислав БРИЛЬ

Керівник: Юрій ЗАДОНЦЕВ
к.т.н.

Рецензент: Вячеслав ТРЕЙТЯК
к.т.н.

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Брилю Владиславу Володимировичу

1. Тема кваліфікаційної роботи: «Система глобальної пошукової оптимізації параметрів технологічних процесів для аналізу ефективності модифікованих еволюційних алгоритмів»

керівник кваліфікаційної роботи Юрій ЗАДОНЦЕВ, к.т.н.,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024 р. № 320.

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

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, параметри алгоритмів, аналіз ефективності еволюційних алгоритмів, вимоги до точності результатів.

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

1. Дослідження систем глобальної пошукової оптимізації.
2. Аналіз ефективності модифікованих еволюційних алгоритмів.
3. Розробка системи глобальної пошукової оптимізації параметрів технологічних процесів для аналізу ефективності модифікованих еволюційних алгоритмів.

5. Перелік ілюстративного матеріалу: *презентація*

1. Мета, об’єкта та предмет дослідження.
2. Глобальна оптимізація.
3. Проблематика оптимізації.
4. Ретельний розбір ЕА, котрий виступає фундаментом розробки проекту.
5. Технології розробки на основі генетичного алгоритму.
6. Аналіз цільової ефективності модифікованого алгоритму FSS.
7. Проведення аналізу результативності виконання генетичного алгоритму.
8. Зіставлення параметрів результативності роботи алгоритмів FSS і GA.

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	07.09-12.09.2024	
2	Вивчення матеріалів для аналізу ефективності модифікованих еволюційних алгоритмів	12.09.-19.09.2024	
3	Дослідження існуючих еволюційних алгоритмів	20.09.-27.09.2024	
4	Аналіз цільової ефективності модифікованого алгоритму FSS	01.10.-10.10.2024	
5	Порівняння параметрів результативності роботи алгоритмів FSS і GA	10.10.-24.10.2024	
6	Проведення аналізу результативності виконання генетичного алгоритму	24.10.-03.11.2024	
7	Оформлення роботи: вступ, висновки, реферат	03.11.-24.11.2024	
8	Розробка демонстраційних матеріалів	25.11-27.11.2024	
9	Попередній захист роботи	28.11.2024	

Здобувач вищої освіти _____ Владислав БРИЛЬ
(підпис)

Керівник
кваліфікаційної роботи _____ Юрій ЗАДОНЦЕВ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 100 стор., 10 табл., 35 рис., 32 джерел.

Мета роботи – детальний аналіз та оптимізація алгоритмів для ефективного розв’язання різноманітних задач. Основні завдання включають: визначення часової та просторової складності алгоритмів, оцінку їх масштабованості та можливостей паралельного виконання, аналіз точних значень часу виконання та обсягу використання пам’яті на конкретних наборах даних, а також розробку нових ефективних алгоритмів і вдосконалення існуючих для вирішення складних NP-повних задач.

Об’єкт дослідження – ефективність модифікованих еволюційних алгоритмів.

Предмет дослідження – дослідження поведінки алгоритмів, їх складності та ефективності. Поведінка алгоритмів у різних середовищах та у розрізі задач. Основна мета – аналіз та оптимізація алгоритмів, які ефективно розв'язують задачі з найменшими витратами часу і ресурсів дослідження. На основі аналізу обрано два найефективніших алгоритми, котрі були обрані задля вирішення завдання глобальної пошукової оптимізації.

КЛЮЧОВІ СЛОВА: АЛГОРИТМ FIREFLY, ГЛОБАЛЬНА ПОШУКОВА ОПТИМІЗАЦІЯ, АЛГОРИТМ FSS, ГЕНЕТИЧНИЙ АЛГОРИТМ.

ABSTRACT

Text part of the master's qualification work: 100 pages, 35 pictures, 10 table, 32 sources.

The purpose of the work detailed analysis and optimization of algorithms for effective solving of various problems. The main tasks include: determination of the time and space complexity of algorithms, assessment of their scalability and parallel execution capabilities, analysis of exact values of execution time and amount of memory usage on specific data sets, as well as development of new efficient algorithms and improvement of existing ones for solving complex NP-complete tasks.

Object of research – efficiency of modified evolutionary algorithms.

Subject of research – study of the behavior of algorithms, their complexity and efficiency. The behavior of algorithms in different environments and in terms of tasks. The main goal is the analysis and optimization of algorithms that effectively solve problems with the least expenditure of research time and resources.

Summary of the work: based on the analysis, the two most effective algorithms were chosen to solve the problem of global search optimization.

KEYWORDS: FIREFLY ALGORITHM, GLOBAL SEARCH OPTIMIZATION, FSS ALGORITHM, GENETIC ALGORITHM.

ЗМІСТ

ВСТУП 10

1 РОГЛЯД ЕВОЛЮЦІЙНИХ АЛГОРИТМІВ ЗАДЛЯ ПОШУКУ РОЗВ'ЯЗАННЯ ЗАДАЧІ ОПТИМІЗАЦІЇ 13

1.1 Глобальна оптимізація 13

1.1.1 Головні терміни та поняття 13

1.1.2 Проблематика оптимізації 15

1.2 Еволюційні алгоритми 18

1.2.1 Генетичний алгоритм 18

1.2.2 Алгоритм Firefly (світлячків) 20

1.2.3 Алгоритм бур'янистої оптимізації (Weed Optimization Algorithm) 22

1.2.4 Пошук зозулі (CS) 23

1.2.5 Bat-inspired algorithm (алгоритм кажанів) 25

1.2.6 Monkey Search 27

1.2.7 Алгоритм FSS 29

1.3 Вибір еволюційних алгоритмів для їх аналізу та модифікації 30

ВИСНОВОК ДО РОЗДІЛУ 1 31

2 ТЕХНОЛОГІЇ РОЗРОБКИ ПРОЕКТУ 32

2.1 Ретельний розбір ЕА, котрий виступає фундаментом розробки проекту 32

2.1.1 Технології розробки на основі генетичного алгоритму 32

2.1.2 Модернізований алгоритм FSS 36

2.2 Основні інструменти розробки системи глобальної пошукової оптимізації аналізу ефективності ЕА 41

2.2.1 C# 41

2.2.2 .NET Framework 4.7.2 46

2.2.3 WF 48

2.2.4 ZedGraph 49

2.2.5 MS Server 50

2.2.7 MSTest 51

ВИСНОВОК ДО РОЗДІЛУ 2 53

3 РОЗРОБКА КОМПОНЕНТІВ І АНАЛІЗ СТВОРЕНОГО СЕРЕДОВИЩА 54

3.1 Основні платформні компоненти генетичного алгоритму 54
3.1.1 Class BaseSpecies 54
3.1.2 Class Population 57
3.1.3 Class BaseDoubleSpecies 61
3.1.4 Class Interval 63
3.1.5 Class Analytics 63
3.2 Програмні елементи алгоритму Fish School Search 64
3.2.1 Class FishPointUniversal 64
3.2.2 Class FishUniversal 66
3.3 Аналіз цільової ефективності модифікованого алгоритму FSS 71
3.4 Проведення аналізу результативності виконання генетичного алгоритму 73
3.5 Зіставлення параметрів результативності роботи алгоритмів FSS і GA . 76
3.6 Користувацький інтерфейс розробленої системи 78
3.7 Аналіз БД 82
3.9 Аналіз якості системи 84
3.8 Ідеї, щодо майбутнього розвитку системи та її покращення 85
ВИСНОВОК ДО РОЗДІЛУ 3 87
ВИСНОВКИ 88
ПЕРЕЛІК ПОСИЛАНЬ 90
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ 93

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

EA	Еволюційні алгоритми
MS	Мавп’ячий алгоритм (Monkey Search)
FSS	Алгоритм скупчення риб (Fish School Search)
GA	Генетичний алгоритм (Genetic Algorithm)
FF	Алгоритм Firefly (світлячків)
WOA	Алгоритм бур’янистої оптимізації (Weed Optimization Algorithm)
CS	Пошук зозулі (Cuckoo Search)
BI	Алгоритм кажанів (Bat-inspired algorithm)
TIOBE Index	Показник актуальності мов програмування, створений компанією TIOBE
Generic	Типи, які надають можливість ініціалізувати класи, де тип заданий як параметр

ВСТУП

Проблеми оптимізації поширені в різних областях, таких як інженерія, інформатика, економіка та дослідження операцій. Багато реальних проблем оптимізації включають складні цільові функції з декількома локальними оптимістами, що робить їх складними для вирішення за допомогою традиційних методів оптимізації. Глобальні методи оптимізації пошуку спрямовані на пошук глобального оптимального рішення, ефективно досліджуючи весь простір пошуку, не потрапляючи в пастку локальної оптимізації.

Еволюційні алгоритми (ЕА) виникли як потужний клас методів глобальної пошукової оптимізації, натхненний принципами природної еволюції. Ці алгоритми імітують процеси природного відбору, розмноження і мутації, що спостерігаються в біологічних системах. ЕА працюють на популяції кандидатських рішень, ітеративно застосовуючи такі оператори, як відбір, кросовер та мутація, щоб розвивати кращі рішення протягом наступних поколінь [1].

Сила еволюційних алгоритмів полягає в їх здатності ефективно досліджувати складні пошукові простори, адаптивно направляючи пошук до перспективних регіонів. На відміну від методів на основі градієнтів, які покладаються на похідну інформацію і можуть легко застрягти в локальній optima, ЕА використовують підхід на основі населення і стохастичні оператори, що дозволяє їм уникнути локальної оптимізації і в кінцевому підсумку сходитися до глобального оптимуму або майже оптимального рішення [1].

Еволюційні алгоритми були успішно застосовані до широкого кола глобальних задач оптимізації, включаючи неперервну та дискретну оптимізацію, багатоцільову оптимізацію, обмежену оптимізацію та задачі комбінаторної оптимізації. Їх універсальність і надійність зробили їх привабливим вибором для вирішення складних задач оптимізації реального світу в різних областях, таких як інженерний дизайн, планування, маршрутизація та машинне навчання. Ця теза спрямована на всебічне вивчення еволюційних алгоритмів для глобальної пошукової оптимізації. Вона починається з введення фундаментальних концепцій

і принципів еволюційних обчислень з подальшим детальним дослідженням різних варіантів еволюційних алгоритмів і їх застосування в глобальній оптимізації [2]. Дисертація також буде досліджувати передові теми, такі як гібридні еволюційні алгоритми, паралельні реалізації та теоретичні аспекти еволюційних алгоритмів для глобальної оптимізації [2].

Завдяки широкому огляду літератури, емпіричним дослідженням та аналітичним дослідженням ця теза прагне сприяти розумінню та просуванню еволюційних алгоритмів глобальної пошукової оптимізації. Кінцевою метою є надання розуміння та рекомендацій щодо ефективного застосування цих алгоритмів для вирішення складних реальних задач оптимізації.

Таким чином, актуальною стоїть задача вивчення сучасних підходів еволюційних алгоритмів оптимізації, апаратних та програмних засобів, які використовуються для розв’язання складних задач оптимізації в різних галузях, та розробки ефективних еволюційних алгоритмів для таких застосувань.

Об’єкт дослідження – процес оптимізації складних задач за допомогою еволюційних алгоритмів.

Предмет дослідження – методи та засоби еволюційної оптимізації для розв’язання багатовимірних і нелінійних задач.

Мета роботи – підвищення ефективності процесу оптимізації складних задач за допомогою розробки та впровадження вдосконалених еволюційних алгоритмів, зокрема генетичних алгоритмів.

Методи дослідження – методи еволюційних обчислень, генетичні алгоритми, методи теорії оптимізації, математичне моделювання, методи проектування та програмної реалізації ПЗ [2].

Практична значущість результатів полягає у використанні розроблених еволюційних алгоритмів для оптимізації складних задач у різних галузях, таких як інженерія, фінанси, логістика, та інші сфери, що потребують знаходження оптимальних рішень в умовах багатовимірності та нелінійності. Це дозволить значно підвищити ефективність розв’язання таких задач і зменшити витрати часу та ресурсів.

Задля досягнення цілі піднімалися такі задачі:

1. Аналіз сучасних методів та засобів еволюційної оптимізації, в тому числі генетичних алгоритмів.

2. Проектування архітектури еволюційних алгоритмів, обрання технологій та засобів реалізації компонентів алгоритмів.

3. Розробка математичної моделі задачі оптимізації для досліджуваних алгоритмів.

4. Розробка алгоритмів оптимізації на основі еволюційних підходів, зокрема генетичних алгоритмів, та їх адаптація до конкретних задач.

5. Проведення моделювання роботи розроблених алгоритмів та оцінка їх ефективності у розв’язанні складних задач оптимізації.

Структура роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків [2].

1 РОГЛЯД ЕВОЛЮЦІЙНИХ АЛГОРИТМІВ ЗАДЛЯ ПОШУКУ РОЗВ'ЯЗАННЯ ЗАДАЧІ ОПТИМІЗАЦІЇ

1.1 Глобальна оптимізація

1.1.1 Головні терміни та поняття

Еволюційні алгоритми - це клас методів оптимізації, натхненний процесом природної еволюції. У контексті глобальної пошукової оптимізації (SEO) ці алгоритми можуть бути застосовані для пошуку оптимальних рішень для поліпшення рейтингу веб-сайту на сторінках результатів search system (SERP) [3]. Основна концепція використання еволюційних алгоритмів глобального SEO включає наступні ключові елементи:

- Представлення: Проблема представлена у вигляді набору параметрів або змінних, які можуть бути оптимізовані, таких як щільність ключових слів, структура вмісту, профіль зворотного посилання та інші фактори на сторінці та поза сторінкою, які впливають на рейтинг пошукової системи.
- Популяція: створюється популяція кандидатських рішень, відомих як індивіди або хромосоми. Кожна людина представляє можливу конфігурацію SEO-параметрів сайту.
- Функція фітнесу: Функція фітнесу визначається для оцінки якості або «придатності» кожного окремого рішення. У контексті SEO функція фітнесу може базуватися на різних метриках, таких як рейтинги пошукових систем, органічний трафік, показники конверсії або комбінація цих факторів.
- Вибір: Люди з більш високими балами фітнесу, швидше за все, будуть обрані для відтворення або переноситися до наступного покоління.
- Розмноження: Окремі люди проходять генетичні операції, такі як кросовер і мутація, щоб створити нові рішення для потомства. Кросовер поєднує в собі характеристики двох батьківських рішень, в той час як мутація вносить випадкові зміни для вивчення нових областей простору пошуку.

- Заміна: нові рішення для потомства замінюють деяких або всіх менш пристосованих осіб у популяції, створюючи нове покоління.
- Припинення: Еволюційний процес триває до тих пір, поки не буде виконана певна умова припинення, наприклад, досягнення задовільного рівня придатності, максимальної кількості поколінь або критерію збіжності.

У аналізі даних глобальна оптимізація відноситься до процесу пошуку найкращого можливого рішення або набору рішень, які оптимізують задану цільову функцію або функцію витрат по всьому пошуковому простору або можливому регіону [3].

Глобальна оптимізація особливо важлива при роботі зі складними, неопуклими, мультимодальними або розривними цільовими функціями, де може існувати кілька локальних оптимумів. У таких випадках традиційні методи оптимізації, які покладаються на локальний пошук або градієнтні методи, можуть потрапити в пастку локальної оптимізації, не знайшовши справжнього глобального оптимуму.

Мета глобальної оптимізації полягає в тому, щоб ретельно вивчити весь пошуковий простір, не застрягаючи в локальній оптимізації, і в кінцевому підсумку визначити глобальне оптимальне рішення або набір майже оптимальних рішень [3].

Оптимізація є процесом знаходження точки, яка мінімізує функцію. У контексті задач оптимізації терміни «глобальний локальний мінімум» і «глобальний мінімум» відносяться до різних понять:

Глобальний локальний мінімум - це точка в просторі пошуку, де цільова функція досягає локального мінімального значення, але це не обов'язково найменше значення по всьому простору пошуку. Іншими словами, глобальний локальний мінімум є найнижчою точкою в певному районі або області простору пошуку, але можуть існувати й інші регіони з ще нижчими значеннями функцій.

Глобальний мінімум, з іншого боку, є найменшим можливим значенням цільової функції над усім пошуковим простором. Він являє собою найкраще рішення, яке оптимізує цільову функцію в усьому світі, не обмежуючись будь-яким місцевим регіоном або районом.

Система у області глобальної оптимізації розрізняє наступні класи:

- детерміновані;
- евристичні.
- стохастичні;

Глобальна оптимізація знаходить своє “покликання” у моделюванні хімічних процесів, фінансовому прогнозуванні, Big Data аналізі, створенні лазерного обладнання, оцінці ефективності роботи підприємства, в обчислювальній фізиці та у вирішенні систем нелінійних рівнянь.

У задачах оптимізації кінцевою метою часто є пошук глобального мінімуму, оскільки він являє собою справжнє оптимальне рішення. Однак через складність пошукового простору або наявність декількох локальних мінімумів може бути складно знайти глобальний мінімум за допомогою методів локального пошуку або методів на основі градієнта [3].

Глобальні алгоритми оптимізації, такі як генетичні алгоритми, імітація відпалу або методи, пов'язані з гілками, призначені для більш ретельного вивчення простору пошуку та уникнення потрапляння в глобальні локальні мінімуми. Ці алгоритми використовують стратегії, щоб уникнути локальної оптимізації та продовжувати пошук, поки не буде знайдено глобального мінімуму або задовільного наближення.

Варто зазначити, що в деяких випадках знаходження точного глобального мінімуму може бути обчислювально нездійсненним або непотрібним, а врегулювання високоякісного глобального локального мінімуму може бути практичним компромісом, особливо у великомасштабних або реальних задачах оптимізації [3].

1.1.2 Проблематика оптимізації

Основна задача пошуку max функції $f(x)$ для допустимої множини $X \subseteq R^n$ зводиться до пошуку точки $x_* \in X$, що $f(x_*) \leq f(x) (f(x_*) \geq f(x))$ для всіх $x \in X$. Моделюємо ситуацію, де стоїть задача мінімізації функції f. Лімітування, котрі базуються на обчислювальній похибці, або нестачею технічних ресурсів, як

правило, забороняють віднайти математично точне вирішення даної планової задачі. В даній ситуації потрібно прямувати до пошуку приблизного результату, іншими словами, точки з незліченної кількості ε-оптимальних рішень $X_*^{\varepsilon} = \{ x \in X: f(x) \leq f(x_*) + \varepsilon \}$. Пошук математично точного результату потрібно вважати як окрему подію пошуку наближеного рішення $\varepsilon = 0$ [4].

Засіб задання множини X надає можливість виокремити досить жорстку класифікацію. Якщо $X = R^n$, то завдання можна класифікувати до класу задач безсумнівної оптимізації. Коли прийнятна множина задана завдяки обмеженням – це умовна оптимізація. Водночас, якщо X задача класифікується до сфери дискретної оптимізації.

Зазвичай виділяють 2 основні класи методів вирішення задач кінцевомірної оптимізації – орієнтовані та математично точні. Математично точні методи надають можливість забезпечувати оптимальність встановленого вирішення. До цієї класифікації співставляють різноманітні варіації методу меж та гілок. Задля математично точних методів властива висока трудомісткість, котра досить часто забороняє застосовувати їх задля подолання реальних задач. Евристичні методи побудовані на гіпотезах про властивості оптимального вирішення. На противагу математично точним рішенням, евристичні методи не забезпечують оптимальність віднайденого вирішення. Проте, в обставинах обмеженості обчислювальних можливостей евристики, як правило, є єдиним варіантом пошуком вирішення задачі. Гібридні та поширені методи, за якими, евристичні методи використовуються задля пошуку вирішення задачі, точні - задля аргументу оптимальності. Результативність гібридних методів спричинена тим, що евристичні алгоритми досить часто мають високу швидкість збіжності до оптимуму у порівняно з точними методами [4].

Найпоширеніший план організації точних методів - метод гілок та меж (МГМ), побудований на розбитті допустимої множини. Для комбінаторної, дискретної та загальної математичної оптимізації використовуються галузеві та зв'язані алгоритми, щоб знайти найкраще рішення. Величезна різноманітність задач оптимізації, таких як планування виробництва та планування екіпажу,

класифікуються як NP-Hard, оскільки вони не можуть бути вирішені за поліноміальний час. Алгоритм, пов'язаний з гілкою (B&B) є часто використовуваним методом для отримання точних рішень задач оптимізації NP-hard. Метод, який був вперше розроблений Лендом і Дойгом, часто називають алгоритмом; однак, більш точно сказати, що B&B містить сімейство алгоритмів, які всі мають спільну процедуру первинного рішення [5].

Схема роботи методу: Під час всього періоду роботи підтримується список $\{X_i\}$ підмножин допустимої множини. Він містить один елемент - допустимої множини X. Наступним кроком обирається один з елементів - підмножина X_i. Якщо X_i необхідне характеристикам відсіву, даний об'єкт видаляється зі списку. В іншій ситуації алгоритм підлягає дробленню на менші дрібні підмножини за правилом декомпозиції. Результати підмножини заміщають в переліку обраний елемент. Алгоритм доводить цикл роботи до кінця, коли в послідовності $\{X_i\}$ відсутні елементи [6].

Правилом відсіву – це будь-яка функція $\xi: S \to \{0,1\}$, котра визначена на деякій множині S підмножин R^n і відповідає кожній підмножині з S число 0 або 1. Якщо $V \in S$ виконується $\xi(V) = 1$, тоді можна стверджувати, що підмножина V відповідає правилам відсіву ξ. У такому разі можна стверджувати, що V не відповідає правилам відсіву ξ.

Нехай задані параметри відсіву $Y = \{\xi_1, .., \xi_m\}$. Кінцевий список підмножин $C = \{X_1, .., X_t\}$ для будь-якого $j = \{1, \dots, t\}$ знайдеться хоча б одне $i = \{1, \dots, m\}$ таке, що $\xi_i(X_j) = 1$, називаємо покриттям безлічі X задля набору правил відсіву Y.

Приведу доказ формулювання часто використовуваних правил відсіву. Характеристика - це призначення $g: S \to R$, встановлене множиною S підмножиною R^n, а для $V \in S$ застосовується $g(V) \leq f(x)$. Рекорд – це найменше значення головної функції, отримане під час активного розв’язання системи. Коли $x_1, .., x_k$ - логіка точок, в котрих шукається корінь головної функції, $f_k = f(x_r) = f(x_i)$ - рекорд; x_r - рекордне рішення [7]. Правило відсіву ξ_{rec} представляється наступним чином:

$$\xi_{rec} = \{1,\ g(V) \geq f_k - \varepsilon\ 0,\ g(V) < f_k - \varepsilon \tag{1.1}$$

Правила відсіву мають визначатися з урахуванням наявності покриття і у підсумку повинно виходити так, що рекорд, який ми шукаємо - ε-оптимальне рішення [7].

Рекорд черговості точок $x_1, .., x_k$, базується на базі розбиваючих множин. Як приклад, коли множини $\{X_i\}$ є паралелепіпедами - у ролі компонентів послідовності $x_1, .., x_k$ беруть їх центри. Є варіант, при якому послідовність точок $x_1, .., x_k$ формується самостійним від побудови покриття методом за допомогою варіативних евристичних способів. У ролі евристик використовують різноманітні локальні алгоритми (Ньютона, метод градієнтного спуску), крім того є складніші алгоритми, як приклад, стохастичні та генетичні методи [8].

Найважливіший компонент методу - правило декомпозиції $\Delta: S \to 2^S$, воно є функцією, обраною на множині S підмножин R^n та порівнює підмножини $V \in S$ кінцевий список його підмножин $V_1, .., V_m$, при цьому, $V = \cup_{i=1}^{m} V_i$, $V_i \cap V_j = \emptyset$ для всіх $1 \leq i < j \leq m$ [5, 7].

1.2 Еволюційні алгоритми

Тепер проаналізуємо алгоритми, які застосовують задля глобальної оптимізації: алгоритм косяка риб, алгоритм мавпячого пошуку, генетичний алгоритм, алгоритм бур'янистої оптимізації, алгоритм світлячків, алгоритм, інспірований кажанами, зозулин пошук.

1.2.1 Генетичний алгоритм

Генетичний алгоритм - це потужна методика оптимізації, натхненна процесом природної еволюції. Він належить до класу алгоритмів, званих еволюційними алгоритмами, які можуть знайти оптимальні або майже оптимальні рішення складних задач, імітуючи принципи генетики та природного відбору. Генетичний алгоритм починається з популяції випадково згенерованих рішень,

званих хромосомами або окремими особами, закодованих у відповідній формі. Кожне рішення оцінюється за допомогою функції фітнесу, яка визначає, наскільки добре вона вирішує проблему. Потім рішення монтажника відбираються для розмноження за допомогою генетичних операторів, таких як кросовер і мутація.

Кросовер: Два батьківські рішення поєднуються для отримання нових рішень для потомства, успадковуючи риси обох батьків. Це аналогічно статевому розмноженню в природі, що поєднує генетичний матеріал від двох батьків.

Мутація: випадкові зміни вводяться в рішення потомства для підтримки різноманітності та вивчення нових регіонів простору пошуку, як і те, як мутації відбуваються природно.

Вибір: Нові рішення для нащадків разом з деякими рішеннями для кріплення від попереднього покоління формують популяцію наступного покоління. Процес відбору покликаний сприяти рішенням монтажників, імітуючи принцип «виживання найбільш пристосованих» у природі. Цей ітераційний процес триває до тих пір, поки не буде виконано критерій зупинки, наприклад, досягнення заздалегідь визначеної кількості поколінь, пошук рішення, яке відповідає бажаному рівню придатності, або досягнення конвергенції (мало або немає поліпшення придатності протягом декількох поколінь). Припустимо, що ми хочемо використовувати генетичний алгоритм, щоб знайти оптимальне рішення для класичної задачі комівояжера (TSP), де комівояжер намагається віднайти найоптимальніший шлях, який відвідує безліч міст рівно один раз і повертається в початкове місто [9].

Представлення: Кожне рішення (хромосома) представляється у вигляді послідовності індексів міста, що кодують порядок відвідування міст.

Початкова популяція: Алгоритм починається з популяції випадкових послідовностей міст (наприклад, [5, 2, 1, 4, 3] для задачі з 5-ма містами).

Фітнес-функція: Фітнес кожного рішення оцінюється шляхом розрахунку загальної відстані відповідного маршруту. Короткі маршрути мають більш високу придатність.

Вибір: Рішення з коротшими маршрутами частіше вибираються для відтворення.

Кросовер: Два батьківські рішення поєднуються шляхом заміни підпослідовностей міських індексів, створюючи нові рішення для нащадків. Наприклад, схрещування [5, 2, 1, 4, 3] і [3, 4, 1, 2, 5] може виробляти [5, 2, 1, 2, 5] і [3, 4, 1, 4, 3].

Мутація: випадковий індекс міста в рішенні потомства обмінюється з іншим індексом, щоб ввести різноманітність. Наприклад, [5, 2, 1, 2, 5] може мутувати до [5, 1, 2, 2, 5].

Заміна: Нові рішення для потомства, а також деякі з найбільш пристосованих рішень попереднього покоління, формують популяцію наступного покоління. Цей процес триває до тих пір, поки не буде знайдено досить короткий маршрут або не буде виконано критерій зупинки. Здатність генетичного алгоритму досліджувати і експлуатувати простір пошуку через генетичних операторів часто призводить до високоякісних рішень для складних задач оптимізації, таких як TSP [10].

1.2.2 Алгоритм Firefly (світлячків)

Алгоритм Firefly - це техніка оптимізації ройового інтелекту, яка імітує поведінку світлячків та їх здатність спілкуватися та притягувати один одного за допомогою миготливих моделей x [11]. Він призначений для ефективного вирішення нелінійних, мультимодальних задач оптимізації. Алгоритм Firefly заснований на наступних трьох ідеалізованих правилах:

1. Всі світлячки унісекс, і їх приваблює будь-хто.
2. Привабливість світлячка пропорційна його яскравості, і для будь-яких двох світлячків менш яскравий буде притягуватися більш яскравим. Однак при збільшенні відстані яскравість зменшується.
3. Яскравість сяйва світлячка аналізується змінною цільової функції даної задачі оптимізації.

Ініціалізувати популяцію світлячків (потенційних рішень) випадковим чином:

1. Визначаємо яскравість або інтенсивність світла кожного світлячка на основі цільової функції.

2. Визначаємо коефіцієнт поглинання світла і коефіцієнт привабливості.

3. Ітеруємо через населення і переміщуємо кожного світлячка до найяскравішого світлячка на основі їх привабливості і відстані.

4. Формуємо рейтинг світлячків на основі їх яскравості і знаходимо поточне найкраще рішення.

Рух світлячка до світлішого визначається наступним рівнянням:

$$X_i = X_i + \beta_0 * exp\ exp\left(-\gamma * r_{ij}^2\right) * \left(x_j - x_i\right) + \alpha * (rand - 0.5) \quad (1.2)$$

$$x\_i = x\_i + \beta\_0 * exp(-\gamma * r\_ij\text{^}2) * (x\_j - x\_i) + \alpha * (rand - 0.5),$$

де X_i і X_j - позиції i-го і j-го світлячків відповідно

β_0 - привабливість при r = 0

γ - коефіцієнт поглинання світла

r_{ij} - це відстань між світлячками i та j

α - параметр рандомізації

rand - випадкове число, отримане з рівномірного розподілу в [0, 1]

Розглянемо проблему знаходження глобального мінімуму функції:

$$f(x) = -\sum \quad (sin\ sin\ (x_i) * Sin^{20}((i * x_i^2)/\pi)), \quad (1.3)$$

де x - вектор n змінних рішень, і кожна X_i знаходиться в діапазоні [0, π].

Алгоритм Firefly може бути застосований до цієї проблеми шляхом ініціалізації популяції світлячків (потенційних рішень) та ітеративного переміщення їх до яскравіших світлячків (кращих рішень) на основі їх привабливості та відстані. Яскравість або інтенсивність світла кожного світлячка визначається значенням функції для цього рішення. Після достатньої кількості ітерацій алгоритм повинен зійтися до глобального мінімуму функції, що становить приблизно -1,801 для n = 2 вимірів [11].

1.2.3 Алгоритм бур’янистої оптимізації (Weed Optimization Algorithm)

Алгоритм оптимізації бур'янів - відносно новий метаевристичний алгоритм оптимізації, запропонований Мехді Зіварі та Аміром Х. Гандомі в 2016.

Алгоритм оптимізації бур'янів - це методика оптимізації на основі популяції, яка імітує спосіб розмноження та поширення бур'янів у природному середовищі. Він призначений для вирішення складних задач оптимізації шляхом ефективного дослідження пошукового простору та використання перспективних рішень.

Алгоритм BOA заснований на наступних ідеалізованих правилах:

- Бур'яни хаотично поширюються по всьому простору пошуку.
- Насіння випадковим чином розподіляються відповідно до нормального розподілу з середнім значенням нуля і змінною дисперсією.
- На ріст і розмноження бур'янів впливають придатність колонії, кількість виробленого насіння і випадковий розподіл насіння.
- Алгоритм ітеративно оновлює популяцію бур'янів і замінює їх відтвореним насінням, якщо насіння має кращі значення придатності.

Алгоритм працює наступним чином:

- Ініціалізується популяція бур'янів (потенційних рішень) випадковим чином в просторі пошуку.
- Проводиться оцінка придатності кожного бур'яну на основі цільової функції.
- Визначається максимальні та мінімальні значення придатності населення.
- Виробляється насіння для кожного бур'янів на основі його значення придатності і максимальних, і мінімальних значень придатності.
- Розподіляється насіння випадковим чином навколо батьківських бур'янів відповідно до нормального розподілу з середнім значенням нуля та змінною дисперсією.
- Замінюються бур'яни їх відтвореним насінням, якщо насіння мають кращі значення придатності.

$$S_{max} = floor\left(\frac{fit_{max} - fit_i}{fit_{max} - fit_{min}}\right) * (max_s - min_s - min_s)) + min_s, \qquad (1.4)$$

де S_{max} - максимальна кількість насіння, яке може дати бур'ян

f_{max} та f_{min} - максимальні та мінімальні значення придатності населення

fit_i - значення поточних бур'янів

min_s та max_s - це максимальна та мінімальна кіл-сть насіння бур'яну, яке він може дати

Проаналізуємо проблематику пошуку глобального мінімуму функції сфери:

$$f(x) = \sum_i^n X\,(x_i^2), \qquad (1.5)$$

де x - вектор n змінних рішень.

Якщо насіння мають кращі значення придатності, ніж їх батьківські бур'яни, вони замінять батьківські бур'яни в популяції. Цей процес триває, поки алгоритм не збіжиться до глобального мінімуму функції сфери, $f(x) = 0$, коли всі $x_i = 0$.

Алгоритм WOA був успішно застосований до різних задач оптимізації, включаючи задачі інженерного проектування, задачі оцінки параметрів та інші складні завдання оптимізації, завдяки простоті, гнучкості та здатності ефективно досліджувати та експлуатувати простір пошуку [12].

1.2.4 Пошук зозулі (CS)

Алгоритм пошуку зозулі (CS) є натхненним природою метаевристичним алгоритмом оптимізації, розробленим Синь-Ше Ян і Суаш Деб в 2009 році. Він натхненний примусовою паразитичною поведінкою деяких видів зозулі, в поєднанні з поведінкою польоту Леві деяких птахів і фруктових мух. Алгоритм пошуку зозулі - це популяційна методика оптимізації, яка імітує селекційну поведінку зозульок і моделі польоту певних птахів і комах. Він призначений для вирішення різних типів задач оптимізації, включаючи неперервні, дискретні та комбінаторні задачі [13].

Алгоритм пошуку зозулі заснований на наступних ідеалізованих правилах:

- Кожна зозуля відкладає по одному яйцю за раз і відкладає його в випадково вибране гніздо.
- Кращі гнізда з високоякісними яйцями (розчинами) будуть перенесені на наступне покоління.
- Кількість доступних гнізд господарів фіксована, і господар може виявити інопланетне яйце з ймовірністю $\rho \in [0, 1]$. При цьому птах-господар може або відкинути яйце, або відмовитися від гнізда і побудувати нове на новому місці.

Алгоритм працює наступним чином:

- Одна зозуля дає одне яйце за один раз у рандомне гніздо;
- найкращі гнізда з яйцями високої якості перекладають у наступне покоління;
- яйце зозулі, яке відклали у гніздо, може бути виявлено з наступною ймовірністю $\xi_a \in (0; 1)$та видалено з гнізда.

План роботи алгоритму:

1. Ініціалізуємо популяцію $S = s_i, i \in [1: |S|]$ з $|S|$ гнізд та зозулі, шукаємо вектори X_0^i та вектор початкового положення зозулі X_c.
2. Проводимо пошук нового положення зозулі

$X_c' = X_c + V \otimes L_{|X|}(\lambda)$,

$V = (v_j, j \in [1: |X|])$ – вектор обсягу кроків до конкретних компонентів вектора X;

$L_{|X|}(\lambda) - (|X| \times 1)$ – випадковий вектор самостійний дійсних випадкових чисел.

1. Рандомно обираємо гніздо s_j, $j \in [1: |S|]$ і, якщо $\varphi(X_c) > \varphi(X_j)$, заміщуємо яйце у гнізді на яйце зозулі, тобто беремо $X_j = X_c$.
2. З ймовірністю ξ_a знищуємо з популяції деякий список гірших рандомно обраних гнізд.

Стартові положення гнізд та зозулі вважаємо рандомними, рівномірно поділеними у гіперпаралелепіпеді. Як правило, всі компоненти вектора V ідентичні та рівними v, де величина v напряму має зв’язки з областю пошуку. Вирішальні

класифікатори алгоритму - константи, їх роль полягає у пошуку траєкторії польотів Леві і ймовірність віднімання гнізд ξ_a [14].

1.2.5 Bat-inspired algorithm (алгоритм кажанів)

Bat Algorithm (BA) - це метаевристичний алгоритм оптимізації, запропонований Сінь-Ше Ян у 2010 році. Він натхненний поведінкою ехолокації кажанів, які використовують звукові хвилі для виявлення здобичі та навігації в темряві. Алгоритм кажанів - це метод оптимізації на основі популяцій, який імітує поведінку кажанів при пошуку здобичі. Він призначений для вирішення різних типів задач оптимізації, включаючи неперервні, дискретні та комбінаторні задачі. Алгоритм кажанів поєднує в собі фази дослідження та експлуатації через зміну частот випромінювання імпульсів та гучності. Він також включає в себе місцеві методи пошуку для підвищення конвергенції до глобального оптимуму [15].

1. Завдяки ехолокації усі миші вимірюють відстань до здобичі та перешкод, ще у них є функція аналізування типу перешкоди.

2. Сталі положення та швидкість миші S_i дорівнюють X_i та V відповідно. Задля пошуку здобичі миші часто змінюють частоту сигналів.

3. Частота сигналів коливається в межах $[\omega^{min}, \omega^{max}]$, $\omega^{max} > \omega^{min} \geq 0$, а гучність сигналів - в межах від 0 до 1.

Якщо допустити, що йдеться про задачу глобальної безумовної мінімізації функції:

1. Задаємо стартові положення агентів S_i.

2. Шукаємо глобально найкращого агента та рішення X^{**}.

3. Наступним кроком, проводиться переміщення усього списку агентів на один крок згідно міграційної процедури.

4. З ймовірністю ξ_{ir} проводимо процедуру локального пошуку в межах найкращого рішення X_i^* знайденого агентом S_i за усі минулі цикли. Вважаємо знайдене рішення в ролі нового сталого становища агента S_i.

5. В околицях поточного рішення X, рандомно генеруємо рішення X_i'. Якщо $\varphi(X_i') < \varphi(X)$, то з ймовірністю ξ_{ia} вважаємл рішення X_i' в ролі нового поточного становища агента S_i. Віднаходимо нове глобально найкраще рішення.

6. Еволюція характеристик ξ_{ir}, ξ_{ia}.

7. Після цього потрібно виконати перевірку завершення ітерацій. Якщо дані отримано, вважаємо за вирішення дану точку X^{**} і завершуємо обчислення, при іншому розкладі повертаємося до пункту №2.

Ініціалізацію популяції проводиться підходом рандомного рівномірного ділення учасників S_i у області пошуку. Обираємо стартові значення зміних ω_i, гучностей a_i та частот дублювання імпульсів r_i, рівномірно рандомно розподіляючи у відповідних межах $[\omega^{min}, \omega^{max}]$, $[a^{min}, a^{max}]$, $[0;\ 1]$.

Міграція агента s_i вирішується за наступними формулами:

$$X_i' = X_i + V_i', \tag{1.6}$$

$$V_i' = V_i + \underline{w_i'}(X_i - X^{**}), \tag{1.7}$$

$$\omega_i = \omega^{min} + (\omega^{max} - \omega^{min}) \cup_1 (0; 1) \tag{1.8}$$

Згідно з міграційною процедурою агент транспортується у напрямку, котрий позначений результатом вектора та зміною положення на мигулій ітерації (доданок V_i) і збуреним вектором направлення на учасника $(X_i - X^{**})$. Локальний пошук здійснюємо за наступним планом. Рандомним методом видозмінюємо наявне положення S_i згідно формули.

$X_i' = X_i + \underline{a} \cup_{|x|} (-1,1)$, $i \in [1: |S|]$, де $\underline{a}$ – актуальна вибірка значення гучності усіх членів популяції:

$$\underline{a} = \frac{1}{|S|} \sum_{i=1}^{|S|} \quad a_i \tag{1.9}$$

Знаходимо значення функції у наступній точці. Засновники алгоритму провели обширне тестовий аналіз ефективності алгоритму з генетичним алгоритмом і Firefly. Було представлено позиції ймовірності локалізації глобального екстремуму алгоритму кажанів більш результативний, порівняно з іншими алгоритмами.

1.2.6 Monkey Search

Алгоритм Monkey Search - це метаевристичний алгоритм оптимізації, запропонований Джагдешем Чандом Бансалом та колегами в 2019 році. Його надихає альпіністська поведінка мавп, де вони піднімаються вгору по деревах або горах.

Алгоритм пошуку мавп - це метод оптимізації на основі популяції, який імітує альпіністську поведінку мавп та їх здатність досліджувати та використовувати простір пошуку. Він призначений для вирішення різних типів задач оптимізації, включаючи неперервні, дискретні та комбінаторні задачі [17]. Після зазначеного фіксованого числа підйомів і локальних стрибків мавпа розуміє, що у достатній мірі дослідила ландшафт у свого стартового положення. Для того, щоб проаналізувати нові інтервали простору пошуку, мавпа здійснює глобальний стрибок.

План виконання М-алгоритму зображають у наступній формі [18]:

Ініціалізацію популяції мавп $s_i, i \in [1:|S|]$ презентують за загальними правилами – за шляхом рівномірного рандомного розподілу агентів у пошуковому просторі.

Перебіг руху вгору представляє процес локального пошуку і є можливість його реалізації великою кількістю варіантів. В оригінальному варіанті автори алгоритму застосовують алгоритм на базі процедури стохастичної апроксимації [19].

План ходу локальних стрибків учасників $s_i, i \in [1:|S|]$ розуміє під собою наступні пункти до виконання:

1. Беручи до уваги поточний стан агенту X_i, створюємо актуальне положення X_i' за формулою:

$$x'_{i,j} = U_1\left(\left(x_{i,j} - b\right); \left(x_{i,j} + b\right)\right), i \in [1: |S|], j \in [1: |X|], \quad (1.10)$$

де нова позиція учасника розуміє під собою рандомну величину, пропорційну розподілену в інтервалі $[\left(x_{i,j} - b\right); \left(x_{i,j} + b\right)]$. $b > 0$ - гранично допустима довжина стрибка [19].

2. Коли точка X_i' - допустима, $\varphi\ (X_i') \geq \varphi(X)$, $X_i = X_i'$ і закінчуємо для конкретного учасника локальні стрибки, умови не виконуються – розпочинається алгоритм дій з 1-го пункту.

Глобальні стрибки учасників стартують з актуального стану у напрямку центру ваги їх координат:

1. Генеруємо рандомне дійсне число $v = U_1(v^-; v^+)$, яке містить дані кроку глобального стрибка. v^-, v^+ - нижня та верхня межі даної величини, вони мають у загальному випадку кардинально різні ознаки, величина v може бути як позитивною та негативною.

2. Шукаємо рандомне положення X_i' s_i':

$$x'_{i,j} = x_{i,j} + v\left(x_j^c - x_{i,j}\right), i \in [1: |S|], j \in [1: |X|], \quad (1.11)$$

де $x_j^c = \frac{1}{|S|}\sum_{i=1}^{|S|}$ $x_{i,j}$ – головна точка тяжіння агентів популяції по j-му координатному напрямку.

3. Коли положення X_i' - допустиме, можна вважати $X_i = X_i'$ та завершувати глобальні стрибки для обраного агента.

Коли обсяг кроку стрибка v припадає на позитивне значення, учасник переходить до стрибку у направленні центра ваги X^c, коли від'ємне значення - в протилежну сторону.

М-алгоритм не надає гарантії, що найкращий результат можна отримати на останньому етапі, у даному випадку потрібно зберігати в пам'яті стале найкраще рішення [20].

1.2.7 Алгоритм FSS

Алгоритм Fish Shoal Search (FSS), також відомий як алгоритм Fish Swarm Optimization (FSO), є метаевристичним алгоритмом оптимізації, запропонованим Хоссамом Аббассом у 2020 році. Він натхненний колективною поведінкою та соціальною взаємодією риб'ячих мілин, особливо їх харчуванням та поведінкою. Алгоритм Fish Shoal Search - це метод оптимізації базуючись на популяції, який імітує поведінку риб'ячих мілин у пошуку їжі або здобичі. Він призначений для вирішення різних типів задач оптимізації, включаючи неперервні, дискретні та комбінаторні задачі. Алгоритм FSS складається з двох найменувань:

- Оператор годування. Він формалізує результат оцінювання ландшафту акваріума;
- Оператори плавання. Оператори плавання вводять процесс міграції учасників.

Оператор годування. W_i – сталі дані учасника S_i. Сталі дані учасника рівнозначна нормалізованій різниці фітнес-функції на наступній і сталій ітераціях [21].

Існує три способи плавання:

- колективно-вольове;
- індивідуальне;
- інстинктивно-колективне.

Дані класи плавань здійснюються на відрізках $(t, \tau], (\tau, \theta], (\theta, t\,']$ головного ітераційного інтервалу $(t, t']$.

Індивідуальне плавання. Направлення переміщення агента у даному випадку рівноймовірно рандомний. Переміщення примушує вийти за межі інтервалу прийнятних значень зміних D – перебіг переміщень зупиняється. Відповідно, переміщення агента S_i не відбувається, коли у створеній точці $X_{i\tau}$ значення фітнес-

функції не більше її значення у першій точці X_{it}. План індивідуального плавання містить у собі деякі стале дані. Отже, індивідуальне плавання учасника можна представляти як локальний пошук у інтервалі актуального стану учасника.

Інстинктивно-колективне плавання зображається після завершення усіма учасниками індивідуальних плавань:

$$W_i' = W_i + \frac{\varphi(X_i') - \varphi(X_i)}{max(\varphi(X_i'), \varphi(X_i))},\ i \in [1: |S|] \quad (1.12)$$

Другий доданок у формулі - крок міграції, він представляє суму індивідуальних переміщень учасників.

Головна задача колективно-вольового плавання у переміні усіх учасників на шляху актуального центру жорсткості популяції, у випадку коли сумарна вага зграї збільшилась, а у протилежному напрямку - коли дана вага зменшилася.

Щільний алгоритм FFS працює на базі модифікації операторів алгоритму, створені оператори – це оператори пам'яті та розбиття. Memory operator фундаментується на базі векторів пам'яті M_i, агентів популяції та визначає перелік впливів конкретного агента на інших членів популяції. Partitioning operator працює з пам'яттю агентів та використовується задля створення на базі підпопуляцій [21].

1.3 Вибір еволюційних алгоритмів для їх аналізу та модифікації

Проаналізувавши усі «за» та «проти», дійшли до висновку, що кожен еволюційний алгоритм особливий та чудово підходить під свої конкретні задачі. На основі цього, було вирішено працювати надалі лише з двома алгоритми, які підходять найбільше для проблематики глобальної оптимізації. Наступним чином буде взято на більш детальний аналіз – FSS та GA алгоритми. GA та FSS алгоритми містять у собі великий перелік інструментів, за допомогою яких, буде вирішуватися завдання пошуку екстремуму.

ВИСНОВОК ДО РОЗДІЛУ 1

У цьому розділі було розглянуто концепції глобальної оптимізації та еволюційних алгоритмів, які є потужним інструментом для вирішення складних задач оптимізації. Було описано основні терміни та проблематику оптимізації, включаючи поняття глобального та локального мінімумів, а також різні класи методів оптимізації. Було проаналізовано декілька популярних еволюційних алгоритмів, таких як генетичний алгоритм, алгоритм світлячків (Firefly Algorithm), алгоритм бур'янистої оптимізації (Weed Optimization Algorithm), Cuckoo Search, Bat Algorithm, пошук мавпи (Monkey Search) та алгоритм Fish Shoal Search (FSS). Для кожного алгоритму було надано опис принципів роботи, основних правил та процедур, а також приклади їх застосування для вирішення задач оптимізації. Еволюційні алгоритми є універсальними та ефективними методами для пошуку глобальних оптимумів завдяки своїй здатності досліджувати складні пошукові простори та уникати потрапляння в локальні оптимуми. Вони імітують процеси природного відбору, розмноження та мутації, що дозволяє їм адаптивно направляти пошук до перспективних регіонів.

Отже, еволюційні алгоритми є привабливим вибором для вирішення широкого кола задач глобальної оптимізації в різних галузях, таких як інженерія, економіка, машинне навчання та багато інших. Їх універсальність, надійність та здатність знаходити високоякісні рішення роблять їх потужним інструментом для вирішення складних реальних задач оптимізації. Незважаючи на свою ефективність, еволюційні алгоритми також мають деякі обмеження та виклики. Одним з них є налаштування параметрів алгоритму, таких як розмір популяції, ймовірності кросоверу та мутації, критерії відбору та зупинки. Неправильний вибір цих параметрів може призвести до повільної збіжності або потрапляння в локальні оптимуми.

2 ТЕХНОЛОГІЇ РОЗРОБКИ ПРОЕКТУ

2.1 Ретельний розбір ЕА, котрий виступає фундаментом розробки проекту

2.1.1 Технології розробки на основі генетичного алгоритму

GA працює на моделюванні формації одиниць на відрізку великого списку наслідувань. Під час виконання генетичного алгоритму генеруються одиниці з більш результативними показниками, одиниці з гіршими показниками - видаляються. Основні терміни потрібні для роботи з GA:

- одиниця - зміст x з нескінченного переліку значень, враховуючи значення головної функції для точки x.
- хромосоми - зміст x.
- хромосома - зміст x та у випадку, якщо x - вектор.
- функція адаптації (fitness функція) - адаптивна функція $f(x)$.
- сполука - сукупність особин.
- формація - номер ітерації GA.

Одиниця - одне значення x серед нескінченної кількості усіх резолюцій. Зміст оптимізуючої функції (мінімум функції) визначається для кожного значення x. Беремо до уваги, що чим менше значення має головна функція, тим результативнішим являє собою рішення [22].

Демонстрація структурної схеми GA (генетичного алгоритму):

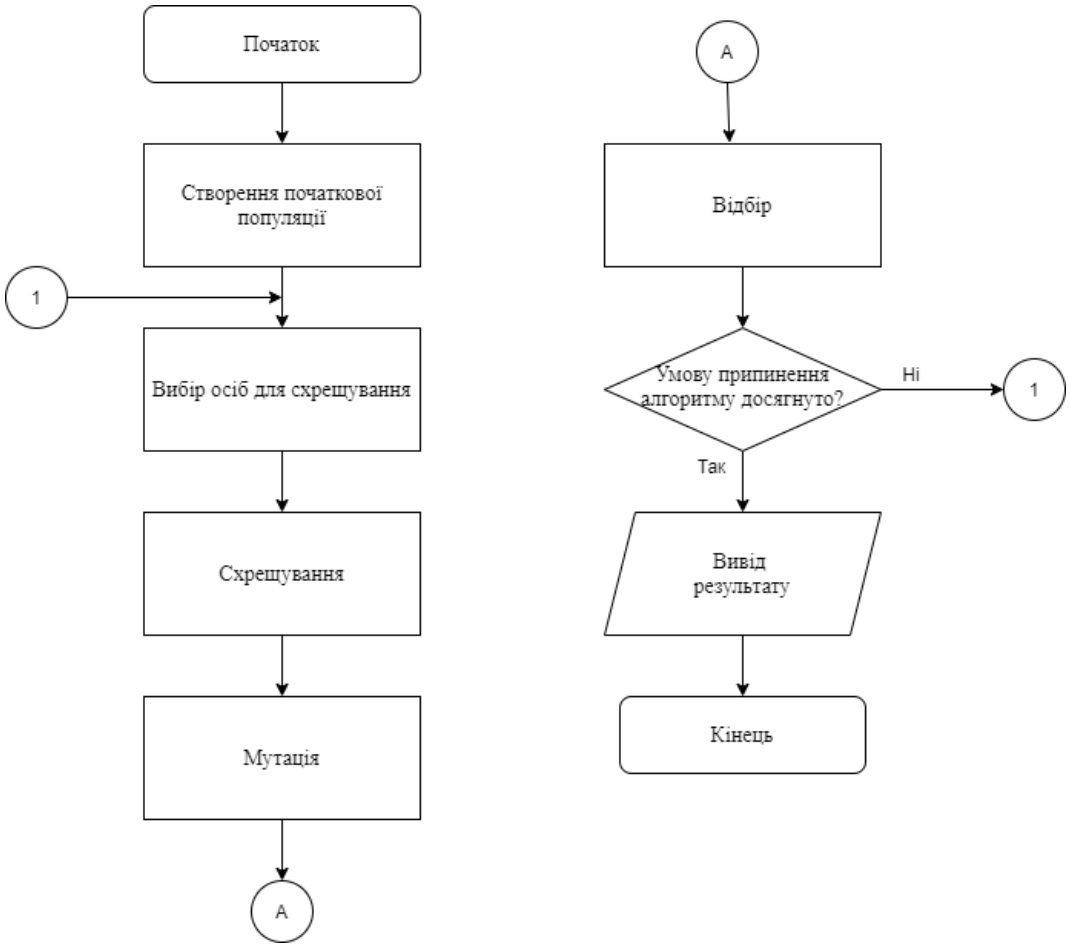


Рис. 2.1 GA

Огляд усіх етапів виконання алгоритму [22].

1) Зародження популяції

Перший етап - формування стартової формації, іншими словами, формування нескінченної кількості одиниць із різноманітними даними хромосом x. Частіше за все, стартову формацію формуються із одиниць з рандоними характеристиками хромосом, намагаючись, щоб характеристики хромосом у формації покривали усю область пошуку рішення рівномірно.

Якщо якомога більше одиниць буде сформовано на цій стадії, тим більша ймовірність того, що алгоритм досягне коректного результату, за умови розширення розміру стартової формації, зазвичай необхідно провести меншу кількість ітерацій GA. При збільшенні розміру формації необхідно більша кількість

операцій головної функції та інших генетичних розрахунків на наступних етапах. При збільшенні обсягів формації збільшується час операції покоління.

Розмір формації необов'язково є сталим на відрізку виконання алгоритму, як правило, коли зростає кількість номерів поколінь - формацію потрібно зменшувати у розмірах, згодом можна прийти до того, що все більший список одиниць розташовуватимуться ближче до рішення.

2) Вибірка одиниць задля симбіозу

Наступним чином потрібно обрати одиниці, котрі візьмуть роль в операції симбіозу. Найпростіша операція – доводити до симбіозу кожну одиницю з усіма рибами, але у такому випадку, на наступному кроці потрібно буде проводити велику кількість розрахунків симбіозу та операцій над даними головного функціоналу. Варто зауважити, що у даній роботі, знаходиться значення імовірності симбіозу одиниць, у симбіозі беруть участь особини, котрі знаходяться у списку імовірнісних сталих, інші одиниці втрачають здатність продовжувати потомство [23].

Отже, на цій ітерації необхідно вивести пари, на черговому кроці з ними пройде процедура симбіозу. Підсумком цієї дії - список партнерів задля симбіозу.

3) Симбіоз

Симбіоз - це генетична операція, результатом якої є зароджування нових одиниць з новими параметрами хромосом. Хромосоми формуються на базі хромосом батьків. Як правило, у процесі симбіозу одного списку даних формується одна дочірня особина, але теоретично дочірніх особин може бути більше. Алгоритм симбіозу можна втілити у життя різноманітними методами.

Мета симбіозу полягає у тому, що потрібно взяти одну частину хромосоми та другу іншого виду і сформувати на їх базі нову хромосому. У більшості випадків, хромосоми проводять процес схрещування - побітово. Головна ідея полягає у рандомно обраній точці симбіозу хромосоми, після цього, потрібно сформувати нову хромосому, котра містить у собі дані з лівої частини першої хромосоми та правої частини другої.

Існує дві хромосоми (8-бітові): 10110111 та 01110010. Рандомним чином визначаємо точку розриву (символ '|'):

101 | 10111

011 | 10010

З цього можна створити дві різні хромосоми - 101 10010 та 011 10111. Вибір хромосом - залежить від нас, рішення приймається за допомогою генератора рандомних одиниць. Як варіант, можна віднайти дві та більше маркерів перетину.

В даній ситуації відбувається симбіоз хромосоми типу Double, тут присутні два маркери перетину - у середині слова та знак числа, іншими словами, на початку проводиться процедура симбіозу хромосоми побітово, незважаючи на знак, а другою дією є те, що обирається рандомним чином знак від однієї з хромосоми [23].

4) Мутація

Мутація – надважлива та необхідна стадія генетичного алгоритму, він урізноманітнює хромосоми одиниць, що зменшує ймовірність того, що результат зійдеться до локального мінімуму замість глобального. Мутація представляє в своїй особі рандомну зміну в одиниці хромосоми, котра була сформована операцією симбіозу.

Зазвичай, ймовірність мутації є не дуже високою, це зроблено для того, щоб мутація не перешкоджувала збіжності алгоритму, в іншому випадку, мутація може погіршувати якість одиниць із здоровими хромосомами [22].

Аналогічно і для симбіозу, мутація має можливість експериментувати з алгоритмами. Як варіант, є можливість замістити одну хромосому, або декілька хромосом на рандомну величину. В даному проекті застосовується побітова мутація, тобто, в хромосомі інвертуються одиничний біт Рис 2.2.

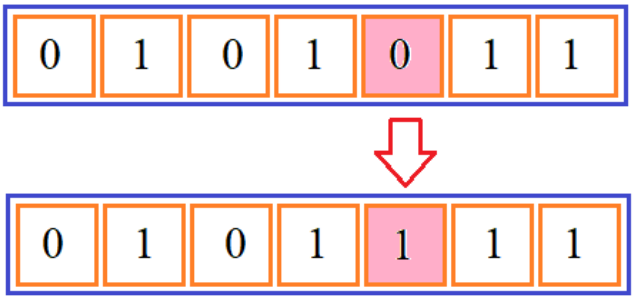


Рис. 2.2 Мутація одного біта

Унаслідок мутації одиниці мають можливість сформуватися як більш вдалі, так і менш, дана процедура надає можливість з ненульовою ймовірністю сформуватися здоровій хромосомі зі списком нулів та одиниць, котрі не були присутні у батьківських одиницях.

5) Селекція

Унаслідок симбіозу створюються одиниці. За умови відсутності селекції, список одиниць збільшується згідно експоненціальної аксіоми, а це, в свою чергу, потребує більшого об’єму ОЗУ та часу оброблення послідовно нових формацій популяції. Після етапу коли створюються нові одиниці необхідно звільняти формацію від менш вдалих.

Алгоритми селекції існують різні. Як правило, першим чином відсіюються одиниці, чиї хромосоми не досягають заданого інтервалу пошуку. В даній проектній роботі проводиться сортування видів за значенням головної функції, іншими словами, залишаються найбільш «живучі» види.

6) Вимоги фінішу алгоритму

Характерною ознакою завершення алгоритму – перехід до задання деяких ітерацій. За умови, що дана характеристика використовується належним чином, через те, що GA - імовірнісний, різні запуски алгоритму часто співпадають з різною швидкістю [25].

2.1.2 Модернізований алгоритм FSS

Алгоритм FSS - криптографічний метод, який використовується для розбиття функції на кілька спільних ресурсів таким чином, що оригінальна функція може бути реконструйована тільки якщо всі спільні ресурси об'єднані. Ця концепція часто використовується в безпечних багатопартійних обчислювальних протоколах та протоколах конфіденційності, де сторони можуть спільно обчислювати функцію над своїми входами, зберігаючи ці входи приватними.

Методи вибору ознак на основі статистичної значущості:

- Статистичний показник: ці алгоритми часто використовують статистичні тести (наприклад, тест хі-квадрат, F-тест, взаємна інформація), щоб виміряти зв’язок між кожною ознакою та цільовою змінною.
- Перевірка гіпотези: для кожної функції зазвичай формулюється нульова гіпотеза (наприклад, «ця ознака не має зв’язку з цільовою змінною») і перевіряється на альтернативну гіпотезу.
- Поріг значущості: Заздалегідь визначений рівень значущості (наприклад, p-value < 0,05) часто використовується для прийняття рішення щодо відхилення нульової гіпотези.
- Ранжування ознак: функції зазвичай ранжуються на основі їх статистичної значущості або сили їх зв’язку з цільовою змінною.
- Вибір: функції, які відповідають порогу значущості, вибираються для включення в модель.
- Ітераційний процес: деякі алгоритми можуть використовувати ітераційний підхід, переоцінюючи значущість ознак після кожного кроку вибору чи виключення.

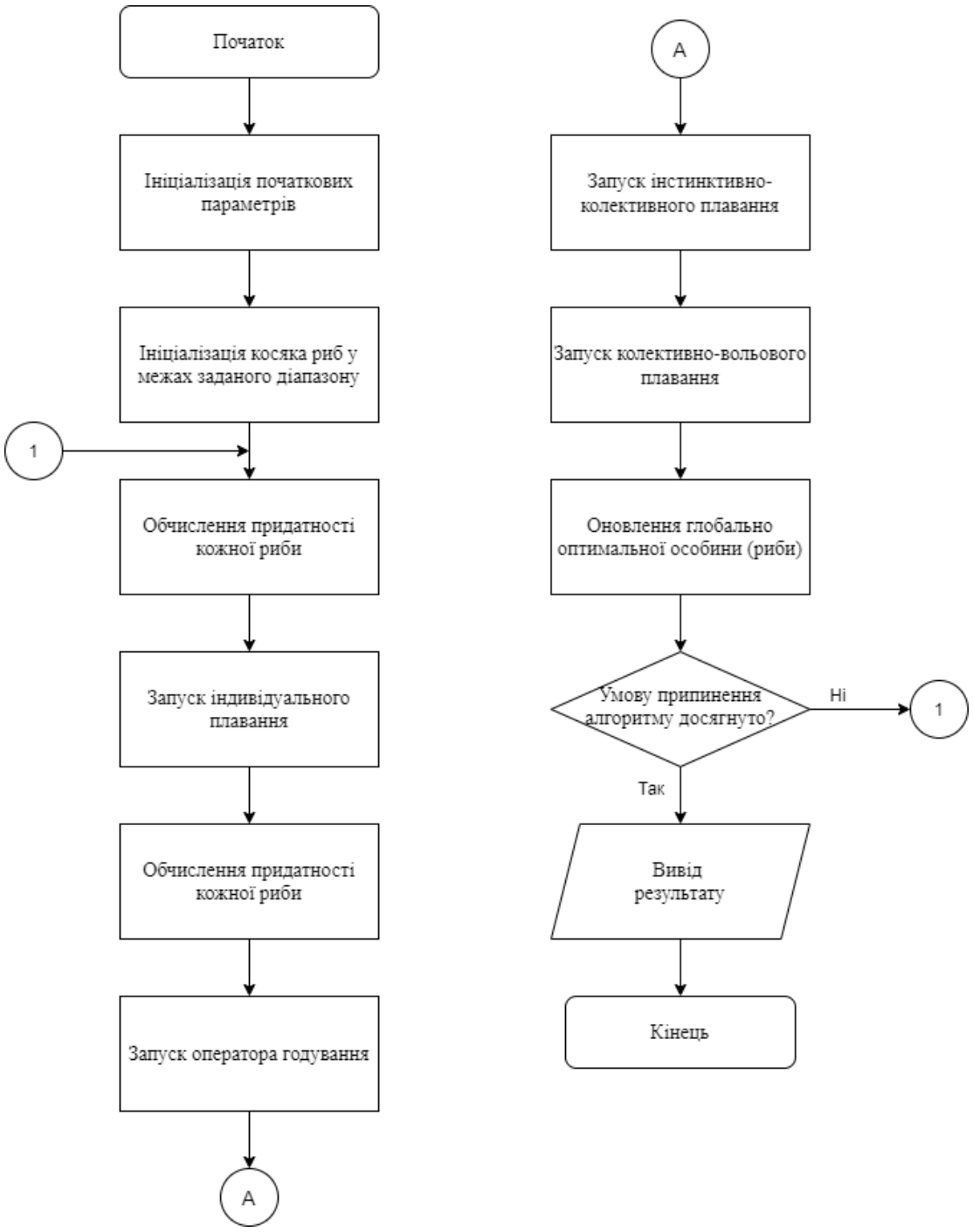


Рис. 2.3 Блок-схема алгоритму FSS

FSS - алгоритм пошуку на базі формації, діях риб, котрі розмножуються та зменшують популяцію у пошуках здобичі. Локація риби - вірогідне вирішення проблематики оптимізації. Алгоритм застосовує вагу для усіх операторів, які демонструють результати, наскільки успішним був пошук риби у зграї. FSS алгоритм поділяється на три типи [22]:

1) Самостійне плавання.

Риба в зграї відіграє головну роль у локальному пошуку в пошуках перспективних областей. За формулою:

$$x_i(t+1) = x_i(t) + rand(-1,1)step_{ind}, \quad (2.1)$$

де $x_i(t)$ та $x_i(t+1)$ зображають стан одиниці i до та після окремого оператора руху. $rand(-1,1)$ - систематично поділене рандомне число, в діапазоні від -1 до 1 та $step_{ind}$ - характеристика, що позначає max переміщення для даного руху. Нова позиція $x_i(t+1)$ узгоджується лише за умови, коли фізична спроможність оператора поліпшується у зв'язку з переміщенням. За умови недотримання вимог, оператор залишається у сталому положенні $x_i(t+1) = x_i(t)$.

У базовому вигляді алгоритму етап самостійного руху допускає цей стан за умови, коли фізична придатність поліпшується. Але у гладкому стані пошуку виникає велика кількість провальних випробувань і у такому випадку є ймовірність розходження алгоритму. Проектна модифікація цієї роботи написана з ціллю отримання найкращих результатів алгоритму.

Задля вирішення даної проблематики, вводиться характеристика α, задля якого $0 \leq \alpha \leq 1$, у роздільній складовій руху. α руйнується експоненціально в парі з циклом та вираховує ймовірність погіршення норми для кожної одиниці. Це вказує на те, щоразу, коли риба робить спроби увійти у стан, обирається рандомне число, та за умови, якщо воно менше за α, можливий рух. Але, лише тільки ті одиниці, котрі проявили поліпшення придатності у діапазоні індивідуального плавання, мають можливість зробити внесок у результати I формули 2.1.

Дана модифікація розроблена задля покращення можливості досліджувати алгоритм. За умови, коли характеристика α експоненційно знищується протягом ітерацій, цей результат інтенсивний тільки на початку процесу пошуку та стає неактуальним після певної кількості ітерацій.

2) Інстинктивно-колективне плавання.

Середні показники деяких переміщень формуються на базі наступного:

$$I = \frac{\sum_{i=1}^{N} \Delta x_i \Delta f_i}{\sum_{i=1}^{N} \Delta f_i} \quad (2.2)$$

Вектор I представляє середньозважене передислокацію усіх риб. Іншими словами, риби, які отримали поліпшення, займатимуться залученням інших риб на своє місцерозташування. За визначенням вектора одиниця риби рухається відповідно:

$x_i(t+1) = x_i(t) + I.$

3) Колективне плавання.

Дана процедура застосовується задля корегування експлуатаційної можливості зграї під час процесу пошуку. Баріцентр B зграї визначається на базі розташування x_i і ваги W_i кожної риби:

$$B(t) = \frac{\sum_{i=1}^{N} x_i W_i(t)}{\sum_{i=1}^{N} W_i(t)}, \quad (2.3)$$

де у випадку, коли вага зграї $\sum_{i=1}^{N} W_i(t)$ збільшується з останнього циклу до сталої, риби прямують до баріцентру згідно рівняння:

$$x_i(t+1) = x_i(t) - step_{vol} rand(0,1) \frac{x_i(t) - B(t)}{distance(x_i(t), B(t))}, \quad (2.4)$$

де у випадку, коли вага зграї не покращується, риби розподіляються від баріцентра відповідно рівняння:

$$x_i(t+1) = x_i(t) + step_{vol} rand(0,1) \frac{x_i(t) - B(t)}{distance(x_i(t), B(t))}, \quad (2.5)$$

де $step_{vol}$ встановлює розмір max переміщення, виконаного цією одиницею. $distance(x_i(t), B(t))$ - евклідова відстань між рибами i розташування і баріцентру зграї. $rand(0,1)$ - рівномірно поділене рандомне значення, в діапазоні від 0 до 1 [22].

За винятком одиниць переміщення, встановлено, що одиниця годування, котра застосовується задля відновлення ваги кожної риби:

$$W_i(t+1) = W_i(t) + \frac{\Delta f_i}{max(|\Delta f_i|)}, \quad (2.6)$$

де $W_i(t)$ - дані ваги риби i, Δf_i - різновид фітнесу нового і останнього стану, $max(|\Delta f_i|)$ - max абсолютний параметр різновиду придатності зі списку риб у косяку. W має здатність до зміни від 1 до $\frac{W_{scale}}{2}$. Ваги усієї зграї визначаються значенням $\frac{W_{scale}}{2}$ [22].

2.2 Основні інструменти розробки системи глобальної пошукової оптимізації аналізу ефективності ЕА

2.2.1 C#

C# — це сучасна універсальна мова програмування, розроблена корпорацією Майкрософт як частина її платформи .NET. C# значно розвинувся протягом багатьох років, ставши однією з найпопулярніших МП у світі [23].

Як багатопарадигмальна мова C# підтримує різні стилі програмування, включаючи об’єктно-орієнтоване, функціональне, імперативне та компонентно-орієнтоване програмування. Ця гнучкість дозволяє розробникам вибирати найбільш прийнятний підхід для своїх конкретних проектів і уподобань кодування. Синтаксис C# походить від C і C++, що робить його знайомим для розробників, які мають досвід роботи з цими мовами. Однак він містить багато особливостей, які відрізняють його, наприклад властивості, делегати, події та LINQ (Language Integrated Query). Ці функції сприяють виразній силі та простоті використання C#.

Одним із ключових аспектів C# є його сильна система типізації, яка допомагає виловлювати помилки під час компіляції, а не під час виконання. Ця функція в поєднанні з потужними можливостями мови обробки винятків сприяє створенню більш надійного коду, який можна підтримувати. C# в основному

використовується для розробки настільних програм Windows, веб-програм та ігор. З появою .NET Core (тепер .NET 5 і далі) C# стає все більш кросплатформним, дозволяючи розробникам створювати програми, які працюють у Windows, macOS і Linux. Мова має багату екосистему бібліотек і фреймворків як від Microsoft, так і від сторонніх розробників. Ця обширна колекція інструментів і ресурсів дозволяє розробникам ефективно вирішувати широкий спектр програмних завдань. Деякі популярні фреймворки та технології, пов’язані з C#, включають ASP.NET для веб-розробки, Unity для розробки ігор і Xamarin для розробки мобільних програм.

У C# велика увага приділяється продуктивності та зручності читання. Такі функції, як автоматичне керування пам’яттю через збирання сміття, визначення типу та підтримка асинхронного програмування, допомагають розробникам писати чистіший і ефективніший код. Мова також підтримує сучасні концепції програмування, такі як узагальнення, лямбда-вирази та типи значень із можливістю нульового значення. Microsoft регулярно оновлює C# новими функціями та вдосконаленнями. В останніх версіях представлено зіставлення шаблонів, типи записів, оператори верхнього рівня та розширені можливості перевірки нульових значень. Ці доповнення продовжують удосконалювати мову та підтримувати її актуальність у динамічному середовищі створення ПЗ. Спільнота C# є великою та активною, з численними ресурсами, доступними для навчання та вирішення проблем. Сюди входить офіційна документація від Microsoft, онлайн-курси, форуми та величезна кількість книг і посібників. Жвавість спільноти сприяє постійному розвитку та прийняттю мови [24].

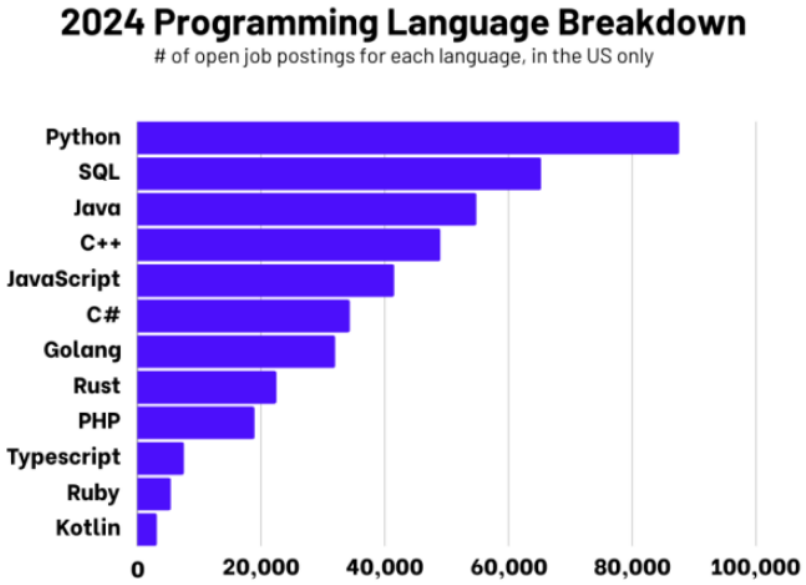


Рис. 2.4 Популярність мов програмування станом на 2024 рік

На рис. 2.5 продемонстровано динаміку актуальності десяти мов програмування за період 2016-2022 роки, за даними ZTM.

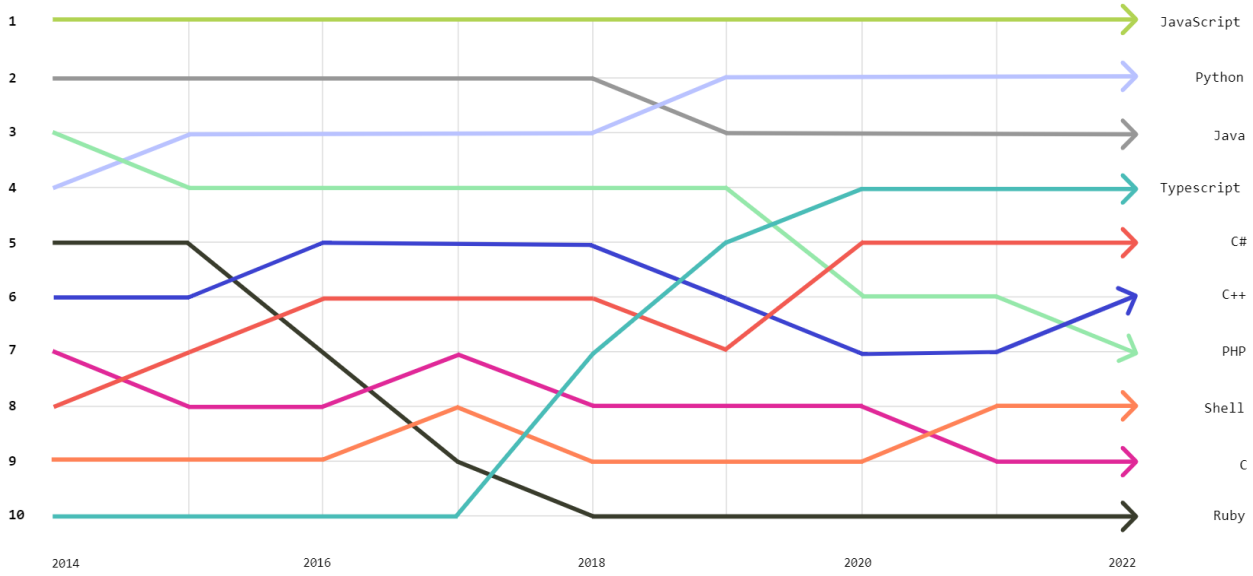


Рис. 2.5 Динаміка актуальності мов програмування, за версією ZTM

За останній час, C# на постійній основі займає місце перших позицій серед мов програмування, за версією TIOBE.

На рис. 2.6 продемонстровано ранжування мов програмування згідно TIOBE.

Programming Language	2023	2018	2013	2008	2003	1998	1993	1988
Python	1	4	8	6	11	24	22	-
C	2	2	1	2	2	1	1	1
C++	3	3	4	3	3	2	2	4
Java	4	1	2	1	1	18	-	-
C#	5	5	5	8	9	-	-	-
JavaScript	6	8	9	9	8	21	-	-
Visual Basic	7	18	-	-	-	-	-	-
PHP	8	7	6	5	6	-	-	-
SQL	9	88	-	-	7	-	-	-
Assembly language	10	13	-	-	-	-	-	-
Ada	24	31	22	20	16	16	7	3
Objective-C	27	12	3	41	48	-	-	-
Lisp	30	29	14	17	15	10	8	2
(Visual) Basic	-	-	7	4	5	3	3	7

Рис. 2.6 Актуальність МП згідно TIOBE

На рис. 2.7 продемонстрована рух мов програмування за критерієм популярності за 22 роки, згідно рейтингу TIOBE.

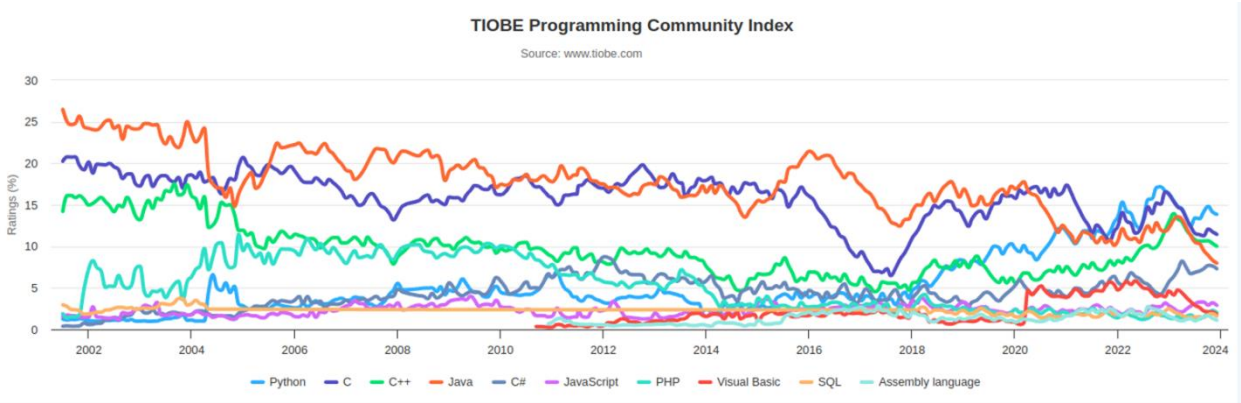


Рис. 2.7 Динаміка поширеності мов програмування, згідно TIOBE

C# замислювалася для того, щоб конкурувати з такою мовою як Java. Експерти відмічають, що дана мова вирізняється середньою складністю.

Це обумовлено тим, що C# - мова високого рівня, її досить легко читати та писати проекти на ній, це робить додає мові актуальності серед новачків та експертів. C# дуже часто застосовують задля автоматизації складних процесів, які потребують велику кількість ресурсів і часу багато часу.

Головні переваги C#:

- Об'єктно-орієнтована мова: C# повністю підтримує принципи об'єктно-орієнтованого програмування.
- Розширена стандартна бібліотека: .NET Framework надає багатий набір попередньо зібраних бібліотек.
- Керування пам’яттю: автоматичне збирання сміття обробляє виділення та звільнення пам’яті.
- Сучасні функції мови: підтримує асинхронне програмування, LINQ.

Головні недоліки C#:

- Проблеми з продуктивністю C#. Коли ми говоримо про продуктивність з точки зору мови програмування та розробки програмного забезпечення, ми вимірюємо цей час через те, скільки триває компіляція, на додаток до швидкості самої програми. У порівнянні з іншими мовами, такими як C++ і C, C# повільніша. Це пояснюється тим, що перед виконанням C# має бути скомпільовано в мову-посередник, а потім у машинну мову.
- Функціональність низького рівня. Якщо потрібно виконувати завдання низькорівневого програмування, як-от взаємодіяти безпосередньо з апаратним забезпеченням, створювати драйвери або возитися з вбудованим програмним забезпеченням, доведеться шукати рішення у іншій мові програмування, оскільки C# не має цієї функції. Але можна використовувати C# для вбудованого програмування та кодування IoT, але справжнє низькорівневе програмування – не головне направлення C# [25].

C# є чудовим вибором для розробки еволюційних алгоритмів через ряд факторів. Мова пропонує багатий набір вбудованих бібліотек і функцій для роботи з різноманітними структурами даних, що спрощує реалізацію складних алгоритмів. Підтримка паралельного програмування в C# дозволяє ефективно використовувати багатоядерні процесори для прискорення обчислень, що є критичним для еволюційних алгоритмів, які часто вимагають значних обчислювальних ресурсів. Інтеграція з .NET середовищем забезпечує доступ до широкого спектру інструментів для аналізу, налагодження та оптимізації коду, що полегшує розробку

і тестування алгоритмів. Крім того, C# добре підходить для розробки додатків з графічним інтерфейсом, що може бути корисним для візуалізації процесу еволюції або результатів. Мова також підтримує об'єктно-орієнтоване програмування, що дозволяє легко створювати модульні та розширювані структури, необхідні для складних систем, таких як еволюційні алгоритми [26].

2.2.2 .NET Framework 4.7.2

.NET Framework 4.7.2, випущений корпорацією Майкрософт у квітні 2018 року, — це всеосяжний і універсальний фреймворк для розробки програмного забезпечення, призначений для полегшення створення широкого спектру програм, від настільних до веб-рішень. Аналіз охоплюватиме компоненти фреймворку, його інтеграцію з Common Language Runtime (CLR) і його роль у розробці сучасного програмного забезпечення. Крім того, у дисертації буде оцінено покращення продуктивності, покращення безпеки та функції сумісності, представлені в цій версії [27].

CLR — це механізм виконання .NET Framework, який відповідає за керування виконанням програм .NET. Він надає основні послуги, такі як керування пам’яттю, безпека та обробка винятків. У версії 4.7.2 CLR було оптимізовано для підвищення продуктивності та стабільності.

BCL — це комплексний набір бібліотек, які забезпечують фундаментальні функції для програм .NET. Ці бібліотеки включають класи для колекцій, файлового введення/виведення, роботи з текстом тощо. Версія 4.7.2 представляє нові API та вдосконалення існуючих, покращуючи продуктивність розробників і продуктивність програм.

.NET Framework підтримує різні моделі програм, зокрема:

- Windows Forms: інструментарій графічного інтерфейсу для створення настільних програм.
- ASP.NET: платформа для створення веб-додатків і служб.
- WPF (Windows Presentation Foundation): графічна підсистема для відтворення інтерфейсів користувача в програмах на базі Windows.

Версія 4.7.2 містить оновлення та вдосконалення цих моделей, покращуючи їхню функціональність і зручність використання.

.NET Framework 4.7.2 широко використовується в корпоративних середовищах для створення надійних і масштабованих програм. Тематичні дослідження висвітлять, як компанії використовують структуру для створення рішень, які відповідають їхнім потребам [28].

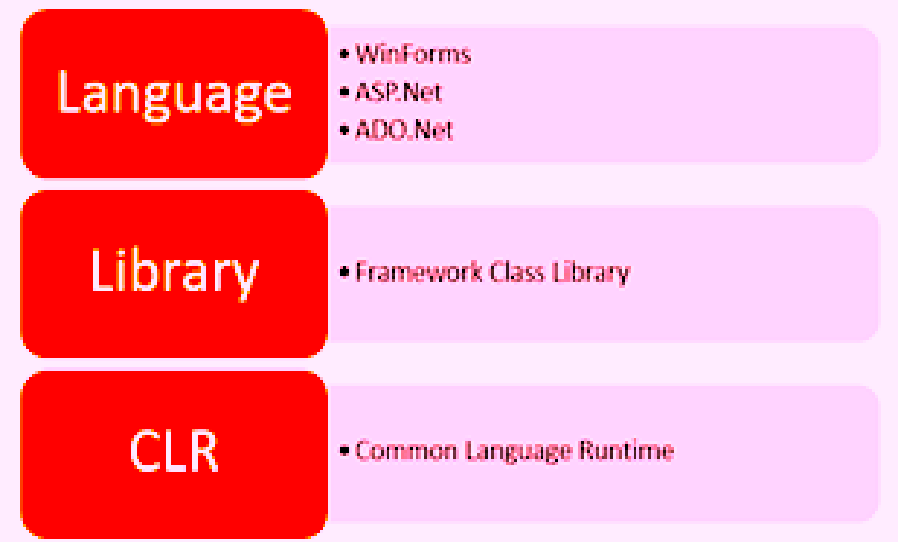


Рис. 2.8 Структура .NET Framework

Переваги .NET Framework 4.7.2:

- Розширені функції безпеки, включаючи підтримку алгоритмів підписання SHA-2.
- Краща підтримка дисплеїв високого DPI та обізнаності DPI на монітор.
- Розширена підтримка криптографії, включаючи додаткові алгоритми та розміри ключів.
- Покращені функції доступності для програм Windows Forms.
- Розширені мережеві можливості, включаючи TLS 1.2 за замовчуванням.
- Доступ до величезної екосистеми бібліотек та інструментів.
- Надійне середовище розробки з інтеграцією Visual Studio.

Недоліки .NET Framework 4.7.2:

- Повільніший цикл випуску оновлень та нових функцій порівняно з .NET Core/5 +.
- Обмежена підтримка сучасних парадигм розвитку, таких як контейнеризація.
- Обмеження продуктивності порівняно з новими реалізаціями .NET.

2.2.3 WF

Windows Forms — це бібліотека класів GUI у середовищі .NET, котра застосовується в цілях створення багатофункціональних настільних програм для Windows. Він забезпечує платформу для розробки програм на стороні клієнта, які працюють в операційній системі Windows, пропонуючи широкий спектр елементів керування, керування макетом і можливостей обробки подій [29]. Ось вичерпний огляд Windows Forms, включаючи її функції, переваги та типові випадки використання:

1. Rich Control Set. Windows Forms містить повний набір вбудованих елементів керування, включаючи кнопки, текстові поля, мітки, прапорці, перемикачі, поля зі списками, поля зі списком, меню, панелі інструментів і сітки даних. Ці елементи керування є будівельними блоками для створення інтерактивних інтерфейсів користувача.

2. Програмування, кероване подіями. Windows Forms підтримує модель програмування, керовану подіями. Елементи керування та форми можуть реагувати на дії користувача, такі як клацання, натискання клавіш та інші взаємодії, обробляючи події, що спрощує реалізацію адаптивних та інтерактивних програм.

3. Прив'язка даних. WF надає потужні можливості зв’язування даних, що дозволяє легко підключати елементи керування до джерел даних: бази даних, колекції, або веб-служби. Це спрощує відображення та керування даними у формах.

4. Графіка та малювання. System.Drawing у WF надає інструменти для візуалізації графіки, зокрема малювання фігур, тексту та зображень. Це корисно

для додатків, які потребують спеціальних візуальних елементів, діаграм або інших графічних елементів [29].

2.2.4 ZedGraph

ZedGraph — це популярна бібліотека діаграм з відкритим кодом для .NET, розроблена, щоб надати розробникам гнучкий і простий у використанні інструмент для створення високоякісних двовимірних лінійних, стовпчастих і секторних діаграм. Відомий своєю простотою та потужними функціями, ZedGraph часто використовується в програмах, які потребують ефективної візуалізації даних. Нижче наведено ключові аспекти ZedGraph, зокрема його функції, переваги та типові випадки використання:

- Високопродуктивний рендеринг. Розроблений для ефективної обробки великих наборів даних, ZedGraph використовує оптимізовані методи візуалізації для підтримки продуктивності навіть при роботі зі складними чи об’ємними даними. Це робить його придатним для додатків даних у реальному часі та сценаріїв, що передбачають візуалізацію великомасштабних даних.
- Гнучкість відкритого вихідного коду. Будучи відкритим кодом, ZedGraph дозволяє розробникам перевіряти, змінювати та розширювати вихідний код відповідно до своїх потреб. Ця гнучкість особливо цінна для проектів із особливими вимогами або для розробників, які хочуть зробити внесок у розвиток бібліотеки.

Переваги використання ZedGraph:

- Простота використання. ZedGraph відомий своєю простотою використання. Бібліотека надає чіткий і добре задокументований API, завдяки чому розробники легко створюють і інтегрують діаграми без тривалого навчання.
- Універсальність. Підтримка бібліотекою різних типів діаграм і параметрів налаштування робить її універсальною. Розробники можуть використовувати ZedGraph у широкому діапазоні додатків, від простих інструментів звітності до складних аналітичних платформ.

ZedGraph пропонує надійне та гнучке рішення для створення діаграм для програм .NET, що поєднує простоту з потужними функціями. Його підтримка кількох типів діаграм, широкі можливості налаштування та природа з відкритим кодом роблять його привабливим вибором для розробників [30].

2.2.5 MS Server

Microsoft SQL Server — середовище управління реляційною БД, розроблена та продана компанією Microsoft. Це комплексна платформа для зберігання, керування та отримання структурованих даних, розроблена для підтримки широкого спектру програм обробки транзакцій, бізнес-аналітики та аналітики.

SQL Server побудований на архітектурі клієнт-сервер. Серверний компонент працює як служба, керуючи файлами бази даних, забезпечуючи безпеку та обробляючи запити клієнтів. Клієнтами можуть бути програми, веб-сервери або інші системи, які підключаються до SQL Server для зберігання або отримання даних. Він зберігає дані в таблицях, які організовані в бази даних. Кожна таблиця складається з рядків (записів) і стовпців (полів). Сервер керує цими структурами на диску, оптимізуючи шаблони зберігання та доступу для продуктивності. Коли клієнт надсилає запит (зазвичай мовою SQL), процесор запитів SQL Server аналізує його, визначає найбільш ефективний спосіб його виконання (оптимізація запиту), а потім виконує операцію.

SQL Server забезпечує цілісність даних через транзакції, сумісні з ACID (Atomicity, Consistency, Isolation, Durability). Він може керувати кількома одночасними транзакціями, забезпечуючи узгодженість даних навіть у середовищах із високим рівнем паралелізму. SQL Server пропонує різні функції для забезпечення безперервності бізнесу:

- Групи доступності AlwaysOn: для високої доступності та аварійного відновлення
- Віддзеркалення бази даних: підтримка гарячого резерву бази даних
- Доставка журналів: автоматизація процесу резервного копіювання та відновлення на вторинному сервері

SQL Server містить кілька функцій для оптимізації продуктивності:

- Індексування: створення структур даних для прискорення пошуку даних
- Кешування плану запиту: зберігання та повторне використання планів виконання
- OLTP у пам’яті: обробка транзакцій у пам’яті для високопродуктивних сценаріїв

Його можна масштабувати від невеликих розгортань з одним сервером до великих розподілених систем. Він підтримує такі функції, як розділення для керування великими таблицями та реплікація для розподілу даних між декількома серверами. Також, він підтримує T-SQL (Transact-SQL), розширення SQL, яке включає конструкції процедурного програмування. Він також інтегрується з різними мовами програмування та фреймворками, дозволяючи розробникам взаємодіяти з базою даних за допомогою знайомих інструментів [31].

2.2.7 MSTest

MSTest, платформа тестування, розроблена Microsoft, є ключовим інструментом у життєвому циклі розробки програмного забезпечення для програм .NET. Цей невід’ємний компонент інтегрованого середовища розробки Visual Studio надає розробникам комплексний набір інструментів і утиліт, призначених для полегшення створення, керування та виконання автоматизованих тестів. Архітектура MSTest побудована на трьох основних компонентах: тестова структура, яка служить основою для написання тестів; виконавець тестування, відповідальний за виконання тесту та звітування про результати; і Test Explorer, вікно Visual Studio, яке пропонує зручний інтерфейс для керування тестами.

Головним у функціональності MSTest є використання атрибутів, які відіграють важливу роль у визначенні елементів тесту та контролі потоку виконання тесту. Ці атрибути, такі як [TestClass], [TestMethod], [TestInitialize] і [TestCleanup], надають декларативні засоби визначення структури та поведінки тесту.

Ця система на основі атрибутів не тільки покращує читабельність коду, але й дозволяє детально контролювати процеси налаштування, виконання та демонтажу тесту.

Підсумовуючи, MSTest представляє зріле та добре інтегроване рішення для тестування для екосистеми .NET. Його поєднання простоти, потужних функцій і повної інтеграції з Visual Studio позиціонує його як цінний інструмент для забезпечення якості та надійності програмного забезпечення.

ВИСНОВОК ДО РОЗДІЛУ 2

У цьому розділі було детально розглянуто технології розробки проекту, зосереджуючись на еволюційних алгоритмах глобальної пошукової оптимізації. Зокрема, було проаналізовано генетичний алгоритм (GA) та модернізований алгоритм пошуку косяка риб (FSS), які є фундаментальними для розробки проекту. Основною мовою програмування для реалізації проекту було обрано C#. Цей вибір обґрунтовано багатьма перевагами мови, включаючи її об'єктно-орієнтовану природу, безпечну типізацію, крос-платформну підтримку та багату стандартну бібліотеку. C# особливо добре підходить для розробки еволюційних алгоритмів завдяки своїй продуктивності та зручності у використанні. Для розробки проекту було обрано ряд ключових технологій та фреймворків:

1. .NET Framework 4.7.2 як основа для розробки.
2. Windows Forms (WF) для створення графічного інтерфейсу.
3. .NET Charting та ZedGraph для візуалізації даних та створення графіків.
4. Бібліотека RegularExpressions для обробки та валідації текстових даних.
5. Microsoft SQL Server для управління базами даних.
6. MSTest для автоматизованого тестування.

Кожна з цих технологій була обрана з урахуванням її специфічних переваг та можливостей, які вони надають для ефективної реалізації проекту з еволюційними алгоритмами. Використання цих інструментів та технологій дозволяє створити потужну, гнучку та ефективну систему для реалізації та дослідження еволюційних алгоритмів глобальної пошукової оптимізації. Це забезпечує надійну основу для подальшої розробки та вдосконалення проекту, а також для проведення експериментів та аналізу результатів.

3 РОЗРОБКА КОМПОНЕНТІВ І АНАЛІЗ СТВОРЕНОГО СЕРЕДОВИЩА

3.1 Основні платформні компоненти генетичного алгоритму

3.1.1 Class BaseSpecies

Необхідно, щоб усі типи були наслідуваними початкового класу *BaseSpecies <TSpecies>*. Необхідно, щоб *TSpecies* був наслідуваний класом *BaseSpecies <TSpecies>*.

BaseSpecies - метафізичний клас, він представляє статичні protected-типи симбіозу та процедури мутації. Наступним кроком розглянемо головні класи.

Наприклад, *abstract public void TestChromosomes().* Даний клас визначає локальну захищену булеву змінну *m_Dead*, вона демонструє суміжність хромосоми за обмеженнями, вони на них накладають (у нашому випадку, визначають потрапляння даних хромосом у діапазоні [-5.0; 5.0]). Коли дані хромосом відповідають очікуванням, клас *m_Dead* визначає маркер *false*. Ця змінна (формується на базі даних Dead) формації видаляє невдалі одиниці.

Черговий метод, котрий потрібно застосувати у наслідуваномц класі - *abstract public TSpecies Cross.* Даний метод формує тип, проводячи симбіоз себе і другого типу, другий тип визначається як аргумент. Як правило, коли одиниця має список хромосом, симбіоз хромосоми одного типу, інакше кажучи, проводимо симбіоз *a0* і *a0* іншого типу, і таким самим чином проводимо симбіоз *a1* та *a1*, у такому випадку, немає резону проводити симбіоз *a0* та *a1* іншого класу.

Ціль симбіозу містить у собі складову хромосоми першого типу та другу складову другого типу, та формується новостворена хромосома. Як правило, хромосоми проводять симбіоз побітово. Мета цього охоплює рандомно обрану точку розриву (симбіозу) хромосоми, другим етапом, формування нової хромосоми, вона містить ліву складову першої хромосоми та праву складову іншої.

BaseSpecies містить public статичні методи у цілях симбіозу хромосом: *Double* та *Int64*. Проаналізуємо більш детально два методи. У даній ситуації існує 2 точки перерізу - у середині слова та знак числа, іншими словами, на початку відбувається симбіоз хромосом біт за бітом.

Роботу даного методу можна охарактеризувати так: форматуємо хромосоми зі стану *Double* у тип *Int64*. Це необхідність, з ціллю отримання можливості проводити симбіоз побітово. Після симбіозу типу *Int64* без уваги знаку, форматуємо число у *Double*, використовуємо знак 1-ї з 2-х хромосом. Метод симбіозу хромосом стану Double продемонстровано на рисунку 3.1.

Процес симбіозу типів *Int64*. Як можна побачити видно на рисунку 3.2, на початку відбувається процес симбіозу хромосоми без врахування знаку. Проводять порівняння знаку конкретної хромосоми з результатом, та у випадку, за умови, що символи не ідентичні - символ наслідку форматується на ідентично протилежний.

```
/// <summary>
        /// Схрестити дві хромосоми типу double
        /// </summary>
        /// <param name="x">1-а хромосома</param>
        /// <param name="y">2-а хромосома</param>
        /// <returns>Нова хромосома</returns>
        static protected double Cross(double x, double y)
        {
            Int64 ix = BitConverter.DoubleToInt64Bits(x);
            Int64 iy = BitConverter.DoubleToInt64Bits(y);

            double res = BitConverter.Int64BitsToDouble(BitCross(ix, iy));

            if (m_Rnd.Next() % 2 == 0)
            {
                if (x * res < 0)
                {
                    res = -res;
                }
            }
            else
            {
                if (y * res < 0)
                {
                    res = -res;
                }
            }

            return res;
        }

```

Рис. 3.1 Виконання процедури симбіозу хромосом значення Double

```
/// <summary>
/// Схрестити побітово без урахування знаку дві хромосоми типу Int64
/// </summary>
/// <param name="x">1-а хромосома</param>
/// <param name="y">2-а хромосома</param>
/// <returns>Нова хромосома</returns>
static protected Int64 BitCross(Int64 x, Int64 y)
{
    // Число біт, що залишилися зліва від точки перетину хромосом
    int Count = m_Rnd.Next(62) + 1;
    Int64 mask = ~0;

    mask = mask << (64 - Count);

    return (x & mask) | (y & ~mask);
}

/// <summary>
/// Схрестити побітово з урахуванням знака дві хромосоми типу Int64
/// </summary>
/// <param name="x">1-а хромосома</param>
/// <param name="y">2-а хромосома</param>
/// <returns>Нова хромосома</returns>
static protected Int64 Cross(Int64 x, Int64 y)
{


    Int64 res = BitCross(x, y);

    if (m_Rnd.Next() % 2 == 0)
    {
        if (x * res < 0)
        {
            res = -res;
        }
```

Рис. 3.2 Виконання процедури симбіозу хромосом значення Int64

Розберемо черговий метод *public void Mutation()*. Мутація впливає на одну хромосому. Чисто теоретично, протягом процесу мутації можуть відбуватися будь-які зміни. У даному вигляді мутація проводить процес змінення один біт у слові, продемонстровано на рис. 3.3.

```
/// <summary>
/// Мутація для типу double
/// </summary>
/// <param name="val">Значення, яке мутуємо</param>
/// <returns>Промутоване значення</returns>
static protected double Mutation(double val)
{
    UInt64 x = BitConverter.ToUInt64(BitConverter.GetBytes(val), 0);
    UInt64 mask = 1;
    mask <<= m_Rnd.Next(63);
    x ^= mask;

    double res = BitConverter.ToDouble(BitConverter.GetBytes(x), 0);

    return res;
}

/// <summary>
/// Мутація для типу Int32
/// </summary>
/// <param name="val">Мутиоване значение</param>
/// <returns>Мутоване значение</returns>
static protected Int32 Mutation(Int32 val)
{
    Int32 mask = (1 << m_Rnd.Next(31));

    return val ^ mask;
}

/// <summary>
/// Мутація для типу Int64
/// </summary>
/// <param name="val">Значення, яке мутуємо</param>
/// <returns>Промутоване значення</returns>
static protected Int64 Mutation(Int64 val)
```

Рис. 3.3 Виконання процедури static задля мутації

Процес мутації проходить не часто. Як приклад, досить часто вірогідність мутації складає 5-10%, деколи можна довести цей показник до 30%. У багатьох випадках, у ролі мутації для дрібних чисел, найкращим чином буде використання не продемонстрованої вище функції, а за допомогою звичайного генератору рандомних значень, він формує рандомне значення у заданому діапазоні. У конкретному прикладі для дрібних чисел таким чином реалізовано, для цілих хромосом застосовується мутація, продемонстрована трохи вище. На Рис. 3.4 продемонстровано діаграму *BaseSpecies.*

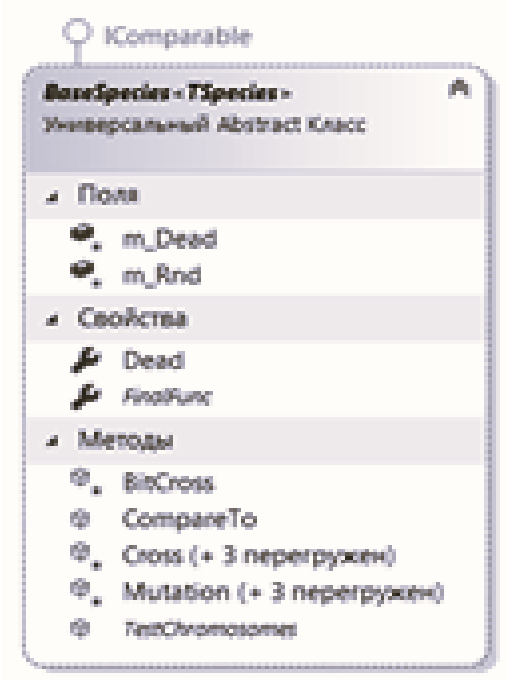


Рис. 3.4 Діаграма типу UML, class *BaseSpecies*

3.1.2 Class Population

Проаналізуємо клас формації, у якому проживають агенти, розмножуються та знищуються. Він містить generic, котрий, у свою чергу, є типом, котрий має бути наслідуваним від BaseSpecies <TSpecies>

Першим чином, необхідно розібрати властивості, котрі потребують налаштування роботою з алгоритмом:

- *MaxSize (Int32)* – гранично-допустима чисельність типів, котра вміщає формація. Даний розмір, до якого обмежується формація після симбіозу. За замовчуванням гранично-допустима чисельність видів - 500.
- *CrossPossibility (Double)* – вірогідність симбіозу. Вірогідність має відповідати діапазону (0,0; 1.0]. У випадку, коли дана умова не відповідає, випадає виключення *ArgumentOutOfRangeException*. За сталими параметрами – значення 0.95.
- *MutationPossibility (Double)* - вірогідність схрещування. Воно має відповідати діапазону значень [0,0; 1.0). Коли дана умова не відповідає - випадає виключення *ArgumentOutOfRangeException*.

Загалом, чисто в теорії, є можливість реалізувати вірогідність відбору, але у нашому випадку, вірогідність дорівнює 1.0. При інакшому сценарії, існувала б вимога знищення видів з формації, з деякою вірогідністю. Знищені види (види, у яких хромосоми не потрапляють у точно визначений інтервал) необхідно вилучати у будь-якому сценарії.

public void Add - формування типу у формації. Необхідно власноруч переміщати потрібну чисельність типів. На старті роботи алгоритму необхідна наявність, як мінімум, двох видів формації. Кількість типів у формації буває менша, порівняно з визначеним значенням *MaxSize*. Коли після процесу симбіозу масштаби формації менше *MaxSize* - знищуються дефектні види (мертві у тому числі).

Для того, щоб здобуту чергову формацію, потрібно застосувати метод *void NextGeneration()*. Процес виконання даного методу, як правило, займає велику кількість часу, через це, його бажано застосувати із індивідуального потоку. Наступним кроком, потрібно подивитися, що даний метод демонструє.

Програмну частину методу *NextGeneration(),* котрий містить у собі оновлення формації, продемонстровано на рис. 3.5.

```
/// <summary>
/// Оновити популяцію (отримати наступне покоління)
/// </summary>
virtual public void NextGeneration()
{
    if (m_Generation == 0 && m_Species.Count == 0)
    {
        throw new ZeroSizePopulationException();
    }

    // Спочатку схрестимо
    Cross();

    // Промутуємо і заодно перевіримо всі хромосоми
    foreach (TSpecies species in m_Species)
    {
        // Якщо треба мутувати з урахуванням ймовірності
        if (m_Rnd.NextDouble() <= m_MutationPossibility)
        {
            species.Mutation();
        }

        species.TestChromosomes();
    }

    // Відбираємо найживучіші види
    m_Species.Sort();
    Selection();

    m_Generation++;
}
```

Рис. 3.5 Програмна демонстрація методу відновлення формації

На старті метод запускає перевірку, яка аналізує, щоб при першому виклику методу (нульове покоління) масштаби формації мали більше ніж 2 (у іншому випадку – відсутні одиниці для симбіозу). У випадку невідповідності випадає виключення *SmallSizePopulationException*. Наступним чином, починається симбіоз типів, продемонстровано на рис. 3.6.

```
/// </summary>
protected void Cross()
{
    // Розмір до початку поповнення популяції (щоб не схрещувати нові види,
    // які додаються в кінець списку)
    Int32 OldSize = m_Species.Count;

    // Номер пари для схрещувати виду
    Int32 Count = m_Species.Count;

    for (int i = 0; i < Count; ++i)
    {
        // Якщо треба схрещувати з урахуванням ймовірності
        if (m_Rnd.NextDouble() <= m_CrossPossibility)
        {
            // Додаємо в список виду, отриманий схрещуванням чергового виду і
            // виду з випадковим номером
            m_Species.Add(m_Species[i].Cross(m_Species[m_Rnd.Next(OldSize)]));
        }
    }
}
```

Рис. 3.6 Програмна демонстрація методу отримання нових видів симбіозу

При симбіозі перебираємо усі види та проводимо даних процес з визначеною вірогідністю симбіозу з іншими типами, вони визначаються рандомним чином.

Після симбіозу йде етап мутації типів із визначеною вірогідністю, після симбіозу відбувається тестування хромосом. Далі відбувається процес сортування типів за значенням головної функції. Метод *Sort* ініціалізовано у класі *BaseSpecies*.

```
/// <summary>
    /// Провести відбір найбільш "живучих" видів
    /// </summary>
    protected void Selection()
    {
        // Скільки видів треба видалити
        Int32 Count = m_Species.Count - m_MaxSize;

        for (Int32 i = 0; i < Count; ++i)
        {
            m_Species.RemoveAt(m_Species.Count - 1);
        }
    }
```

Рис. 3.7 Програмна демонстрація методу відбору «найживучіших» видів

На Рис. 3.8 продемонстровано діаграму class *Population* (властивості і методи).

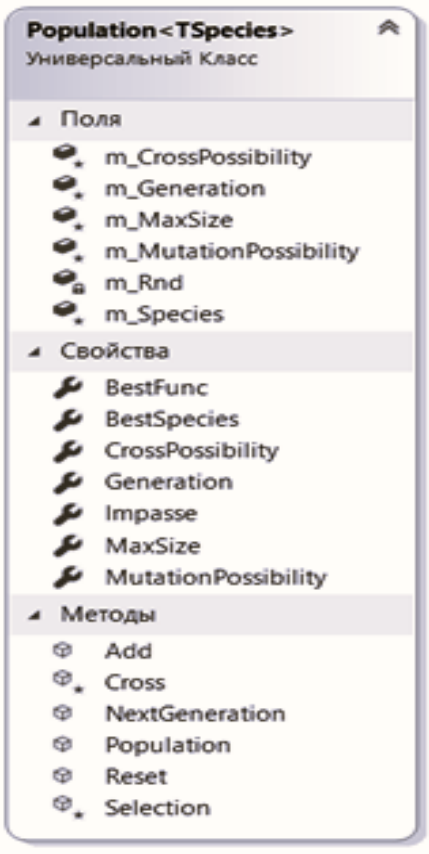


Рис. 3.8 Діаграма класу *Population*

3.1.3 Class BaseDoubleSpecies

Class *BaseSpecies* абстрагується від визначеного втілення одиниць, від того, що представляють дані хромосоми (дробові, цілі числа, масиви та екземпляри інших класів).

Але дуже часто застосовується GA для цілей мінімізації математичних функцій, залежних від чисельності змінних дрібного типу. У даній ситуації хромосоми зручно ініціалізувати у вигляді масиву дрібних чисел, тобто, кожен елемент масиву - це одна хромосома. Також, вважаємо, що розмір даного масиву залишається статичним для усіх одиниць під час усього часу роботи алгоритму.

Для цієї задачі був написаний клас *BaseDoubleSpecies*. Він надає можливість скорочення програмного коду, котрий впроваджує алгоритм.

Виклик класу:

```
public abstract class BaseDoubleSpecies<TSpecies> :
        BaseSpecies<BaseDoubleSpecies<TSpecies>>, ICloneable
        where TSpecies : BaseDoubleSpecies<TSpecies>
```

Рис. 3.9 Виклик класу *BaseDoubleSpecies*

TSpecies - визначений вид оператора.

Class *BaseDoubleSpecies* - метафізичний, але даний клас відповідає за роботою над класом одиниць, проте додає свій метафізтчний метод, в цьому методі реалізується розрахунок головної функції. Дана функція має назву *CalcFinalFunc()*, за її задумкою, вона займається поверненням типу *double*.

Для того, щоб при абсолютно кожному порівнянні одиниці з іншою, не потрібно було розраховувати значення головної функції, розраховується вона 1 раз у конструкторі особини. Отримане значення головної функції зберігається у змінній *m_FuncVal*.

class *BaseDoubleSpecies* вміщає дані масиви:

- *m_Chromosomes* - хромосоми. Задля отримання конекту до даного типу заздалегідь передбачено властивість *Cromosomes*.

- *m_Intervals* - сталий масив класів *Interval*. Використовуючи даний клас, можна задати діапазони зміни кожної хромосоми. Для конекту до цього масиву існує властивість *Intervals*. Дана властивість ініціалізується до етапу заповнення формації. За умови, що при формуванні одиниці одна хромосома потрапляє у свій діапазон - вона визначається мертвою. Масштаби даного масиву повинні бути ідентичні розміру масиву *m_Chromosomes*. Class *BaseDoubleSpecies* представляє усі етапи GA, котрі не були представлені у класі *BaseSpecies:* симбіоз та мутація. Class *BaseDoubleSpecies* містить 2 компоненти:

```
public BaseDoubleSpecies()
public BaseDoubleSpecies(double[] chromosomes)
```

Рис. 3.10 Компоненти class *BaseDoubleSpecies*

Перший формує одиницю з рандомними значеннями хромосом, це досить зручно застосовувати при стартовому наповненні формації. Другий конструктор застосовується в центрі самого класу *BaseDoubleSpecies* задля процесів симбіозу та мутації.

На рис. 3.11 продемонстровано діаграму class *BaseDoubleSpecies.*

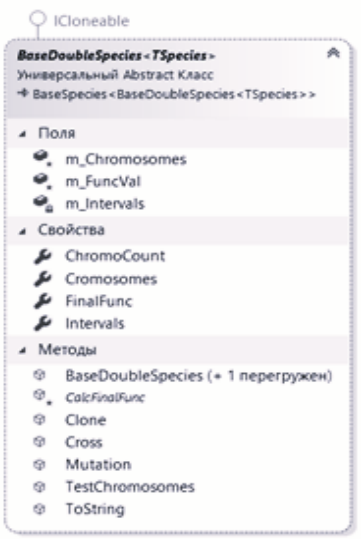


Рис. 3.11 UML діаграма class *BaseDoubleSpecies*

3.1.4 Class Interval

Class *Interval* належить до допустимої області пошуку і вміщає такі властивості і методи:

- *MinValue* - min значення діапазону.
- *MaxValue* - max значення діапазону.
- *Interval(double minval, double maxval)* – компонент з 2-ма параметрами.
- *IsInside (double val)* – вірогідність отримання значення у конкретному діапазоні.

3.1.5 Class Analytics

Class *Analytics* відповідає за відстеження роботи GA. Даний клас виконує функцію збереження найкращої одиниці з кожної генерації. Клас *Analytics* відповідає за роботу з класами, наслідуваними від *BaseDoubleSpecies*.

Виклик класу A*nalytics*:

- Написати екземпляр класу *Analytics <TSpecies>*, *TSpecies* - даний клас наслідуваний від *BaseDoubleSpecies*.
- Перед етапом старту алгоритму видалити статистику за допомогою методу *Clear().*
- На кожній ітеративній обробці алгоритму необхідно викликати метод *Add()*, передаючи йому шаблон найкращої одиниці.
- Під час виконання етапу вираховування є можливість звертатися до списку збережених одиниць, через метод *BestSpecies*.
- Якщо застосовувати метод *ToString()*, є можливість збереження історії, у вигляді текстової таблиці (кожен рядок відповідає одній генерації, кожен стовпець - одній дробовій хромосомі). Останній стовпець передає значення головної функції.

На рис. 3.13 продемонстровано діаграму class *Analytics.*

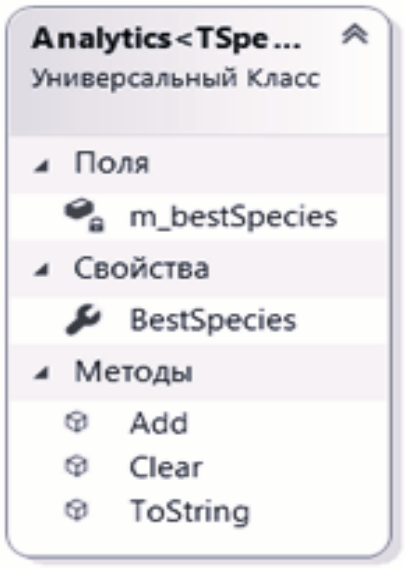


Рис. 3.12 UML діаграма class *Analytics*

3.2 Програмні елементи алгоритму Fish School Search

3.2.1 Class FishPointUniversal

Class *FishPointUniversal* – даний клас відповідає за статичні методи задля пошуку евклідової відстані між рибами, методи головних операцій над особинами.

Обсяги, які відповідають за стан риб - структурні типи n-мірної точки, для яких визначені перелічені етапи: додавання / віднімання двох точок, додавання / віднімання точки та числа, множення / ділення точки та числа, порівняння точок. Вважаємо дані обсяги за векторні типи даних.

Проаналізуємо дані методи:

- *FishPointUniversal(params double[] coords)* – компонент, якому відправляються координати одиниць.
- *FishPointUniversal(int dimensionSize)* – компонент, якому відправляються масштаби формації (зграї риб).
- *GetCoords()* – метод, котрий відповідає за процес повернення масиву з координатами операторів.

- *EuclidDistance(FishPointUniversal p1, FishPointUniversal p2)* – сталий метод розрахунку евклідової відстані між одиницями, він необхідний під час стадії колективного плавання.
- *FishPointUniversal operator - (FishPointUniversal p1, FishPointUniversal p2)* – сталий метод, який реалізовує операцію віднімання векторів, котрі відображають рибин. Програмна демонстрація методу на рис. 3.14.

```
public static FishPointUniversal operator -(FishPointUniversal p1, FishPointUniversal p2)
{
    if (p1.vector.Length != p2.vector.Length)
        throw new IndexOutOfRangeException();
    else
    {
        double[] to_return = new double[p1.vector.Length];
        for (int i = 0; i < to_return.Length; i++)
        {
            to_return[i] = p1[i] - p2[i];
        }
        return new FishPointUniversal(to_return);
    }
}
```

Рис. 3.13 Програмна демонстрація методу *FishPointUniversal*

- *FishPointUniversal operator +(FishPointUniversal p1, double number)*– сталий метод, який здійснює етап додавання векторів, котрі (риб).
- *FishPointUniversal operator *(FishPointUniversal p1, double factor)* – сталий метод, який здійснює етап збільшення вектора на змінну *factor*. Метод необхідний задля калькуляція середнього значення індивідуальних переміщень операторів і задля встановлення центру ваги.
- *FishPointUniversal operator /(FishPointUniversal, double divisor)* – стала властивість, яка проводить етап збільшення вектора на змінну *divisor*. Зазначена властивість необхідний задля встановлення центру ваги системи і скорочення величини кроку індивідуального пошуку.
- *IsInArea(FishMarkUniversal lowerBound, FishMarkUniversal higherBound)* – властивість, котра проводить процес повернення *true*, за умови, що одиниця у границях області пошуку, у іншій ситуації – *false*.

На рис. 3.15 графічно продемонстровано діаграму class *FishMarkUniversal.*

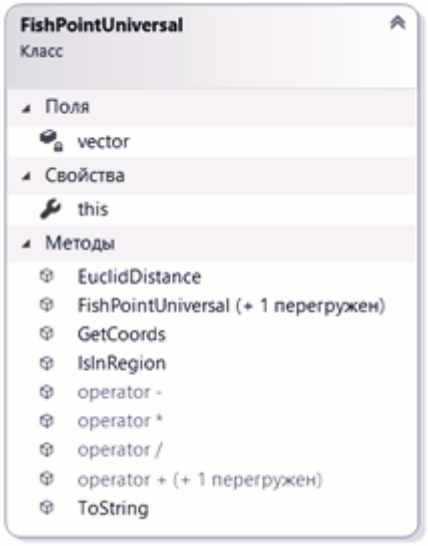


Рис. 3.14 UML діаграма class *FishPointUniversal*

3.2.2 Class FishUniversal

Class *FishUniversal* – даний клас представляє методи, які втілюють у життя головну складову алгоритму. Він представляє методи, які представляють усі етапи плавання риб, пошук max значення дельта-функції, пошук і пошук барицентра системи, пошук сумарної ваги усіх операторів і годування риб.

Якщо починати з етапу міграції риб, котра проводиться поітераційно, під час кожної з ітеративних обробок застосовуються оператори 2-х груп:

- Одиниці плавання, які гарантують переміщення агентів у рамках акваріума.
- Одиниці годування, фіксують досягнення дослідження усіх областей акваріума.

Параметри акваріуму (інакше кажучи, область пошуку) та його мешканці. Було введено перелік змінних, притаманні для усього акваріума:

- *populationSize* – масштаби формації (чисельність риб у зграї).
- *iterationCount* – чисельність ітеративних обробок під час етапу «Міграції операторів».
- *lowerBoundPoint, higherBoundPoint* – верхня та нижня границі пошуку.

- *individStepStart, individStepFinal* – надає стартовий та фінальний радіус пошуку ресурсів поруч з рибами.
- *weightScale* – максимальна вага риби.

Дані параметри повинен задавати користувач. З їх допомогою контролюється співвідношення точність / час роботи алгоритму.

Визначення агентів проводиться методом *InitializeData()*, його програмна реалізація продемонстрована на рис. 3.16. При визначенні одиниць, рандомним чином обираємо розташування агентів у допустимих межах.

Основні етапи ітераційної обробки:

- *individStep* – дані самостійного етапу зміщення під час конкретної обробки.
- *individDeltaFitnessMax* – max змінення фітнес-функції.
- *instinctAverage* – середнє арифметичне, після проходження етапу інстинктивного плавання.
- *instinctSumWeightOld* – вага операторів після інстинктивного етапу у минулій обробці.
- *instinctSumWeightNew* – вага операторів після етапу інстинктивного плавання у сталій обробці.
- *willStep* – етап вольового зміщення.
- *barycentre* – фокус тяжіння операторів, застосовується під час вольового етапу.
- *individDeltaPoint* – переміщення після етапу індивідуального плавання.
- *individDeltaFitness* – ротація функції після проходження індивідуального етапу.
- *weight* – вага оператора, за замовчуванням 50% від максимального значення *weightScale* / 2.

```
public static void InitializeData(UserFunctions.FuncTemplate _function, int _dimensionSize, out FishUniversal[] _fishes,
                                  int _popsize, int _iterationCount, FishPointUniversal _lowerBound, FishPointUniversal _higherBound,
                                  double _individStepStart = 0.5, double _individStepFinish = 0.1, double _weightScale = 10)
{
    funcTemplate = _function;
    dimensionSize = _dimensionSize;
    populationSize = _popsize;
    iterationCount = _iterationCount;
    lowerBoundPoint = _lowerBound;
    higherBoundPoint = _higherBound;
    individStepStart = _individStepStart;
    individStepFinal = _individStepFinish;
    individStep = individStepStart;
    weightScale = _weightScale;
    instinctSumWeightOld = populationSize * weightScale / 2;
    _fishes = new FishUniversal[populationSize];
    for (int fishIndex = 0; fishIndex < _fishes.Length; fishIndex++)
    {
        _fishes[fishIndex] = new FishUniversal();
        // Ініціалізуємо положення риб виходячи з арності функції
        for (int swimState = 0; swimState < 4; swimState++)
            _fishes[fishIndex].swimStatePoint[swimState] = new FishPointUniversal(dimensionSize);
        for (int dimensionIndex = 0; dimensionIndex < dimensionSize; dimensionIndex++)
            // Вибираємо початкове положення для 0-й стадії
            _fishes[fishIndex].swimStatePoint[0][dimensionIndex] = lowerBoundPoint[dimensionIndex] +
                (higherBoundPoint[dimensionIndex] - lowerBoundPoint[dimensionIndex]) * randGen.NextDouble();
    }
}
```

Рис. 3.15 Програмна демонстрація методу *InitializeData()*

Самостійні параметри для кожної одиниці знаходяться у масиві *swimStatePoint = new FishPointUniversal [4]*:

- 0 – старт індивідуального плавання.
- 1 – стан після етапу індивідуального плавання.
- 2 – стан після етапу інстинктивного плавання.
- 3 – кінцеве розташування (після проходження етапу колективного плавання).

Підсумовуючи, першим чином відбувається ідентифікація усієї формації: рандомне розташування позиції риби у границях акваріума (*swimStatePos [0])* та встановлення ваги, котра відповідає половині від max (*weight = weightScale* / 2) для усього списку агентів.

Наступним чином, вступає у силу головний цикл алгоритму, цей цикл уособлює етап «Міграція риб до джерела живлення». Як характеристика зупинки у роботі застосовується величина чисельності ітеративних обробок (*iterationCount*).

Наступним кроком, відбувається індивідуальне плавання риб, дана стадія застосована у методі *IndividSwim()*, програмна реалізація методу продемонстрована на рис. 3.17. Дана стадія вирізняється тим, що усі агенти у деякій області навколо себе (*individStep*) роблять спроби у пошуках найкращого значення функції. Якщо успіх здобуто – даний крок фіксується. У інших ситуаціях, вважаємо – даної ротації

не було. На рис. 3.17 продемонстровано параметр α, котрий розділяється експоненціально разом з циклом. В усіх випадках, коли агент робить спробу переходу у стан, який не покращує її придатність, обирається рандомне чином число randomNum, за умови, якщо воно менше за α, рух - дозволено.

```
/// <summary>
/// Стадія індивідуального плавання
/// </summary>
public void IndividSwim()
{
    // Даємо рибкам шанс зробити переміщення
    for (int dimensionIndex = 0; dimensionIndex < dimensionSize; dimensionIndex++)
        this.swimStatePoint[1][dimensionIndex] = this.swimStatePoint[0][dimensionIndex] + (-1 + randGen.NextDouble() * 2) * individStep;
    //Delta X и F(x)
    this.individDeltaPoint = this.swimStatePoint[1] - this.swimStatePoint[0];
    this.individDeltaFitness = Fitness(this.swimStatePoint[1].GetCoords()) - Fitness(this.swimStatePoint[0].GetCoords());
    // Перевіряємо переміщення на ефективність
    if (!this.swimStatePoint[1].IsInRegion(lowerBoundPoint, higherBoundPoint) | this.individDeltaFitness > 0)
    {
        // Якщо переміщення неефективне або неможливе, то вважаємо, що його не було
        this.swimStatePoint[1] = this.swimStatePoint[0];
        this.individDeltaPoint = new FishPointUniversal(dimensionSize);
        this.individDeltaFitness = 0;
    }
```

Рис. 3.16 Програмна демонстрація методу *IndividSwim()*

Потрібно зафіксувати результати індивідуального плавання. Для цього застосовується параметр «вага». Цей параметр відповідає зміні функції пристосованості для визначеного оператора до та після індивідуального плавання, нормованого max значенням функції формації. Даний випадок - це відмінна характеристика алгоритму, тому що відсутні зобов’язання запам'ятовування найкращих операторів на попередніх ітеративних обробках.

Далі агенти переходять до наступного етапу - інстинктивно-колективного плавання, воно впроваджене у методі *InstinctSwim()*, програмна демонстрація методу висвітлена на рис. 3.18. Для усієї зграї риб визначається величина «загальний етап міграції». Сенс цього етапу: на кожну рибу впливає уся формація, при цьому значення окремої одиниці пропорційно її успішної реалізації у індивідуальному плаванні. Після цього, уся формація переміщується на нараховану величину *instinctAverage*.

```
/// <summary>
    /// Стадія інстинктивного плавання
    public void InstinctSwim()
    {
        this.swimStatePoint[2] = this.swimStatePoint[1] + instinctAverage;
    }
```

Рис. 3.17 Програмна демонстрація методу *InstinctSwim()*

Перед наступним етапом потрібно виконати проміжні кроки: вирахувати центр тяжіння усієї зграї (метод *SearchBarycentre(FishUniversal[] fishes)*). Баріцентр зграї вираховується на базі розташування *swimStatePoint* і ваги *weight* кожної одиниці.

Тепер оператори можуть перейти до фінального етапу: колективно-вольового плавання, яке було програмно реалізовано у методі *CollectiveSwim()*, програмна демонстрація методу на рис. 3.19. Потрібно отримати наступну інформацію: зміна ваги формації у порівнянні з минулою ітерацією. У випадку, якщо формація збільшилася - це означає, що вона наблизилася до області мінімуму функції, у такому випадку, потрібно зменшити діапазон пошуку, цей крок допоможе виявити інтесифікаційні властивості. Також це працює у інверсійному порядку: за умови, що вага зграї зменшилась, агенти починають шукають мінімум в поганій позиції, тому потрібно змінити направлення траєкторії та виявити диверсифікаційні властивості.

Обсяг *swimStatePoint[3]* у даній формулі відповідальний за етап вольового переміщення. Рекомендується застосовувати значення, у два рази більше індивідуального кроку пошуку. *euclidD* визначає відстань між 2-ма точками в евклідовому просторі.

```
/// <summary>
    /// Стадія колективного плавання
    public void CollectiveSwim()
    {
        double euclidD = FishPointUniversal.EuclidDistance(this.swimStatePoint[2], barycentre);
        if (euclidD == 0)
        {
            this.swimStatePoint[3] = this.swimStatePoint[2];
        }
        else if (instinctSumWeightNew > instinctSumWeightOld)
        {
            this.swimStatePoint[3] = this.swimStatePoint[2] - (this.swimStatePoint[2] - barycentre) * willStep * randGen.NextDouble() / euclidD;
        }
        else
        {
            this.swimStatePoint[3] = this.swimStatePoint[2] + (this.swimStatePoint[2] - barycentre) * willStep * randGen.NextDouble() / euclidD;
        }
    }
```

Рис. 3.18 Програмна демонстрація методу *CollectiveSwim()*

3.3 Аналіз цільової ефективності модифікованого алгоритму FSS

Значення розміру формації під час роботи алгоритму FSS продемонстрований у табл. 3.1.

Таблиця 3.1

Значення масштабів формації на час роботи алгоритму

Масштаби формації	Термін виконання алгоритму, (с)
5	0,013
10	0,008
25	0,007
35	0,007
50	0,007
75	0,008
100	0,005
200	0,01
500	0,012
1000	0,02

Аналіз ефекту масштабів формації під час виведення результату алгоритму будувалося на базі пошуку екстремуму функції $y = 0{,}1(x^2 + z^2) + 10$, найбільш ефективні показники продемонстровані у таблиці 3.1.1. З цієї таблиці зрозуміло, що оптимальні масштаби формації – 1 сотня одиниць. Максимальний час отримання результату алгоритму за умови малої формації – 5 одиниць, при великій формації – 1000 і 500 одиниць.

Задля дослідження швидкості пошуку від значення max ваги оператора, визначена функція $y = 3x^2 + xz + 2z^2 - x - 4z$. Результати аналізу продемонстровані у таблиці 3.2.

Таблиця 3.2

Значення максимальної ваги оператора на час роботи алгоритму

Вага оператора	Час виконання алгоритму, (с)
5	0,012
10	0,012
25	0,012
35	0,012
50	0,011
75	0,011
100	0,013
200	0,015
500	0,016
1000	0,014

Таблиця 3.2 демонструє оптимальну вагу риби – 75. У таблиці 3.3. продемонстровано вплив чисельності ітеративних процесів на точність та час завершення алгоритму, задля функції $y = 0{,}1(x^2 + z^2) + 10$. У таблиці 3.3. продемонстрована точність виконання алгоритму збільшується з чисельністю ітеративних обробок. Час виконання алгоритму також збільшується з чисельністю ітеративних процесів.

Таблиця 3.3

Значення чисельності ітеративних обробок на точність та час виконання алгоритму

Чисельність ітеративних обробок	Точність виконання алгоритму, (%)	Час виконання алгоритму, (с)
15	99,9412	0,007
25	99,97482	0,045
35	99,987	0,055

Продовження таблиці 3.3

Значення чисельності ітеративних обробок на точність та час виконання алгоритму

Чисельність ітеративних обробок	Точність виконання алгоритму, (%)	Час виконання алгоритму, (с)
50	100	0,056
75	100	0,033
100	100	0,038
200	100	0,045
350	100	0,11
500	100	0,115

Аналіз роботи алгоритму Fish School Search продемонстрували, що результативність алгоритму залежна від визначених параметрів, чим краще параметри підібрані, тим точнішим буде результат та меншим час виконання алгоритму.

3.4 Проведення аналізу результативності виконання генетичного алгоритму

Значення масштабу формації під час виконання генетичного алгоритму. Аналіз значення масштабів формації під час роботи алгоритму проводилася на базі пошуку екстремуму функції $y = 0{,}1(x^2 + z^2) + 10$, найбільш результативні значення продемонстровані у таблиці 3.4. З цієї таблиці можна побачити, що оптимальний масштаб формації становить 100 особин.

Задля аналізу залежності швидкості пошуку рішень від значення вірогідності симбіозу хромосом була визначена функція $y = 3x^2 + xz + 2z^2 - x - 4z$.

Таблиця 3.4

Значення масштабів формації під час виконання алгоритму

Масштаби формації	Час виконання алгоритму, (с)
5	0,17
10	0,091
25	0,032
35	0,032
50	0,026
75	0,025
100	0,023
200	0,026
500	0,037
1000	0,077

Таблиця 3.5

Значення вірогідності симбіозу хромосом під час виконання алгоритму

Вірогідність симбіозу, (%)	Час виконання алгоритму, (с)
95	0,051
90	0,061
85	0,046
80	0,046
75	0,053
70	0,09
65	0,098
60	0,165

Таблиця 3.5 демонструє, що оптимальна вірогідність симбіозу хромосом для визначеної функції - складає 80-85%. Таблиця 3.6. демонструє значення чисельності ітеративних обробок на точність та час виконання алгоритму для функції $y = 0{,}1(x^2 + z^2) + 10$.

Таблиця 3.6

Значення чисельності ітеративних обробок на точність та час виконання алгоритму

Чисельність ітеративних обробок	Точність виконання алгоритму, (%)	Час виконання алгоритму, (с)
15	99,9998	0,025
25	100	0,03
35	100	0,054
50	100	0,073
75	100	0,094
100	100	0,121
200	100	0,267
350	100	0,399
500	100	0,584

Таблиця 3.6. демонструє точність виконання алгоритму і час збільшується з чисельністю ітеративних обробок.

Аналіз роботи генетичного алгоритму демонструють, що результативність алгоритму залежить від заданих параметрів, чим точніше параметри визначені, тим точнішим є результат та меншим час виконання.

3.5 Зіставлення параметрів результативності роботи алгоритмів FSS і GA

Задля зіставлення параметрів результативності роботи алгоритму пошуку FSS у рамках виконання задачі глобальної оптимізації було визначено генетичний алгоритм. Таблиця 3.7 демонструє результати обчислень значень оптимуму функції Де Джонга $f(x) = \sum_{i=1}^{n} \quad x_i^2, i = 1{:}\,n$, з різною кількістю змінних

Таблиця 3.7

Точність отриманого результату визначеними алгоритмами для функції Де Джонга

Чисельність змінних Точність, (%)	3	4	5
FSS	99,988	99,99	99,993
GA	99,999	100	100

Беручи в увагу дані таблиці 3.7 не важко дійти до висновку: алгоритми – досить точні у своїх результатах, тому що точність FSS алгоритму складає майже 100%, а у GA - 100%. Таблиця 3.8 та 3.9 демонструє відсутність автономії кількості ітеративних обробок, потрібних задля пошуку оптимуму функцій Розенброка $f(x) = \sum_{i=1}^{n-1} \quad 100\left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2, i = 1{:}\,n-1$ та Растрігіна $f(x) = 10n + \sum_{i=1}^{n} \quad (x_i^2 - 10\ cos\ cos\ (2\pi x_i)\ ), i = 1{:}\,n.$

Таблиця 3.8

Чисельність ітеративних обробок задля функції Розенброка з іншою чисельністю змінних

Чисельність змінних Чисельність ітерацій	3	4	5
FSS	170	250	330
GA	130	370	650

Таблиця 3.9

Чисельність ітеративних обробок задля пошуку оптимуму функції Растрігіна з різноманітною чисельністю змінних

Чисельність змінних Чисельність ітерацій	3	4	5
FSS	150	170	200
GA	210	390	735

Згідно таблиць 3.8 і 3.9 можемо помітити, що алгоритму FSS потрібно менша кількість ітеративних обробок при пошуку екстремуму багатомірної функції, розмірність більше 3-х змінних.

Таблиця 3.10 демонструє відсутність автономії часу виконання, потрібного задля пошуку оптимуму функції Швефеля $f(x) = \sum_{i=1}^{n} \quad -x_i \; sin \; sin \; (\sqrt{|x_i|}) \, , i = 1 : n.$

Таблиця 3.10

Час потрібний задля пошуку оптимуму функції Растрігіна з різною чисельністю змінних

Чисельність змінних Час роботи, (с)	3	4	5
FSS	0,18	0,29	0,404
GA	0,155	0,37	0,64

Таблиця 3.10 демонструє те, що алгоритм Fish School Search - швидший, в порівняні з генетичним алгоритмом.

При досліджені ЕА задля пошуку оптимуму функцій можна побачити, що ці два алгоритми - універсальні методи задля пошуку екстремуму, незважаючи на складності функцій. Задля знаходження оптимуму простих функцій найкращим чином буде застосовувати класичні методи, вони, у цій ситуації, виконують свою

роботу швидше ніж ЕА. Під час вирішення питання глобальної оптимізації для функції зі складним ландшафтом доцільніше використовувати алгоритм FSS, через те, що звичайні алгоритми з високою частотою спричинює некоректний результат, а GA працює дещо повільніше. За допомогою модифікації алгоритму Fish School Search, алгоритм сходиться на функціях з гладкою поверхнею. Алгоритму Fish School Search потрібна менша чисельність ітеративних обробок задля функцій багатьох змінних, на функціях з невеликою чисельністю змінних генетичний алгоритм має перевагу у чисельності ітеративних обробок. Коректність розрахунків алгоритмів - майже 100%. Слід уточнити, під час роботи алгоритму і точність пошуку рішення напряму впливають задані дані, необхідно намагатись задавати параметри якомога оптимальніші, звертаючи увагу на проведені дослідження.

3.6 Користувацький інтерфейс розробленої системи

На рисунку 3.1 продемонстрований створений користувацький інтерфейс системи задля обробки алгоритму Fish School Search. Потрібно задати дані параметри алгоритму:

- Функція аналізу – етап пошуку екстремума.
- Верхній максимум – верхня границя діапазону пошуку.
- Нижній мінімум – нижня границя діапазону пошуку.
- Кількість ітеративних обробок
- Розмір формації – перелік одиниць, котрі будуть приймати участь у пошуку вирішення задачі.
- Стартовий індивідуальний етап – етап, який має на увазі рух операторів (риб), стартуючи з 1-ї ітераційної обробки.
- Фінальний індивідуальний етап - який має на увазі рух операторів (риб), на фінальних етапах ітераційної обробки.
- Max вага одиниці.

На рисунку 3.2 продемонстровано користувацький інтерфейс генетичного алгоритму. У полях системи потрібно надати такі дані алгоритму:

- Досліджувана функція – функція пошуку екстремума.
- Верхня границя – верхнє обмеження області пошуку рішення задачі.
- Нижня границя – нижнє обмеження області пошуку рішення задачі.
- Чисельність ітеративних обробок
- Масштаби формації – чисельність операторів, котрі будуть займатися пошуком рішення.
- Вірогідність симбіозу – вірогідність, з якою проходить процес симбіозу хромосом (за замовчуванням - 90%).
- Вірогідність мутації – вірогідність того, що пройде процес мутації хромосом (за замовчуванням - 10%).
- Чисельність хромосом – чисельність змінних функції.

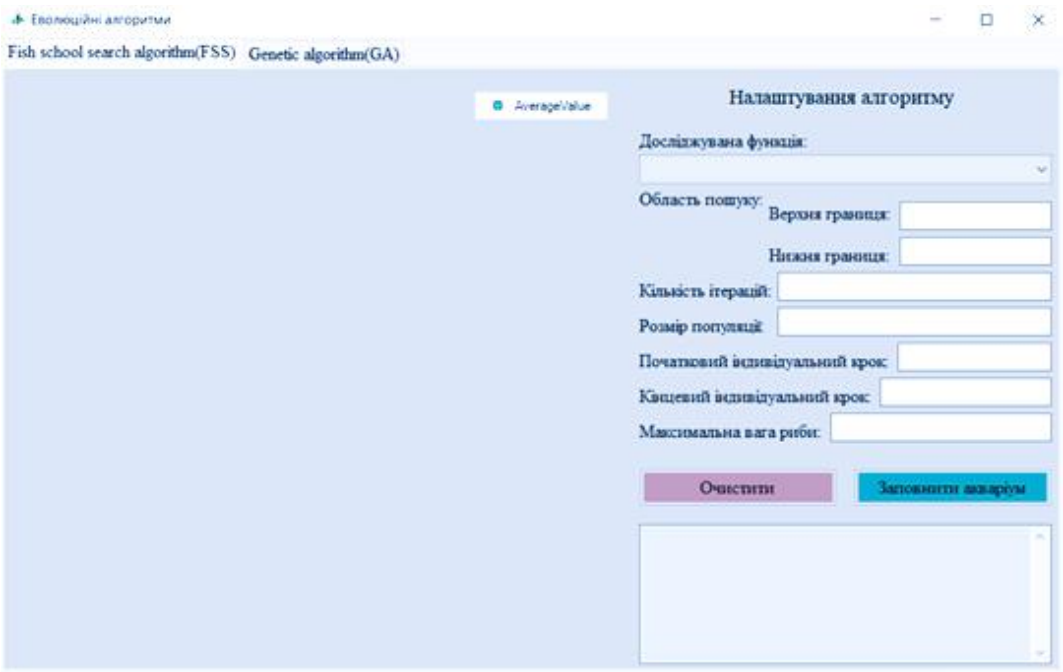


Рис. 3.1 Користувацький інтерфейс системи для алгоритму FSS

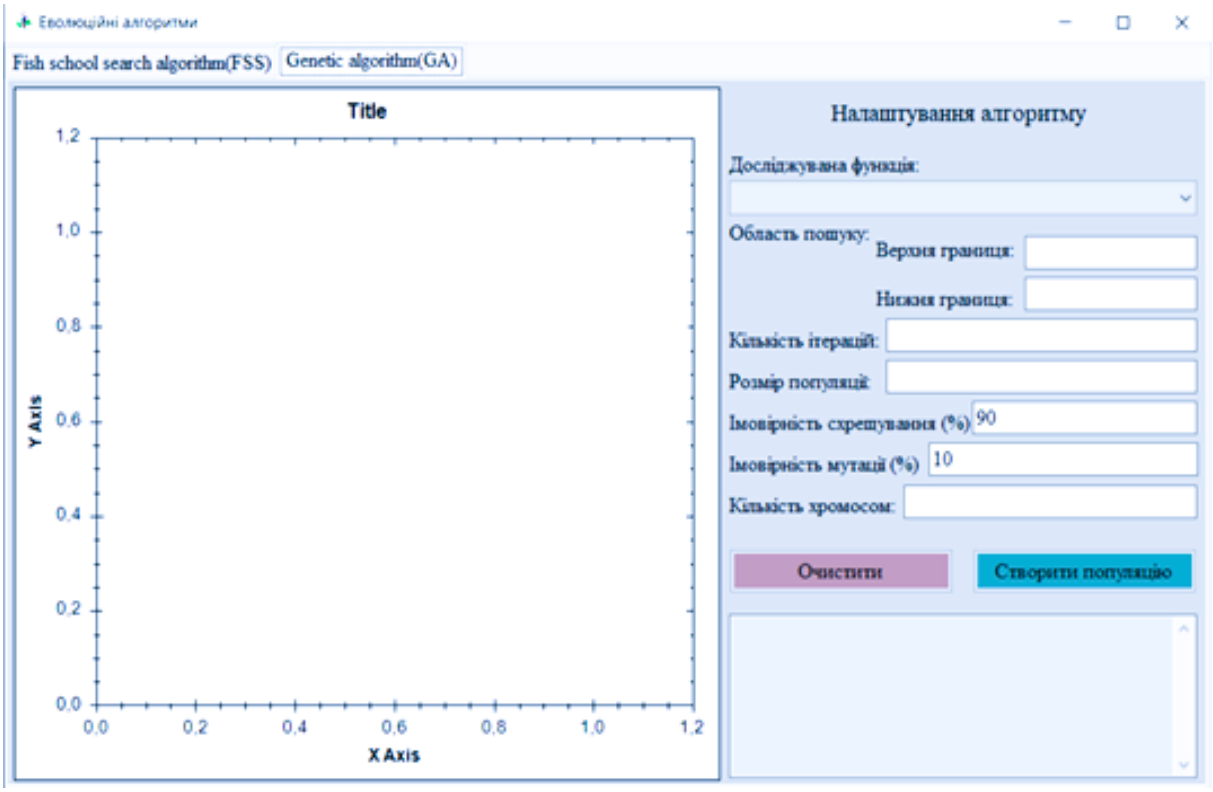


Рис. 3.2 Користувацький інтерфейс системи для алгоритму GA

Дані, які потрібно ввести юзеру повинні пройти етап валідації на коректність. За умови, що один із параметрів не надано, або цей параметр буде надано некоректно – з’явиться поле з попередженням «Перевірте чи введено усі дані!» або «Перевірте чи правильно надано усі дані!», дане поле буде виділене червоним кольором.

Рис. 3.3 Задані некоректні дані

Коли усі дані введено коректно – система починає обробку. Результатом буде отриманий мінімум функції, час оброблення алгоритму та графік значень мінімумів на кожній ітеративній обробці.

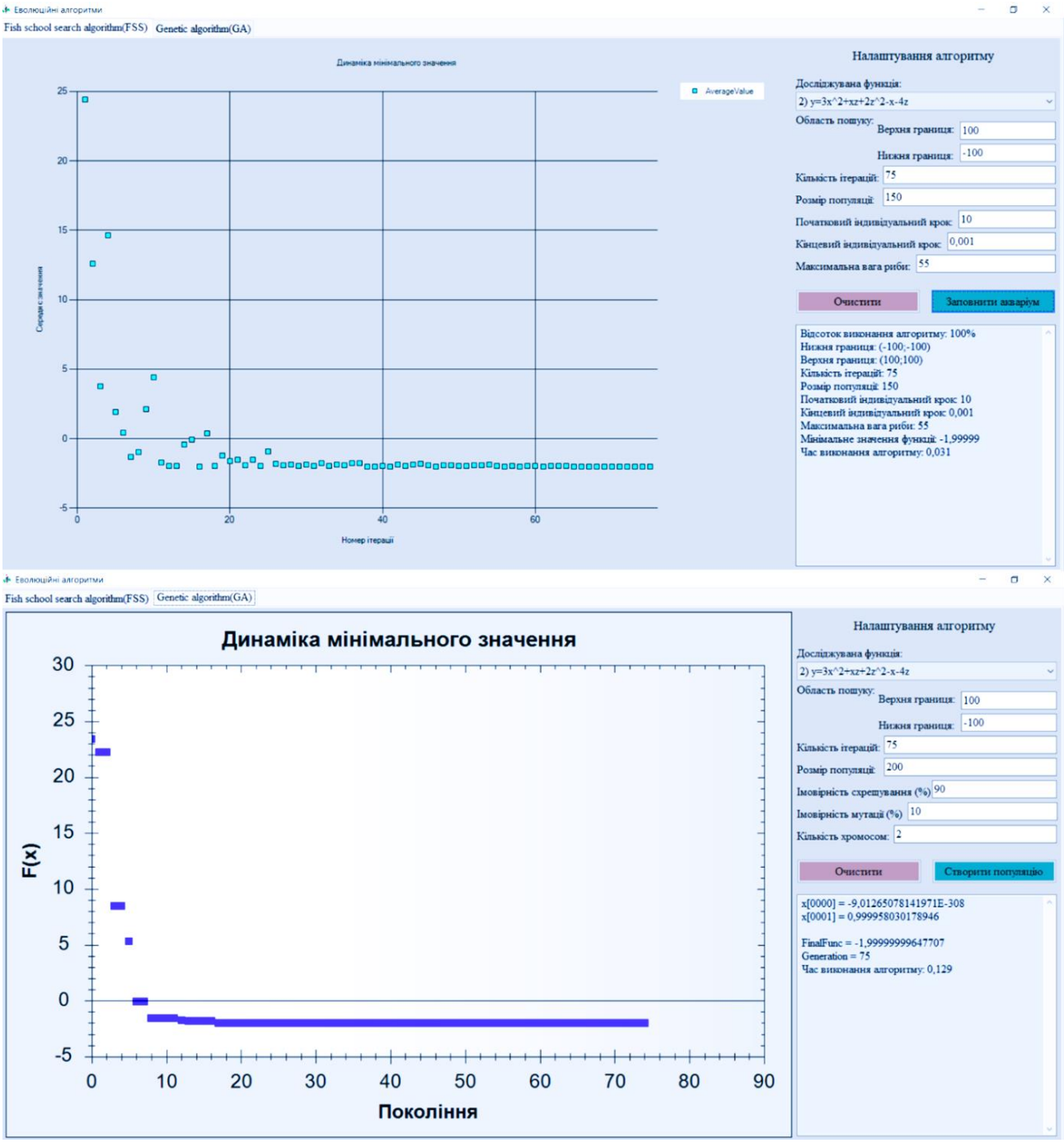


Рис. 3.4 Результати виконання задачі у розробленій системі

3.7 Аналіз БД

База даних *ResearchOfEvolutionaryAlgorithms* складається з 2-х таблиць:

- *GA_parameters* – набір даних у вигляді таблиці, яка містить дані оброблені користувачем і результат виконання генетичного алгоритму та час його оброблення.

- *Fish School Search* – набір даних у вигляді таблиці, яка містить дані введені юзером та результат роботи алгоритму.

Таблиці демонструють дані поля, визначені на рисунку 3.5

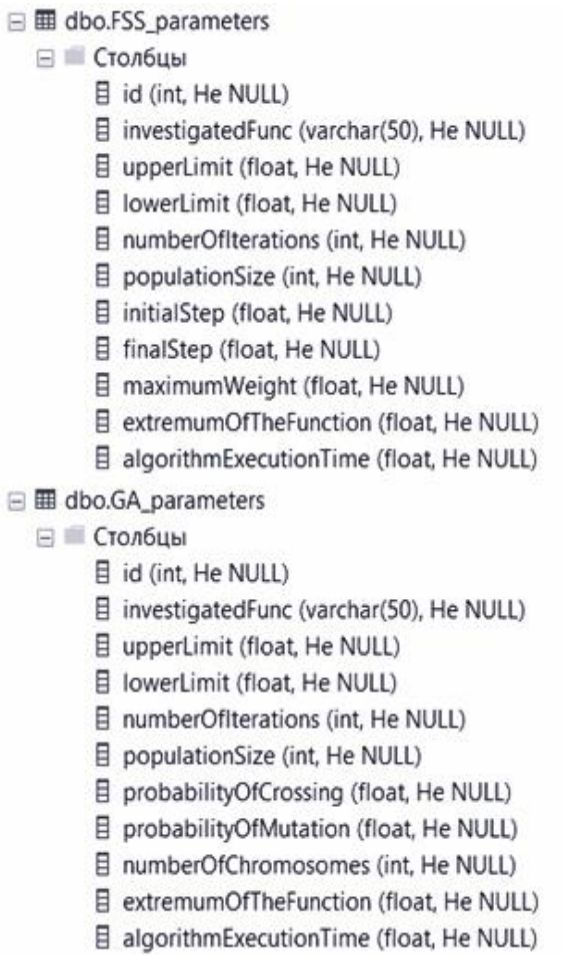


Рис. 3.5 Таблиці бази даних

Таблиця *GA_parameters* після процесу старту роботи системи продемонстрована на рисунку 3.6

Результаты Сообщения

	id	investigatedFunc	upperLimit	lowerLimit	numberOfIterations	populationSize	probabilityOfCrossing	probabilityOfMutation	numberOfChromosomes	extremumOfTheFunction	algorithmExecutionTime
1	823	0) y=-x(10-x)	10	-10	100	200	0,9	0,1	1	-25	0,168999999761581
2	824	0) y=-x(10-x)	10	-10	44	200	0,9	0,1	1	-24,999927520752	0,0860000029206276
3	825	2) y=3x^2+xz+2z^2-x-4z	100	-100	44	200	0,9	0,1	2	-1,99999988079071	0,0900000035762787
4	826	4) y=10(sin(0,1a)+sin(0,1b)+sin(0,1c))	100	-100	44	200	0,8	0,2	3	-29,9958629608154	0,0850000008940697
5	827	4) y=10(sin(0,1a)+sin(0,1b)+sin(0,1c))	100	-100	440	200	0,8	0,2	3	-29,9999942779541	0,778999984264374
6	828	5) y=0,1(a^2+b^2+c^2+d^2+ad+bd+bc)+100	100	-100	440	200	0,8	0,2	4	100,003913879395	0,828999996185303
7	829	5) y=0,1(a^2+b^2+c^2+d^2+ad+bd+bc)+100	107	-109	1000	200	0,8	0,2	4	100,000465393066	0,805999994277954
8	830	5) y=0,1(a^2+b^2+c^2+d^2+ad+bd+bc)+100	10	-10	10	500	0,8	0,2	4	100	0,0410000011324883
9	817	15) Функція Шевефеля (3 змінні)	500	-500	100	100	0,9	0,1	3	-1260,04931640625	0,180000007152557
10	818	15) Функція Шевефеля (3 змінні)	500	-500	500	400	0,9	0,1	3	-1262,193359375	0,398000001907349
11	819	16) Функція Шевефеля (4 змінні)	500	-500	500	400	0,9	0,1	4	-1682,94079589844	0,509999990463257
12	820	16) Функція Шевефеля (4 змінні)	500	-500	1000	400	0,9	0,1	4	-1682,88596632813	0,941999971866608
13	821	17) Функція Шевефеля (5 змінних)	500	-500	1000	400	0,9	0,1	5	-2103,67651367188	0,129999995231628
14	822	17) Функція Шевефеля (5 змінних)	500	-500	400	500	0,7	0,3	5	-2103,671875	0,458999991416931

Рис. 3.6 Таблиця *GA_parameters*

Таблиця *FSS_parameters* після процесу старту роботи системи продемонстрована на рисунку 3.7.

Результаты | Сообщения

	id	investigatedFunc	upperLimit	lowerLimit	numberOfIterations	populationSize	initialStep	finalStep	maximumWeight	extremumOfTheFunction	algorithmExecutionTime
1	486	0) y=-x(10-x)	100	-100	100	100	10	1	50	-24,9999980926514	0,046000000089407
2	487	0) y=-x(10-x)	100	-100	100	100	10	0,001	50	-24,9999904632568	0,0410000011324883
3	488	1) y=0,1(x^2+z^2)+10	100	-100	100	100	10	0,001	50	10,0000066757202	0,0359999984502792
4	489	2) y=3x^2+xz+2z^2-x-4z	1000	-1000	100	1000	10	0,001	50	2,61325216293335	0,250999987125397
5	490	2) y=3x^2+xz+2z^2-x-4z	100	-100	100	100	10	0,01	50	-1,99974954128265	0,0370000004768372
6	491	2) y=3x^2+xz+2z^2-x-4z	100	-100	100	100	10	0,01	50	-1,99998688697815	0,0410000011324883
7	492	3) y=x^2-xz+z^2+3x-2z+1	100	-100	100	100	10	0,01	50	-1,33333218097687	0,0350000001490116
8	493	5) y=0,1(a^2+b^2+c^2+d^2+ad+bd+bc)+100	100	-100	100	100	10	0,01	50	100,000183105469	0,0480000004172325
9	494	8) Функція Де Джонга (5 змінних)	10	-10	100	100	1	0,01	50	7,08668085280806E-05	0,063000001013279
10	495	9) Функція Розенброка (3 змінні)	10	-10	100	100	1	0,01	50	0,459905475378037	0,0560000017285347
11	496	9) Функція Розенброка (3 змінні)	10	-10	400	100	1	0,01	50	0,106386631727219	0,193000003695488
12	497	9) Функція Розенброка (3 змінні)	10	-10	400	100	1	0,001	50	0,0854343846440315	0,188999995589256
13	498	8) Функція Де Джонга (5 змінних)	10	-10	400	100	1	0,001	50	9,68388349065208E-07	0,150999993085861
14	499	4) y=10(sin(0,1a)+sin(0,1b)+sin(0,1c))	10	-10	400	100	1	0,001	50	-24,6504001617432	0,155000001192093
15	500	0) y=-x(10-x)	10	-10	400	100	1	0,001	50	-24,9999332427979	0,0750000029802322

Рис. 3.7 Таблиця *FSS_parameters*

3.9 Аналіз якості системи

Задля перевірки працездатності визначених алгоритмів Fish School Search і генетичного алгоритму, були реалізовані тести на базі MSTest. Виконання алгоритмів досліджували на різноманітних багатомірних функціях, із відмінними вхідними даними.

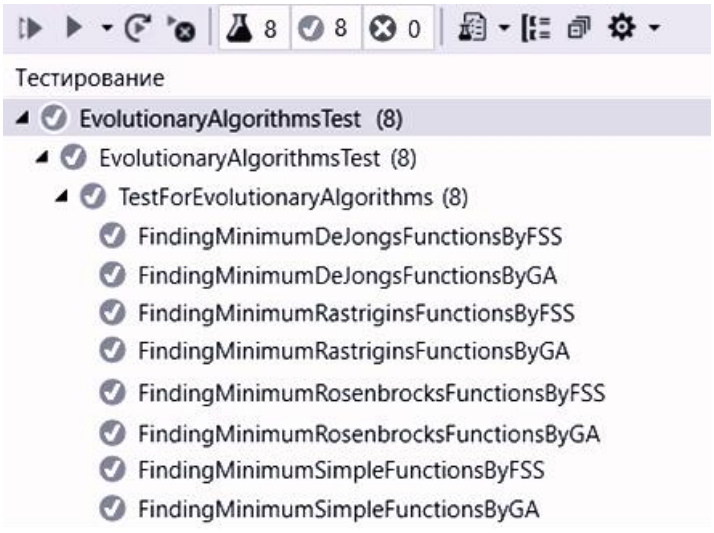


Рис. 3.8 Тестування працездатності алгоритмів

На рисунку 3.8 продемонстрована робота алгоритмів FSS і GA, дані алгоритми виконуються відповідно коректно з функціями з базовою складовою та функціях зі складним ландшафтом.

3.8 Ідеї, щодо майбутнього розвитку системи та її покращення

Розглядаючи майбутнє розвитку та покращення створеної системи, можна окреслити кілька перспективних напрямків. Перш за все, варто зосередитися на розширенні набору алгоритмів оптимізації, додавши інші еволюційні алгоритми, такі як диференціальна еволюція чи алгоритм бджолиного рою, а також імплементувати класичні методи оптимізації для порівняння з еволюційними. Важливим аспектом розвитку системи є вдосконалення користувацького інтерфейсу. Це може включати додавання можливості візуалізації процесу оптимізації в реальному часі, реалізацію інтерактивного налаштування параметрів алгоритмів під час виконання та створення модуля для порівняння результатів різних алгоритмів.

Розширення функціональності системи також є ключовим напрямком розвитку. Можна додати можливість оптимізації функцій з обмеженнями, реалізувати багатокритеріальну оптимізацію та впровадити механізми паралельних обчислень для прискорення роботи алгоритмів. Вдосконалення роботи з даними, включаючи експорт результатів у різні формати та можливість завантаження користувацьких функцій з файлу, також значно покращить користувацький досвід. Покращення аналітичних можливостей системи може бути досягнуто шляхом розробки модуля для статистичного аналізу результатів оптимізації та додавання інструментів для оцінки ефективності алгоритмів на різних типах функцій. Розширення бази даних для збереження повної історії оптимізації та порівняння результатів різних запусків алгоритмів також сприятиме глибшому аналізу та розумінню процесів оптимізації.

Вдосконалення самих алгоритмів, зокрема реалізація адаптивного налаштування параметрів та впровадження гібридних версій, може значно підвищити ефективність системи. Створення веб-версії системи з можливістю віддаленого запуску оптимізаційних задач розширить доступність та зручність використання.

Інтеграція з іншими системами через розробку API та можливість використання системи як бібліотеки в інших проектах відкриє нові можливості для застосування. Нарешті, покращення документації та створення навчальних матеріалів допоможе користувачам ефективніше використовувати систему та розуміти її можливості.

Усі ці вдосконалення в комплексі зроблять систему більш функціональною, гнучкою та зручною для користувачів, розширять її можливості для вирішення різноманітних оптимізаційних задач та зміцнять її позиції як потужного інструменту в галузі еволюційних алгоритмів та оптимізації.

ВИСНОВОК ДО РОЗДІЛУ 3

Даний розділ дипломної роботи присвячений дослідженню та розробці системи для пошуку екстремумів багатовимірних функцій з використанням еволюційних алгоритмів, зокрема генетичного алгоритму (GA) та алгоритму Fish School Search (FSS). Було проведено ґрунтовний аналіз теоретичних основ цих алгоритмів та розробив програмну систему для їх реалізації та порівняння.

У роботі детально описано структуру та основні компоненти розробленої системи, включаючи класи для реалізації алгоритмів, роботи з базою даних та користувацький інтерфейс. Особлива увага приділена модифікації алгоритму FSS для підвищення його ефективності. Також було проведено серію експериментів для оцінки ефективності розроблених алгоритмів на різних тестових функціях, таких як функції Де Джонга, Розенброка, Растрігіна та Швефеля. Результати показали, що обидва алгоритми демонструють високу точність (близько 100%) при пошуку оптимуму, але FSS виявився більш ефективним для функцій з більшою кількістю змінних, потребуючи менше ітерацій для досягнення оптимуму.

Важливою частиною роботи є аналіз впливу різних параметрів алгоритмів на їх ефективність. Це дозволило визначити оптимальні налаштування для кожного алгоритму, що є цінним внеском для практичного застосування цих методів.

Розроблена система має зручний користувацький інтерфейс, який дозволяє легко задавати параметри алгоритмів та візуалізувати результати. Крім того, система інтегрована з базою даних для зберігання результатів експериментів, що полегшує подальший аналіз та порівняння.

Загалом, ця дипломна робота представляє собою комплексне дослідження еволюційних алгоритмів оптимізації з практичною реалізацією у вигляді програмної системи.

ВИСНОВКИ

У рамках даної дипломної роботи було проведено комплексне дослідження еволюційних алгоритмів глобальної пошукової оптимізації, зокрема генетичного алгоритму (GA) та модифікованого алгоритму пошуку косяка риб (FSS). Робота охоплює теоретичні основи оптимізації, детальний аналіз різних еволюційних алгоритмів, їх практичну реалізацію та експериментальну оцінку ефективності. У ході дослідження було розроблено програмну систему на мові C# з використанням .NET Framework, яка дозволяє ефективно застосовувати та порівнювати алгоритми GA та FSS для вирішення складних оптимізаційних задач. Експериментальні результати на різних тестових функціях показали високу точність обох алгоритмів, при цьому FSS продемонстрував кращу ефективність для функцій з більшою кількістю змінних. Важливим аспектом роботи стало дослідження впливу параметрів алгоритмів на їх ефективність, що дозволило визначити оптимальні налаштування для різних типів задач. Розроблена система має зручний користувацький інтерфейс та інтеграцію з базою даних, що робить її потужним інструментом для подальших досліджень та практичного застосування в різних галузях, таких як інженерія, економіка та машинне навчання. Таким чином, дана робота не лише розширює теоретичне розуміння еволюційних алгоритмів, але й надає практичний інструментарій для їх ефективного використання у вирішенні складних оптимізаційних задач реального світу.

Результати цього дослідження мають значний потенціал для практичного застосування в різноманітних сферах, де виникають складні оптимізаційні задачі. Розроблена система може бути адаптована для вирішення реальних проблем в інженерії, фінансах, логістиці та інших галузях, де потрібно знаходити оптимальні рішення в умовах багатовимірності та нелінійності. Особливо цінним є порівняльний аналіз алгоритмів GA та FSS, який надає користувачам можливість вибору найбільш підходящого методу залежно від специфіки конкретної задачі.

Важливо відзначити, що робота не обмежується лише реалізацією відомих алгоритмів, але й включає їх модифікацію та удосконалення. Зокрема,

модифікований алгоритм FSS показав підвищену ефективність у порівнянні з класичним варіантом, що відкриває перспективи для подальших досліджень у напрямку оптимізації еволюційних алгоритмів.

Крім того, розроблена програмна система з її гнучкою архітектурою та інтуїтивно зрозумілим інтерфейсом може служити основою для подальших досліджень у галузі еволюційних обчислень. Вона надає можливість легко інтегрувати нові алгоритми, проводити порівняльні експерименти та візуалізувати результати, що робить її цінним інструментом як для науковців, так і для практиків у галузі оптимізації.

У підсумку, дана дипломна робота не тільки вносить вклад у теоретичне розуміння еволюційних алгоритмів, але й надає практичний інструментарій для їх ефективного застосування. Вона закладає основу для подальших досліджень та інновацій у галузі глобальної пошукової оптимізації, відкриваючи нові можливості для вирішення складних задач у різних сферах людської діяльності.

ПЕРЕЛІК ПОСИЛАНЬ

1. Floudas C. A., Pardalos P. M. State of the Art in Global Optimization: Computational Methods and Applications. New York City : Springer; 1996th edition, 2011. 664 p.
2. Christodoulos A. F. State of the Art in Global Optimization: Computational Methods and Applications. Dordrecht, Netherlands : Springer, 2011. 664 p.
3. An Algorithm for the Traveling Salesman Problem / J. D. C. Little et al. 11th ed. FB &c Ltd, Dalton House, 60 Windsor Avenue, London, SW19 2RR : Forgotten Books, 2018. 83 p.
4. Баженов Є. В. Оптимізаційні алгоритми та їх застосування. Київ : Київ: Техніка, 2021. 320 с.
5. Goldberg D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. Boston : Addison-Wesley Professional, 1989. 412 p.
6. Mehrabian A. R., Lucas C. A novel numerical optimization algorithm inspired from weed colonization. Ecological informatics. 2006. Vol. 1, no. 4. P. 355–366. URL: https://doi.org/10.1016/j.ecoinf.2006.07.003 (date of access: 13.11.2024).
7. Robbins H., Monro S. A Stochastic Approximation Method. 22nd ed. Virginia, USA.: Institute of Mathematical Statistics, 1951. 407 p.
8. Mallawaarachchi V. Introduction to Genetic Algorithms – Including Example Code. URL: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3.
9. Yang X.-S. Firefly algorithm, stochastic test functions and design optimisation. International journal of bio-inspired computation. 2010. Vol. 2, no. 2. P. 78–84. URL: https://doi.org/10.48550/arXiv.1003.1409.
10. Yang X.-S. Cuckoo search via Levy flights. in: Proc. of World Congress on Nature & Biologically Inspired Computing. 2009. Vol. 1, no. 1. P. 210-214. URL: https://doi.org/10.48550/arXiv.1003.1594.
11. The Expanded Invasive Weed Optimization Metaheuristic for Solving Continuous and Discrete Optimization Problems / H. Josiński et al. The Scientific

World Journal. 2014. Vol. 2014. P. 1–14. URL: https://doi.org/10.1155/2014/831691 (date of access: 13.11.2024).

12. Ibe O. C. Markov Processes for Stochastic Modeling. 2nd ed. Amsterdam, Netherlands : Elsevier, 2013. 424 p.
13. Deb K. Optimization for Engineering Design: Algorithms and Examples. 2nd ed. New Delhi : PHI, 2012. 396 p.
14. Mucherino A., Seref O. Monkey Search: A Novel Metaheuristic Search for Global Optimization. Gainesville : AIP Conference on Data Mining, Systems Analysis and Optimization in Biomedicine, 2007. 173 p.
15. Tuba M., Subotic M., Stanarevic N. Modified cuckoo search algorithm for unconstrained optimization problems. Proc. of the 5th European Computing Conference (ECC'I1), Paris, France. 2011. Vol. 1. P. 263–268.
16. Yang X.-S. A New Metaheuristic Bat-Inspired Algorithm. arXiv.org. URL: https://doi.org/10.48550/arXiv.1004.4170.
17. Zhao R. Monkey algorithm for global numerical optimization. Journal of Uncertain Systems. 2008. Vol. 3. P. 165–176.
18. Bastos-Filho C., Nascimento D. An Enhanced Fish School Search Algorithm. IEEE : International Joint Conference on Neural Networks, Ipojuca, 8–11 September 2013. Brazil, 2013.
19. Evolutionary Algorithms Enhanced with Quadratic Coding and Sensing Search for Global Optimization / A.-R. Hedar et al. Mathematical and Computational Applications. 2020. Vol. 25, no. 1. P. 7. URL: https://doi.org/10.3390/mca25010007.
20. Dynamic motion based evolutionary algorithm for enhancement of the search capability for global search space / N. Parashar et al. International Journal of System Assurance Engineering and Management. 2024. URL: https://doi.org/10.1007/s13198-024-02556-9.
21. A Hybrid Algorithm Based on Fish School Search and Particle Swarm Optimization for Dynamic Problems / G. M. Cavaicanti-J'unior et al. 2nd ed. Internauional Conference on Swarm Intelligence 2011 (ICSI), 2011. 552 p.

22. Microsoft. C# documentation. Microsoft. URL: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.
23. PYPL PopularitY of Programming Language. GitHub PYPL. URL: https://pypl.github.io/PYPL.html.
24. TIOBE Index for November 2024. TIOBE. URL: https://www.tiobe.com/tiobe-index/.
25. Desktop Guide (Windows Forms .NET). Microsoft. URL: https://learn.microsoft.com/enus/dotnet/desktop/winforms/overview/?view=net desktop-9.0&viewFallbackFrom=netdesktop-5.0.
26. C Sharp – Features, Advantages and Disadvantages. URL: https://urbannaturale.com/c-sharp-features-advantages-and-disadvantages/.
27. What is .NET Framework?. Microsoft. URL: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet-framework.
28. Getting started with .NET Charts. I Programmer. URL: https://www.i-programmer.info/programming/uiux/2756-getting-started-with-net-charts.html.
29. A flexible charting library for .NET. CodeProject. URL: https://www.codeproject.com/Articles/5431/A-flexible-charting-library-for-NET.
30. .NET regular expressions. Microsoft. URL: https://learn.microsoft.com/en-us/dotnet/standard/base-types/regular-expressions.
31. Hughes A. Microsoft SQL Server. TechTarget. URL: https://www.techtarget.com/searchdatamanagement/definition/SQL-Server.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО - КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

СИСТЕМА ГЛОБАЛЬНОЇ ПОШУКОВОЇ ОПТИМІЗАЦІЇ ПАРАМЕТРІВ ТЕХНОЛОГІЧНИХ ПРОЦЕСІВ ДЛЯ АНАЛІЗУ ЕФЕКТИВНОСТІ МОДИФІКОВАНИХ ЕВОЛЮЦІЙНИХ АЛГОРИТМІВ

Виконав: студент групи ПДМ-62 Бриль Владислав Володимирович

Керівник: доцент кафедри ІПЗ Задонцев Юрій Вікторович

Київ - 2025

МЕТА, ОБ'ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи оптимізація алгоритмів, які ефективно розв'язують задачі з найменшими витратами часу і ресурсів.

Об'єкт дослідження еволюційні алгоритми.

Предмет дослідження аналіз модифікованих еволюційних алгоритмів, їх складності в залежності від умов задач та середовища

ПОРІВНЯННЯ ХАРАКТЕРИСТИК ІСНУЮЧИХ АЛГОРИТМІВ ІЗ МОДИФІКОВАНИМ

Метод/Алгоритм	Переваги	Недоліки	Причини обмеженості для поставленої задачі
Алгоритм Firefly(FFA)	- Підходить для мультимодальних задач.	- Чутливість до коефіцієнта атракції.	- Не здатен гарантувати ефективну оптимізацію в умовах складної поверхні.
Fish School Search(FSS)	- Висока точність на кінцевих ітераціях. - Інтуїтивність у роботі з групами агентів.	- Потребує великої кількості агентів для точності.	- Вимагає великих обчислювальних ресурсів при високій розмірності задачі.
Метод градієнтного спуску	- Швидкість для гладких задач.	- Застрягає в локальних оптимумах.	- Не підходить для задач з мультимодальним ландшафтом.
Модифікований Fish School Search алгоритм (FSS)	**- Підрахунок ймовірності погіршення норми для кожного агента.**	**- Складність налаштування параметрів.**	
Генетичний алгоритм (GA)	**- Ефективний для глобального пошуку.**	**- Чутливість до налаштувань параметрів (розмір популяції, ймовірності мутацій та**	**- Потребує багато ітерацій для досягнення прийнятного результату.**

УМОВНІ ПОЗНАЧЕННЯ

Наступним чином, задля подальшого спрощення та розуміння термінів, пропоную ввести загальновідомі скорочення термінів:

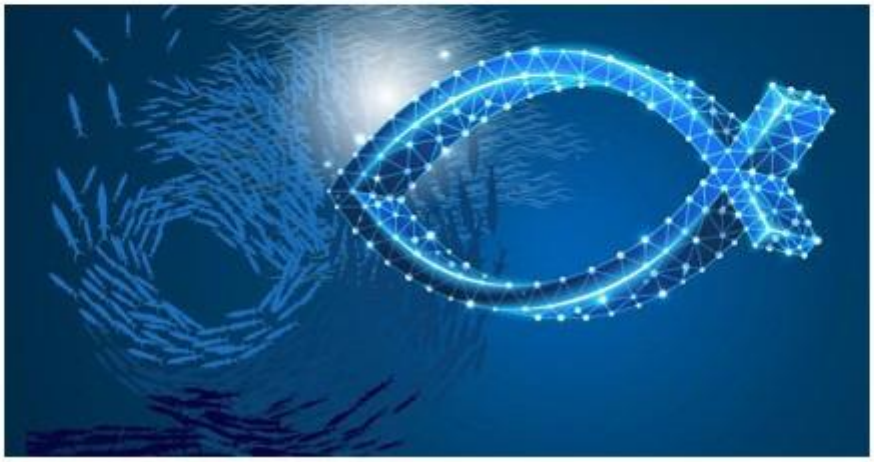
FSS – Fish School Search .

EA – Еволюційні алгоритми.

GA – Генетичний алгоритм.

ПРАКТИЧНИЙ РЕЗУЛЬТАТ

FSS - алгоритм пошуку на базі формації, діях риб, котрі розмножуються та зменшують популяцію у пошуках здобичі. Локація риби - вірогідне вирішення проблематики оптимізації. Алгоритм застосовує вагу для усіх операторів, які демонструють результати, наскільки успішним був пошук риби у зграї.



5

ПРАКТИЧНИЙ РЕЗУЛЬТАТ

Риба в зграї відіграє головну роль у локальному пошуку в пошуках перспективних областей. За формулою :

$$x_i(t+1) = x_i(t) + rand(-1,1)step_{ind},$$

де $x_i(t)$ та $x_i(t+1)$ зображають стан одиниці i до та після окремого оператора руху;

$rand(-1,1)$ - систематично поділене рандомне число, в діапазоні від -1 до 1;

$step_{ind}$ - характеристика, що позначає max переміщення для даного руху

Нова позиція $x_i(t+1)$ узгоджується лише за умови, коли фізична спроможність оператора поліпшується у зв'язку з переміщенням.

За умови недотримання вимог, оператор залишається у сталому положенні $x_i(t+1) = x_i(t)$.

6

ПРАКТИЧНИЙ РЕЗУЛЬТАТ РОБОТИ ПРОГРАМИ

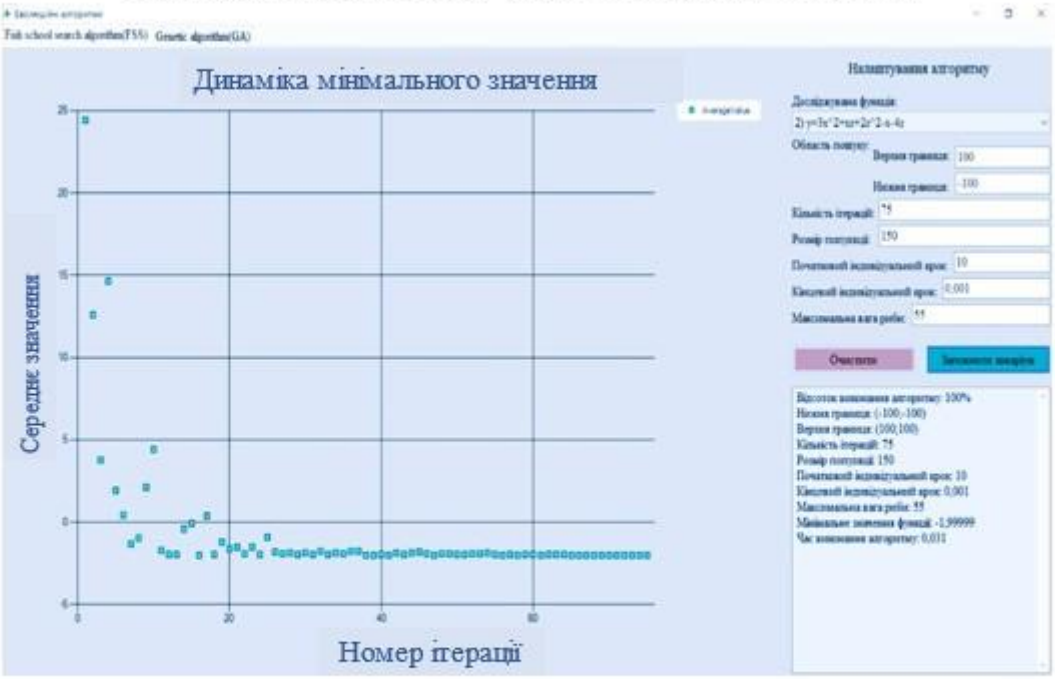


Рис. 1 Графік значень мінімумів функції на кожній ітерації

ПРАКТИЧНИЙ РЕЗУЛЬТАТ РОБОТИ ПРОГРАМИ

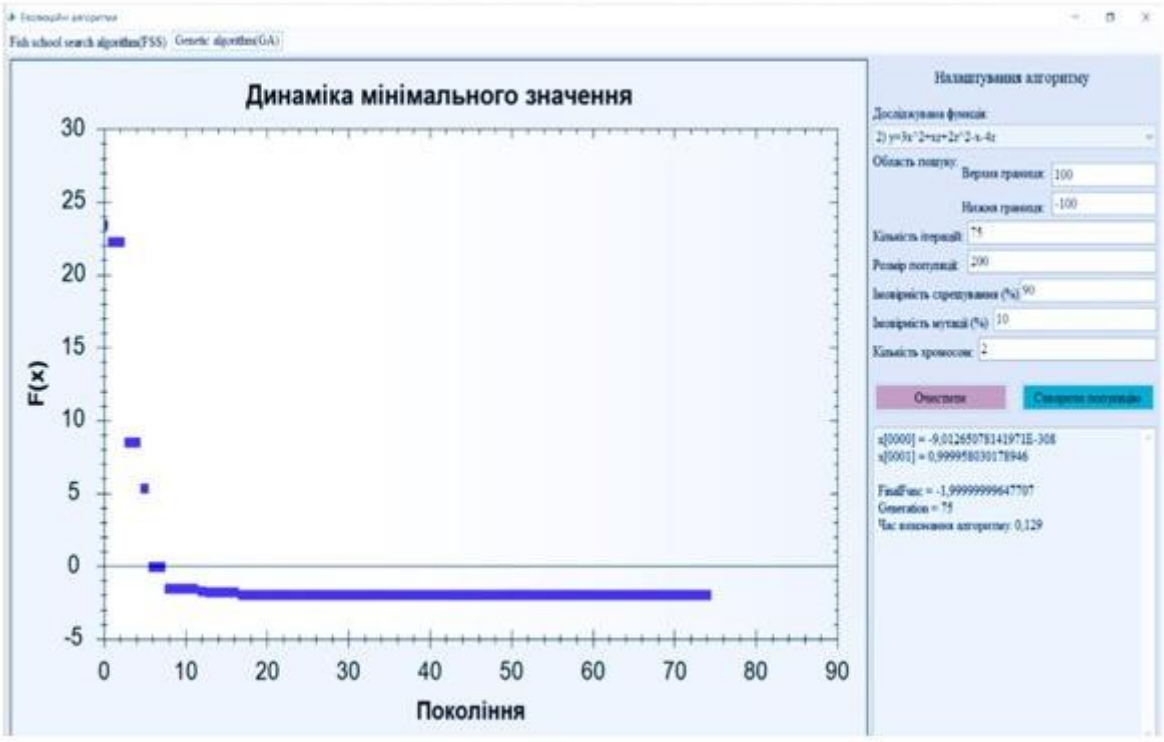


Рис. 2 Графік значень мінімумів функції у поколіннях

РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ МАСШТАБІВ ФОРМАЦІЇ

Аналіз ефекту масштабів формації під час виведення результату алгоритму будувалося на базі пошуку екстремуму функції :

$$y = 0{,}1(x^2 + z^2) + 10$$

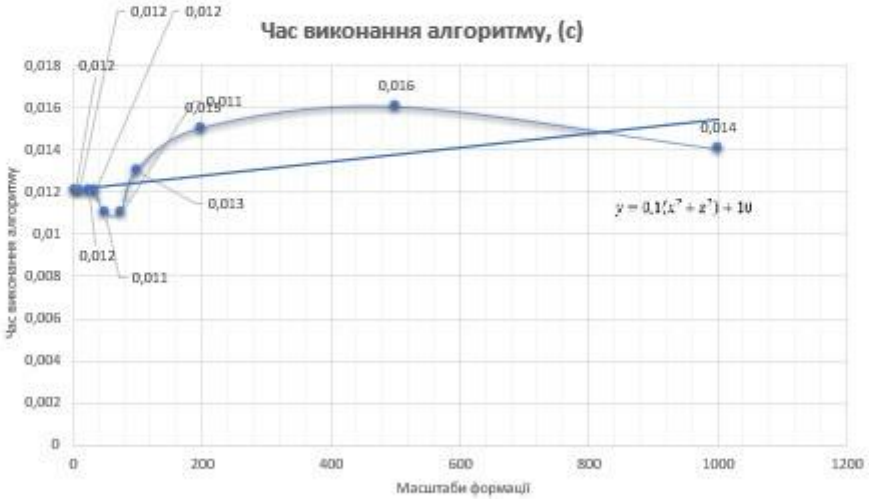


Рис. 3 Графік значення масштабів формації на час роботи алгоритму

9

РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ ВПЛИВУ ЧИСЕЛЬНОСТІ MAX ВАГИ

Задля дослідження швидкості пошуку від значення max ваги оператора, визначена функція $y = 3x^2 + xz + 2z^2 - x - 4z$.

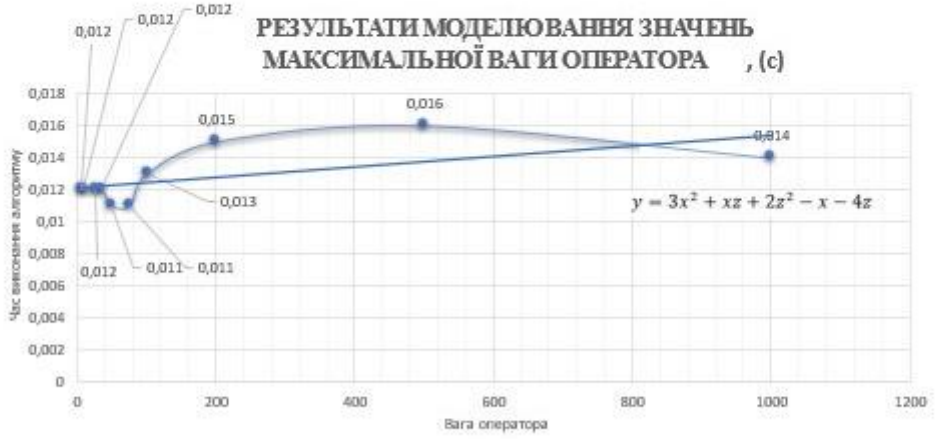


Рис. 4 Графік впливу максимальної ваги оператора на час роботи алгоритму

10

РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ ВПЛИВУ ІТЕРАТИВНИХ ОБРОБОК

У таблиці 1 продемонстрований вплив чисельності ітеративних процесів на точність та час завершення алгоритму, для функції $= 0{,}1(x^2 + z^2) + 10$.

Таблиця 1

Вплив чисельності ітеративних обробок на точність та час виконання алгоритму

Чисельність ітеративних обробок	Точність виконання алгоритму, (%)	Час виконання алгоритму, (с)
15	99,9412	0,007
25	99,97482	0,045
35	99,987	0,055
50	100	0,056
75	100	0,033
100	100	0,038
200	100	0,045
350	100	0,11
500	100	0,115

11

РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ ЗІСТАВЛЕННЯ ПАРАМЕТРІВ РЕЗУЛЬТАТИВНОСТІ РОБОТИ АЛГОРИТМІВ FSS ТА GA

Даний графік демонструє результати обчислень значень оптимуму функції, з різною кількістю змінних функції Розенброка :

$$f(x) = \sum_{i=1}^{n-1} 100\left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2, i = 1{:}n - 1$$

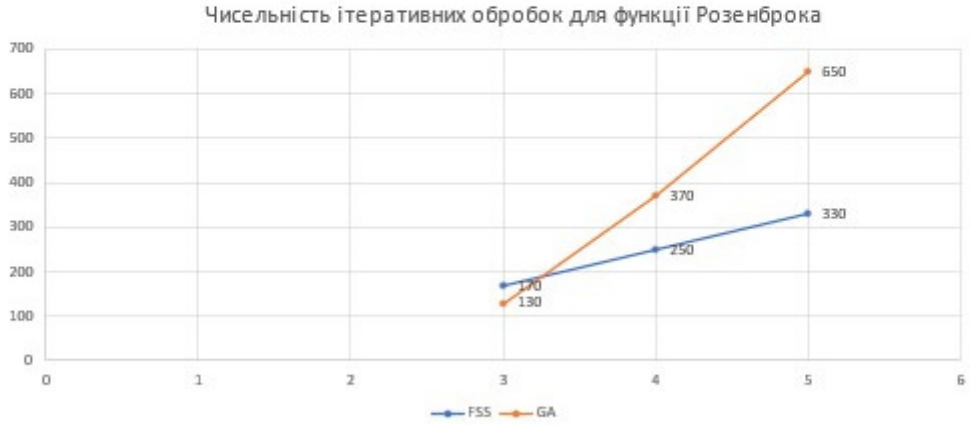


Рис. 5 Графікоптимуму функціїРозенброка

12

ВИСНОВКИ

1. Проведено аналіз сучасних методів глобальної оптимізації, виділено їх переваги та обмеження.

2. Модифіковано алгоритм косяка риб (FSS), адаптований для задач глобальної оптимізації.

3. Розроблено програмний продукт для вирішення задачі глобальної оптимізації, який дозволяє ефективно знаходити глобальні екстремуми багатомірних функцій з різними характеристиками ландшафту.

4. Проведено чисельні експерименти, які підтвердили високу ефективність запропонованих алгоритмів порівняно з традиційними підходами . Проведено аналіз ефективності GA та FSS, зокрема впливу параметрів, таких як розмір популяції, кількість ітерацій, ймовірність схрещування, початковий та кінцевий кроки.

5. Було встановлено, що обидва алгоритми демонструють високу точність (до 100%) для складних функцій, хоча час виконання залежить від обраних параметрів. Під час аналізу було виявлено, що алгоритм FSS виявився швидшим для задач з більшою кількістю змінних, тоді як GA краще працює для функцій з меншою кількістю змінних.

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Статті:

1. Бриль В. В., Задонцев Ю. В. Еволюційні алгоритми глобальної пошукової оптимізації // Наукові записки Державного університету інформаційно-комунікаційних технологій, 2024. – **подано до друку**

Тези доповідей:

1. Бриль В.В., Худік Б.О., Огляд алгоритмів глобальної пошукової оптимізації. // Всеукраїнська науково-технічна конференція «ЗАСТОСУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЯХ». – Київ: ДУТ, 2024. – С. 137
2. Бриль В.В., Худік Б.О., Оптимізація інформаційної інфраструктури для задач аналізу супутникових даних. // IV ВСЕУКРАЇНСЬКА НАУКОВО-ПРАКТИЧНА КОНФЕРЕНЦІЯ «СУЧАСНІ ІНТЕЛЕКТУАЛЬНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В НАУЦІ ТА ОСВІТІ». – Київ: ДУТ, 2024. – С. 292