Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

ДОСЛІДЖЕННЯ ЕВОЛЮЦІЙНИХ МЕТОДІВ ДЛЯ ГЛОБАЛЬНОЇ ПОШУКОВОЇ ОПТИМІЗАЦІЇ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-19-1
Магніцький Є.Д.
(прізвище, ініціали)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Шафроненко А.Ю.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Магніцькому Євгенію Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження еволюційних методів для глобальної пошукової оптимізації

затверджена наказом університету від 15 травня 2023 року № 474 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 27 травня 2023 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, анкетні дані, середовище розробки Visual Studio.

4. Перелік питань, що потрібно опрацювати в роботі
1. Розгляд та аналіз методів глобальної пошукової оптимізації.
2. Розгляд та моделювання еволюційних методів.
3. Підбір програмних методів та програмна реалізація методу.
4. Імітаційне моделювання роботи алгоритмів.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Аналіз предметної області, постановка задачі, називання методів, аналіз отриманих результатів.

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

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Консультант з дотримання діючих стандартів та норм	Доцент Творошенко І.С.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	10.04.2023	
2	Аналіз завдання, підбір літератури	11.04.23-17.04.23	
3	Аналіз літератури з досліджуваної проблеми	18.04.23-20.04.23	
4	Аналіз технічних засобів	21.04.23-30.04.23	
5	Розробка методу	01.05.23-14.05.23	
6	Програмна реалізація	15.05.23-23.05.23	
7	Оформлення пояснювальної записки	24.05.23-26.05.23	
8	Перевірка на плагіат	29.05.23	
9	Рецензування	30.05.23	
10	Підготовка презентації та доповіді	29.05.23-30.05.23	
11	Занесення роботи в електронний архів	31.05.23	
12	Попередній захист кваліфікаційної роботи	05.06.23	

Дата видачі завдання 10 квітня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Шафроненко А.Ю.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 62 с., 3 табл., 19 рис., 32 джерела.

КЛАСТЕРНИЙ АНАЛІЗ, БАГАТОЕКСТРИМАЛЬНІ ДАНІ, ФУНКЦІЇ ЩІЛЬНОСТІ, ЕВОЛЮЦІЙНІ АЛГОРИТМИ, АЛГОРИТМ ЗГРАЇ РИБ, ГЕНЕТИЧНИЙ АЛГОРИТМ.

Об’єктом роботи є вибірка анкетних даних, введених користувачем.

Метою роботи є розробка та дослідження еволюційних методів для глобальної пошукової оптимізації.

Розглянуто задачу кластеризації масивів даних, що описано як у векторній, так і матричній формах на основі оптимізації функцій щільності розподілу даних у цих масивах. Для оптимізації цих функцій – пошуку локальних екстремумів запропоновані методи Fish School Search, та Genetic Algorithm випадкового пошуку та еволюційної оптимізації.

У результаті роботи здійснена програмна реалізація еволюційних методів для глобальної пошукової оптимізації та здійснене імітаційне моделювання методів, проведено аналіз роботи еволюційних алгоритмів.

CLUSTER ANALUSIS, MULTI-EXTREME DATA, DENSITY FUNCTIONS, EVOLUTIONARY ALGORITHMS, FISH SWARM SCHOOL, GENETIC ALGORITHM.

The object of the work is a sample of questionnaire data entered by user.

The purpose of the work is the development and the research evolutionary methods for global search optimization.

The problem of clustering data arrays is considered, which is described in both vector and matrix forms based on the optimization of data distribution density functions in these arrays. For the optimization of these functions – the search for local extrema, the methods of Fish School Search and Genetic Algorithm of random search and evolutionary optimization are proposed.

As a result of the work, the software implementation of evolutionary methods for global search optimization was carried out and simulation modeling of the methods was carried out, the analysis of the work of evolutionary algorithms was carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

GA – Genetic Algorithm (генетичний алгоритм)

FSS – Fish Swarm School (алгоритм зграї риб)

PSO – Particle Swarm Optimization (алгоритм рою частинок)

SA – Simulated Annealing (алгоритм імітації випалення)

SOA – Swarm-Based Optimization Algorithm (алгоритм, який оснований на поведінці бджіл)

ВСТУП

Дослідження еволюційних методів для глобальної пошукової інформації є актуальним і важливим напрямком досліджень в галузі штучного інтелекту та оптимізації.

Глобальна оптимізація використовується для пошуку оптимальних рішень в складних проблемах, де існує велика кількість можливих рішень. Це можуть бути задачі оптимізації функцій, пошук оптимальних параметрів моделей, або пошук оптимальної конфігурації системи.

Еволюційні методи засновані на ідеях природнього відбору та еволюції, і вони можуть ефективно вирішувати проблеми глобальної оптимізації. Ці методи працюють за принципом створення популяції кандидатів на рішення і послідовного цих рішень через ітераційний процес. Популяція проводить еволюцію, змінюючи свою структуру та параметри, що дозволяє знаходити кращі рішення з плином часу.

Актуальність роботи полягає в тому, що еволюційні методи можуть працювати на проблемах зі складною функцією цілі, великою кількістю обмежень та нелінійними залежностями між змінними. Також, вони можуть бути використані для пошуку оптимальних рішень в проблемах, де важко визначити аналітичний вираз для функції цілі або її похідних. Крім того, еволюційні методи можуть бути ефективними в задачах оптимізації, де цільова функція не має гладкого або диференційованого вигляду, або коли вона є недостатньо визначеною. Вони можуть допомогти знайти приблизне рішення коли точні методи можуть бути неефективними та обмеженими.

Отже, дослідження еволюційних методів для глобальної пошукової оптимізації залишається актуальним завданням, оскільки ці методи можуть знайти широке застосування в різних сферах, де є пошук оптимальних рішень великої складності.

1 ЕВОЛЮЦІЙНІ АЛГОРИТМИ ДЛЯ ВИРІШЕННЯ ЗАДАЧ ОПТИМІЗАЦІЇ ТА ЇХ РОЗГЛЯД

1.1 Глобальна оптимізація

Глобальна оптимізація – це процес пошуку найкращого рішення у заданому просторі параметрів, де можливі багато локальних мінімумів, або максимумів. Глобальна оптимізація може бути застосована в різних областях, а саме: наука, інженерія, будівництво та інше.

Глобальна оптимізація ставить за мету знаходження найкращого рішення, яке відповідає заданим критеріям у всьому просторі параметрів, забезпечуючи перехід через всі можливі мінімуми та максими. Це вимагає великої кількості обчислень та розрахунків, особливо при складних функціях або при великому простору даних [1].

1.2 Основні поняття глобальної оптимізації

Насамперед, глобальна оптимізація – це пошук стану деякої системи (виробничої, технічної, або бізнес-системи), який забезпечує її оптимальне функціонування (максимізує прибуток, мінімізує витрати), а також комплекс заходів, які спрямовані для досягнення даного стану [2].

В аналізі даних глобальна оптимізація – це розділ прикладної математики та обчислювальної науки, яка вивчає методи та алгоритми для знаходження глобальних мінімумів або максимумів функцій, які можуть мати дуже складні форми та містити багато локальних екстремумів. В більшості випадків буде вирішуватися завдання мінімізації, тому що максимізація завжди еквівалентна зворотного функціоналу для завдання мінімізації.

Оптимізація – це процес знаходження точки функції, яка мінімізує дану функцію. Локальний мінімум функції – це значення функції, яке менше або

рівне значенням функції в будь-якій точці, розташованій поруч з даною точкою. Інакше кажучи, локальний мінімум – це точка, в якій значення функції досягає найменшого значення на деякому найменшому проміжку. Локальний максимум функції – це значення функції, яке більше або рівне значенням функції в будь-якій точці, розташованій поруч з даною точкою. Іншими словами, локальний максимум являє собою точку в якій функція досягає найбільшого значення на деякому обмеженому проміжку.

Важливо зазначити, що точка, в якій функція має локальний мінімум або максимум, може бути точкою глобального мінімуму або максимуму функції, якщо вона має єдине значення на всій області визначення функції. Однак, в інших випадках, функція може мати кілька локальних мінімумів та максимумів, тоді глобальний мінімум або максимум будуть знаходитись в інших точках області значення.

Тим не менше, глобальна оптимізація є більш складнішою, чим локальна, оскільки чисельні методи в більшості випадків призводять до дуже складних рішень, і по цій причині неможливо застосувати аналітичні методи.

Звичайно, методи глобальної оптимізації – це клас методів математичної оптимізації, що застосовуються для пошуку глобального мінімуму або максимуму функції, яка може бути нелінійною, нестачною та з нескінченною кількістю локальних мінімумів або максимумів [3].

1.3 Розгляд еволюційних алгоритмів

Далі будуть розглянуті такі еволюційні алгоритми що використовуються для глобальної оптимізації як генетичний алгоритм, алгоритм рою частинок, алгоритм поведінки бджіл, алгоритм риб’ячої зграї.

1.3.1 Генетичний алгоритм

Генетичний алгоритм [2] – це алгоритм, який являє собою метод, що відображає природну еволюцію методів вирішення проблем і задач оптимізації. Насамперед, генетичні алгоритми являють собою процедури пошуку, які засновані на механізмах природного відбору та спадкоємства. В даних алгоритмах використовується еволюційний принцип найбільш пристосованих особин. Від традиційних алгоритмів генетичні алгоритми декількома базовими елементами:

– генетичні алгоритми обробляють закодовану форму значень параметрів завдання, а не їх значення;

– генетичні алгоритми здійснюють пошук рішень виходячи з деякої популяції точок, а не з єдиної точки;

– генетичні алгоритми тільки цільову функцію, а не її похідні або іншу додаткову інформацію;

– генетичні алгоритми застосовують не детерміновані правила вибору, а імовірнісні.

Під час опису генетичних алгоритмів використовуються визначення, які були запозичені з генетики. Наприклад, якщо мова йде про популяцію особин, в якості базових значень зазвичай використовуються ген, хромосома, генотип, фенотип, алель. Також можуть бути використані відповідні до цих термінів визначення з технічного лексикону: ланцюг, двійкова послідовність, структура та інші.

Популяція – кінцева множина особин [4].

Особини, вхідні в популяцію особин у генетичних алгоритмах зазвичай представляються хромосомами, в які закодовані множини параметрів, які інакше називаються точками в просторі пошуку (search points). Хромосомами зазвичай називають впорядковані послідовності генів особини. Генами називають атомарні елементи фенотипу. Генотипом зазвичай називають набор хромосом особи. Особинами популяції можуть бути генотипи або одиничні

хромосоми. Алель – це значення конкретного гена, яке також визначається як значення властивості.

Важливим поняттям у генетичних алгоритмах вважається функція пристосованості (fitness function), яка ще називається функцією оцінки. Дана функція являє собою міру пристосованості даної особини в популяції. Вона відіграє найважливішу роль в оцінці ступеню пристосованості конкретної особини в популяції та дозволяє віднайти найбільш пристосовані особини до життя відповідно з еволюційним принципом виживання «найсильніших». Безпосередньо дана функція отримала свою назву із генетики. Вона повинна мати точне і коректне визначення тому що вона надає сильний вплив на функціонування генетичного алгоритму. Функція пристосованості у задачах оптимізації максимізується і має назву «цільова функція». Під час ріішення задач мінімізації цільова функція , і проблема зводиться до максимізації. У теорії похибки дана функція приймає вигляд функції похибки, а теорії ігор вона приймає вигляд вартісної функції.

За допомогою функції пристосованості на кожній ітерації генетичного алгоритму оцінюється пристосованість кожної особини популяції, на цій основі створюється наступна популяція особин, які, в свою чергу, являють собою безліч потенційних рішень проблеми. Чергова популяція в генетичному алгоритмі називається поколінням, а до новостворюваної популяції особин застосовується термін «нове покоління» або «покоління нащадків».

1.3.2 Алгоритм рою частинок

Алгоритм рою частинок (англ. Particle Swarm Optimization, PSO) [5] – стохастичний алгоритм оптимізації, який використовує поняття «рій» для оптимального рішення проблеми.

В алгоритмі PSO кожне рішення (або «частинка») представлено вектором параметрів, а кожна з частинок має швидкість та позицію в просторі

параметрів. Алгоритм ітеративно оновлює швидкість та позицію кожної частинки на основі її власного досвіду, який є найкращим результатом, якого досягла частинка коли-небудь та досвіду найкращої частинки з усього рою.

Під час кожної з ітерацій алгоритму PSO частинки переміщуються в сторону кращої з позицій, яку вони коли-небудь досягли, і кращої позиції, якої досягала будь-яка частинка рою. Напрямок руху частинки визначається швидкістю кожної частинки та вагами, які повинні визначити, наскільки сильно кожен з двох впливів (кращій особистий і найкращій загальний) може впливати на швидкість частинок.

Зазвичай алгоритм PSO використовується для вирішення широкого спектру задач оптимізації, включаючи задачі з безперервним, дискретним та комбінаторним простором параметрів. Також даний алгоритм використовується для оптимізації функцій, нейронних мереж, моделей машинного навчання а також для вирішення завдань планування та керування.

1.3.3 Алгоритм випалення

Алгоритм імітації випалення (англ. Simulated Snnealing SA) [6, 7] – це евристичний алгоритм для глобальної оптимізації, який є найбільш ефективним для рішення дискретних та комбінаторних задач.

Даний алгоритм оснований на імітації фізичного процесу, використовуваного в металургії, а саме в нагріванні та контрольованій кристалізації речовини такої я камінь, розплавлений метал для збільшення його кристалізованості та зменшення дефектів речовини. Симуляція випалення в задачах перебору зазвичай використовується для приближеного находження глобального мінімума функцій з деякою великою кількістю вільних змінних. Алгоритм випалення майже не дає ніяких гарантій сходження, але на практиці добре працює під час вирішення NP-повних задач.

Алгоритм випалення починається з деякого початкового стану, зазвичай, вибраного випадковим образом. Потім відбувається процес ітерацій, під час якого відбувається зміщення до нового стану шляхом випадкових змін. Також під час кожної з ітерацій підраховується значення оптимізованої функції.

Основна ідея алгоритму випалення полягає в дозволенні випадкових змін стану на початку процесу та поступово зменшувати їхню ймовірність у міру наближення алгоритму до оптимального рішення. Це досягається під час зменшення «температури» під час кожної з ітерацій. «Температура», насамперед визначає можливість прийняття нового стану, який має більш високе значення функції, ніж поточний стан. В міру того як знижується «температура», також знижується ймовірність прийняття нового стану, який мав би більш високе значення функції і алгоритм починає зосереджуватися на пошуку локального оптимуму.

Зазвичай алгоритм відпалу завершується при досягненні певного критерія зупинки, наприклад, коли «температура» падає до певного рівня, або алгоритм досягає певний заданий ліміт числа ітерацій. Головною перевагою алгоритму відпалу є його здатність знаходити глобальні оптимуми функцій у присутності шуму чи невизначеності даних.

1.3.4 Алгоритм поведінки бджіл

Алгоритм, який оснований на поведінці бджіл (Swarm-based Optimization Algorithm, SOA) [8] для знаходження оптимальних рішень використовують природні методи. Головна відмінність алгоритмом SOA та алгоритмами прямого пошуку полягає, такими як градієнтний спуск або перебір полягає в тому, що SOA використовує не один розв’язок, а множину розв’язків при виконанні кожної з ітерацій. В випадку коли задача оптимізації має лише єдине рішення, то всі члени множини сходяться до даного рішення.

Якщо задача має декілька рішень, то алгоритм SOA може бути використаний для захоплення значень рішення в кінцеву множину.

Алгоритм поведінки бджіл це один з самих останніх алгоритмів, які були створені Дервісом Карабогою у 2005 році. За основу даного алгоритму було взято інтелектуальну поведінку звичайних медоносних бджіл. Він використовує такі загальні параметри керування, такі як кількість бджіл в колонії та максимальне число ітерацій.

За глобальний екстремум береться та ділянка поля, де знаходиться найбільша кількість нектару, причому дана ділянка є єдиною та найкращою, тобто, на інших ділянках находиться нектар також, але його значно менше ніж на найкращій ділянці. Бджоли живуть не на площині де для визначення знаходження бджоли достатньо двох координат, а в багатовимірному просторі, де кожна координата являє собою параметр функції, який треба оптимізувати. Знайдена кількість нектару являє собою значення цільової функції в даній точці. Кожне рішення в даному алгоритмі подається під виглядом бджоли, яка зберігає інформацію про місцезнаходження (параметри багатомірної функції або координати) деякої ділянки поля, де бджола може зібрати нектар. Під час першого кроку алгоритму у точки, яка описана її координатами в просторі, відправляється певна кількість бджіл-розвідників для розвідки місцевості на наявність нектару. В залежності від значення цільової функції, яке визначено координатами бджоли на місцевості, виділяється дві перспективні ділянки на поверхні функції, близько яких знаходиться глобальний максимум. А саме:

– вибирається деяка кількість n кращих ділянок з найбільшим значенням функції;

– вибирається деяка кількість m ділянок з меншим значенням цільової функції, але все рівно вони непогані з точки зору значення цільової функції.

Передбачається, що дві бджоли знаходяться на різних ділянках, які перетинаються, але вважається що дана ділянка це ділянка, центр якої

знаходиться в точці, яка відповідає координатам, де в даний момент перебуває бджола з найбільшим значенням цільової функції.

Другий варіант поведінки розглядається коли в окіл n кращих ділянок прибуває N бджіл, а в окіл m ділянок прибуває M бджіл, причому на кращих ділянках кількість бджіл має бути більшою, ніж на кожній з прийнятих ділянок. Чим більше значення цільової функції, тим більше бджіл має бути відправлено на дану ділянку. Бджоли прибувають не точно на ділянку, в яку вони були відправлені, а на окіл ділянок, координати яких вибрані випадковим чином. Крім того окіл ділянки, на яку прибуває бджола зменшується під час змінення кількості ітерацій. Якщо область зменшувати занадто швидко то рішення застряє в локальному екстремумі.

Після відправки бджіл-збирачів на кращі та прийняті ділянки можна відправити бджіл-розвідників на інші ділянки.

Алгоритм працює поки не спрацює один з критеріїв зупинки: якщо ми знаємо значення цільової функції в глобальному екстремумі, то можемо алгоритм виконується до знаходження деякого значення, яке близьке до бажаного; в випадку коли значення функції в екстремумі невідоме, кроки алгоритму повторюються поки протягом деякої кількості ітерацій рішення буде покращуватись.

1.3.5 Алгоритм зграї риб

Скупчення риби, насамперед, це загальний термін для де-якої групи риб, що зібрались в якомусь місці. Скупчення риби бувають як структурованими так і не структурованими. Неструктуровані скупчення зазвичай складаються з різних риб різних видів різних розмірів, які випадково зібрались біля місць їх інтересу (локальні ресурси з яких складається їх кормова база, або місця для нересту риби). Тим часом, група риб, яка тримається як колектив з соціальних причин вже вважається зграєю. Більша частина особин може перебувати в

однаковій фазі життєвого циклу, також особини активно контактують між собою і будь-якої миті можуть проявляти біологічно активну та корисну організовану діяльність для кожного з членів групи риб. Відміною індивідуального способу існування де немає конкуренції у добуванні їжі, та є висока можливість стати кормом для хижаків над зграйним способом існування живих організмів є компроміс між перевагою життя в групі в плані захищеності від хижаків в силу того, що в групі, де багато особин, хижаку легко втратити свою ціль, та посиленням конкуренції у добуванні їжі [9].

Зазвичай, зграї риб утворюються в природі за кількома параметрами. Риби, як правило, віддають перевагу великим косякам риби, які включають в себе тільки особини одного виду. Якщо в зграї є член, який зовнішніми ознаками відрізняється від більшості особин з зграї, то він зазвичай стає переважною мішенню для хижаків. Даний факт пояснює те, чому в дикій природі риби частіше збираються в зграї з особин, які ідентично схожі на себе, тим самим, зумовлюючи повну гомогенність зграї.

Для того щоб пояснити поведінку зграї було представлено безліч гіпотез, які пояснюють поведінку риби. А саме:

– орієнтація руху риби;

– синхронізація полювання риби(якщо риби хижаки);

– заплутування хижаків для зниження ризику стати кормом даного хижака;

– кормова поведінка однієї з риб швидко стимулює всю зграю повторювати те ж саме;

– зграйний спосіб життя риб має на увазі не, що у риб є достатньо розвинуті сенсорні механізми, які здатні миттєво реагувати на зміни в середовищі по відношенню до свого сусіда.

Алгоритм зграї риб (Fish Swarm School, FSS) – це мета евристичний алгоритм глобальної оптимізації, який був розроблений як і всі еволюційні алгоритми під час спостережень за поведінкою тварин, насамперед, риб, для використання даного алгоритму в вирішенні задач оптимізації.

Цей алгоритм базується на поведінці риб в природі, де було зафіксовано що зграям риби зграйність, координація руху, своєчасне сповіщення про приближення хижака, та інше.

Сам алгоритм складається з кількох етапів:

– ініціалізація – створення початкової популяції особин;

– оцінка – обчислення функції придатності (Fitness function) для кожної особини популяції;

– керування – регулювання швидкості та напряму руху кожної особи залежно від позиції та поведінки інших особин зграї;

– адаптація – зміна позиції кожного організму залежно від її швидкості та напряму руху;

– обмін інформацією – обмін кращими рішеннями між рибами з метою покращення рівня придатності популяції в цілому;

– перевірка критерію зупинки – перевірка, чи був досягнутий заданий ліміт точності, або часовий ліміт;

– завершення – повернення кращого рішення, яке було знайдено протягом процесу оптимізації.

Алгоритм зграї риб може бути застосований для вирішення різних задач оптимізації, таких як пошук максимуму або мінімуму функції, вибір параметрів моделі, задачі класифікації та кластеризації даних.

1.4 Порівняльний аналіз еволюційних алгоритмів

Далі описані переваги та недоліки у таблиці 1.1 тих алгоритмів, які використовуються для глобальної оптимізації, як генетичний алгоритм, алгоритм рою частинок, алгоритм випалення, алгоритм поведінки бджіл, алгоритм зграї риб.

Таблиця 1.1 – Переваги та недоліки еволюційних алгоритмів

Алгоритм	**Переваги**	**Недоліки**
1	**2**	**3**
Генетичний алгоритм	– Використовується для оптимізації складних функцій, де інші методи оптимізації можуть не впоратися. може працювати з великою кількістю змінних і обмежень, що робить його ефективним для оптимізації завдань що вимагають великої кількості ресурсів; – він є стохастичним методом оптимізації, може мати високу швидкість збіжності та стійкість до локальних оптимумів; – застосовується для оптимізації числових функцій, так і інших типів завдань.	– Дуже обчислювально витратний при великих розмірах простору пошуку та має велику кількість обмежень; – продуктивність сильно залежить від конкретної установки параметрів; – у деяких випадках алгоритм призводить до отримання неможливих або неприйнятих значень, потребує додаткової обробки значень; – менш точний ніж інші методи оптимізації, що може бути недостатнім для деяких завдань.
Алгоритм рою частинок	– Відносно простий в реалізації, не потребує великої кількості налаштувань; – має високу збіжність, яку можна порівняти з іншими методами оптимізації; – під час глобальної оптимізації алгоритм може знаходити глобальний оптимум, що являється важливою важливістю для багатьох завдань оптимізації	– Алгоритм може бути чутливим до початкових умов, що може призвести до різних оптимальних рішень під час різних запусків алгоритму; – алгоритм може застрягати в локальних мінімумах, якщо група частинок потрапить до зони де оптимальне рішення знаходиться поблизу локального мінімуму; – має проблеми з різноманітністю рішень, особливо

Продовження таблиці 1.1

1	2	3
	має гарну продуктивність на багатьох задачах оптимізації.	на задачах з великою кількістю параметрів.
Алгоритм випалення	– Алгоритм випалення є досить простим та швидкісним методом для видалення шуму з зображень що дозволяє ефективно використовувати його для обробки великих обсягів даних; – алгоритм дуже легкий в реалізації та простий в розумінні; – до певної міри алгоритм стійкий до зміни зображень, таких як зміна контрастності чи яскравості.	– Алгоритм може видаляти не тільки шум, а і деякі корисні деталі зображення, що погіршує якість остаточного результату; – алгоритм впливає на контрастність та яскравість зображення, що призводить до погіршення якості зображення; – алгоритм не ефективний для видалення великих обсягів шумів, також він може видаляти не всі шуми, а тільки ті, що перевищують загальний поріг.
Алгоритм поведінки бджіл	– Завдяки тому що багато бджіл працюють над пошуком глобального рішення одночасно і обмінюються інформацією, алгоритм може знайти оптимальне рішення швидко; – алгоритм рою бджіл може знаходити глобальний мінімум або максимум функції, що робить його корисним для глобальної оптимізації; – працює навіть при втраті деяких бджіл.	– Високий рівень обчислюваної складності та високі вимоги для ресурсів, оскільки даний алгоритм використовує багато ітерацій для знаходження оптимального рішення; – алгоритм дуже чутливий до початкових умов та параметрів що може призвести до неефективної та нестабільної роботи алгоритму; – використання даного алгоритму обмежене у випадку, коли не вдається отримати достатньо точні дані, які потребують оптимізації.

Продовження таблиці 1.1

1	2	3
Алгоритм зграї риб	– Алгоритм зграї риб дуже ефективний у розв'язанні задач оптимізації; – даний алгоритм дуже адаптуємий для рішення різних задач оптимізації. Він може бути використаний для пошуку мінімуму функції, класифікації даних, кластеризації та ін; – простий в і реалізації та використанні. Може бути використаний в різних мовах програмування; – алгоритм може бути масштабований для вирішення великих задач оптимізації, що включають багато змінних; – алгоритм може бути використаний для рішення задач в реальному часі, так як він може бути оновлений динамічно, що дозволяє отримувати нові рішення в режимі реального часу.	– Алгоритм зграї риб має багато параметрів, які потрібно налаштувати, що може зайняти багато часу і зусиль; – алгоритм може потребувати багато ітерацій для досягнення оптимального розв'язку, що призводить до повільної швидкості збіжності; – алгоритм не підходить до всіх типів оптимізації, але може бути ефективним для певних типів оптимізації; – алгоритм не гарантує збіжності до оптимального розв'язку. Він може виявитися у локальному мінімумі або максимумі.

1.5 Постановка задачі

В даній роботі треба розглянути та проаналізувати деякі еволюційні алгоритми. Також треба вибрати два еволюційних алгоритми, які ляжуть за виконання проєктної області. Насамперед, це алгоритми GA та FSS. Також

буде потрібно розробити програмний застосунок, який буде включати роботу даних алгоритмів.

Об’єктом роботи є вибірка анкетних даних, введених користувачем.

Метою роботи є розробка та дослідження еволюційних методів для глобальної пошукової оптимізації.

Для досягнення мети необхідно вирішити такі задачі:

- провести аналіз існуючих алгоритмів глобальної оптимізації;
- реалізувати генетичний алгоритм;
- реалізувати алгоритм зграї риб;
- реалізувати комп’ютерну модель для виведення графіків.

2 РОЗГЛЯД АЛГОРИТМІВ ДЛЯ РОЗРОБКИ

2.1 Розгляд генетичного алгоритму (GA)

Генетичний алгоритм [10] – це алгоритм пошуку, який моделює природний процес еволюції для знаходження оптимальних рішень задачі. Генетичні алгоритми використовуються в різних галузях, таких, як штучний інтелект, оптимізація, машинне навчання та інші. Насамперед, основна ідея генетичного алгоритму полягає в тому, що він працює з популяцією можливих рішень замість роботи з одним окремих рішень. Кожен елемент популяції представляється як послідовність генів, де кожен ген кодує один параметр рішення.

Генетичний алгоритм складається з наступних кроків [11]:

- ініціалізація – це вибір вихідної популяції хромосом;
- оцінка пристосованості хромосом в популяції;
- перевірка умови зупинки алгоритму;
- селекція хромосоми;
- застосування генетичних операторів;
- формування нової популяції;
- вибір найбільш пригідної для рішення задачі хромосоми.

Кожна особина генетичного алгоритму являє собою одне значення x серед безлічі можливих рішень. Також, для кожного значення x шукаємо значення мінімуму функції.

Структурна блок-схема генетичного алгоритму показана на рисунку 2.1.

Перший етап роботи програми, яка використовує еволюційний алгоритм полягає в тому, що користувач даного додатку вводить дані для пошуку.

Перший етап роботи алгоритму полягає в тому, що створюється початкова популяція, тобто, створюється безліч різних особин с безліччю різних хромосом у них (x).

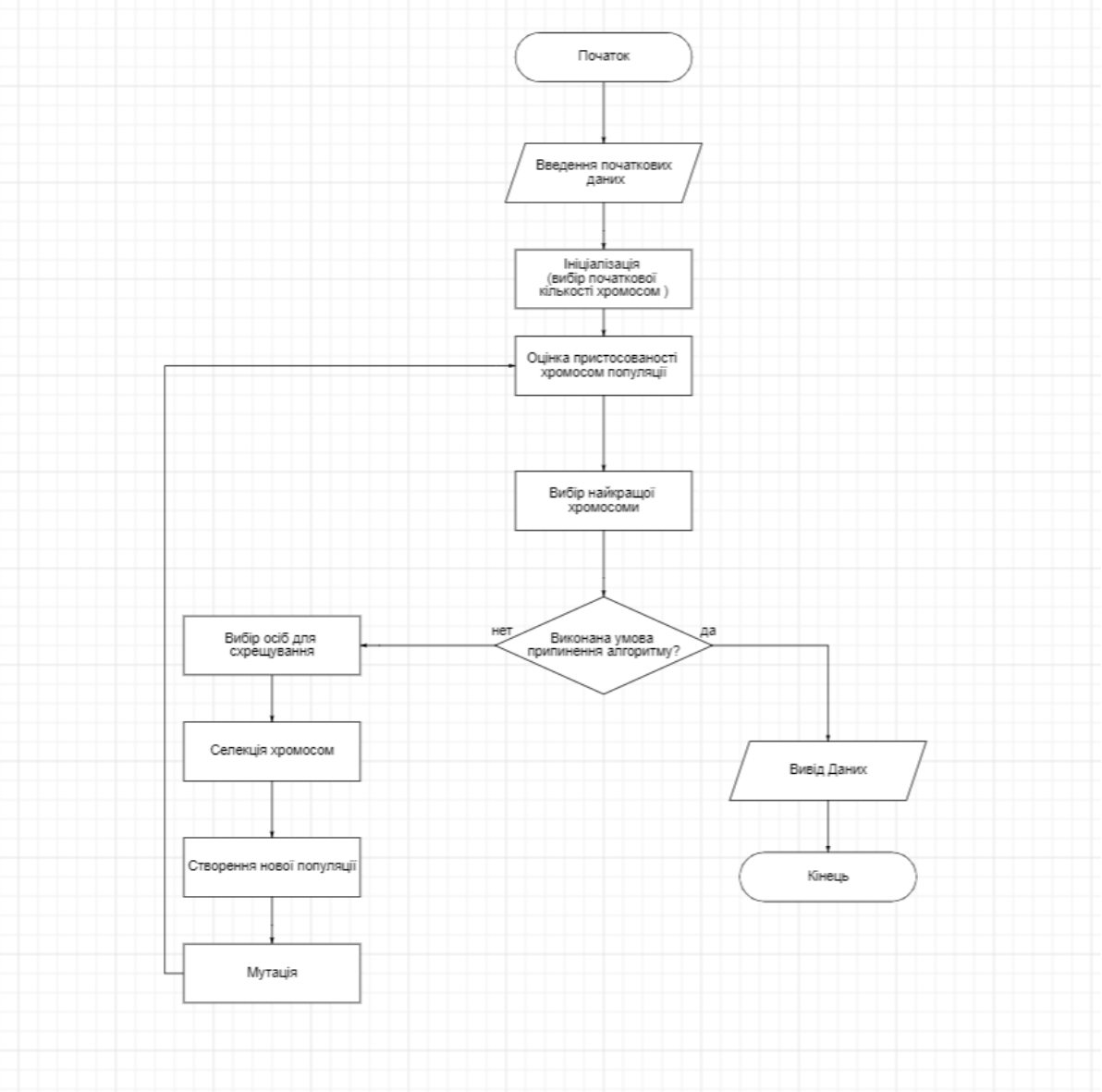

Рисунок 2.1 – Блок-схема генетичного алгоритму

Початкова популяція, насамперед, створюється з особин, які мають випадкове значення хромосом, значення хромосом популяції повинні відносно рівномірно покривати всю область рішення, якщо не присутствує припущень до того, де перебуває глобальний екстремум.

Чим більше особин популяції буде створено на даному етапі, тим більше стає ймовірність того, що алгоритм знаходить правильне рішення, також, як правило, при збільшенні кількості особин початкової популяції потрібна менша кількість ітерації генетичного алгоритму, тобто, треба утворити меншу кількість поколінь. При зростанні розміру популяції потрібно робити більшу

кількість розрахунків цільової функції та виконання інших генетичних операцій алгоритму також. З чого можна сказати що при збільшенні розміну популяції збільшується час розрахунку одного покоління.

Розмір популяції не обов’язково буде залишатись незмінним під час робити генетичного алгоритму, часто популяцію можна зменшити зі збільшенням кількості поколінь, тому що більша кількість особин вже буде розташована ближче до шуканого рішення, однак, часто розмір популяції приблизно залишається постійним [12].

На даному етапі еволюційного алгоритму проводиться оцінка пристосованості хромосом в популяції. Вона полягає в тому, що проводиться розрахунок функції пристосованості кожної хромосоми даної популяції. Чим значення функції буде більше, тим вище буде «якість» хромосоми. Сама форма функції пристосованості залежить від характеру завдання. Передбачено що функція пристосованості завжди має невід’ємні значення, і, крім того, що для вирішення оптимізаційної задачі потрібно дану функцію максимізувати. Але якщо вихідна форма функції пристосованості задовольняє дані умови, то виконується відповідне перетворення.

Насамперед, схрещування – це генетична операція, в результаті виконання якої створюються нові особини популяції з новими значеннями хромосом. Нові хромосоми виводяться на основі хромосом батьків даних особин. Часто під час процесу схрещування деякого набору партнерів утворюється одна дочірня особа, але також може бути й більша кількість дочірніх особин, що може бути реалізовано в різних випадках реалізації генетичного алгоритму.

Сам сенс схрещування полягає в тому, що при взятті одної частини хромосоми одної особини, та другої частини хромосоми іншого організму (особини) та зданих хромосом створюється інша хромосома вже іншого організму, яка вже буде записана в новому вигляді. За часту хромосоми схрещують побітово. Суть побітового схрещування полягає в тому, що спочатку вибирається випадково точка розриву (яку по іншому називають

схрещування) хромосоми, потім створюється нова хромосома, яка вже буде складатися з лівої частини одної хромосоми та правої частини іншої хромосоми.

Наприклад, є дві хромосоми, які для простоти можна представити 8-бітовим: 10011011 і 11000101. Випадково може бути вибрана точка розриву, яка бути позначена символом «\»:

– 100\11011;

– 110\00101.

Звідси можна виразити дві різні хромосоми: 100 00101 та 110 11011. Хромосому, яку треба вибрати вирішує генератор випадкових чисел. Таким шляхом можна шукати всі точки перетину.

Мутація – це важлива частина генетичного алгоритму, яка підтримує різноманітність хромосом та зменшує шанси того, що рішення зможе зійтись до якогось локального мінімуму замість глобального мінімуму. Мутація являє собою випадкову зміну в хромосомі особини, яка була створена шляхом селекції (схрещування).

Ймовірність мутації, як правило, встановлюється не дуже великою тому, що мутація буде заважати алгоритму псуючи особини з вдалими хромосомами.

Так як і для схрещування, для виконання мутації можна використати різні алгоритми. Наприклад, замінити одну хромосому, або декілька на випадкову величину. В даній роботі мною була використана побітова мутація. Це коли в даній хромосому замінюється значення лише одного біта. Приклад показаний на рисунку 2.2.

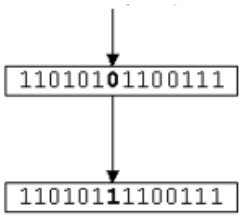

Рисунок 2.2 – Мутація одного біта

В результаті мутацій можуть вийти як і більш вдалі, так і менш вдалі особини. Під час мутації існує ймовірність вивести вдалу хромосому, у якої будуть набори одиниць та нулів, яких нема в батьківських хромосом.

Результатом схрещування та подальшої мутації з'являються нові особини з новим значенням хромосом. Якщо би не було етапу відбору, то кількість організмів зростала би за експоненціальною шкалою, що би вимагало більшої та більшої кількості часу та оперативної пам'яті для обробки кожного нового покоління особин. Як і в природі, коли появляється настільки велика кількість особин одного виду на певній території, а стара популяція не вмирає, всьому виду на даній території грозить вимирання в зв'язку з нехваткою кормової бази. Тому після виведення нової популяції в генетичному алгоритмі, потрібно чистити популяцію на наявність найменш вдалих особин.

Алгоритми відбору, як і алгоритми схрещення, можуть бути різними також. За часту, в першу чергу відкидаються такі особини, значення хромосом яких не потрапляють в інтервал пошуку рішення. Під час написання програмного застосунку я сортував види за значенням їх цільових функцій залишаючи тільки найбільш «живучі» види.

Умови закінчення алгоритму полягають в тому, що задано певну кількість поколінь (ітерацій). Дані умови треба використовувати обережно тому, що генетичний алгоритм ймовірнісний, та різні запуски алгоритму можуть сходитися з різною швидкістю. Ось чому в якості порога кількості ітерацій треба задавати досить велике число.

2.2 Алгоритм зграї риб (FSS)

В даному алгоритмі (fish swarm school) риби перебувають в області пошуку, яка, зазвичай називається «Акваріум» та виконує пошук вирішення задачі оптимізації «пошук їжі». Вага кожної рибки формалізує успіх даної рибки в виконанні пошуку рішення та роль пам'яті особини. Наявність ваги

особини популяції є основною особливістю парадигми алгоритму зграї риб в порівнянні, наприклад, з парадигмою оптимізації зграї мурах. Ця особливість алгоритму FSS на відміну від алгоритму зграї мурах позволяє відмовлятися від необхідності відшукування та фіксування глобально кращих рішень.

Основні принципи алгоритму FSS [13] наступні:

– координати риб визначають їх положення в просторі параметрів, які шукаються;

– координати риб змінюються з кожним кроком алгоритму для пошуку нових і більш оптимальних точок;

– у кожному кроці алгоритму риби обмінюються інформацією про їхнє положення та фітнес функції яка вимірює якість поточного розв’язку;

– знаходячись в зоні впливу інших риб риби можуть рухатися в напрямку кращих розв’язків;

– алгоритм FSS використовує випадковість для збільшення ймовірності пошуку більш оптимального розв’язку;

– кожна рибка вибирає в просторі параметрів свою початкову точку випадково за замовчуванням;

– алгоритм FSS має параметри, які можуть бути настроєні між експлорацією та експлуатацією розв’язків;

– алгоритм має механізм регулювання параметрів, який змінює їх відповідно до результатів пошуку для збереження балансу між експлорацією та експлуатацією;

– для покращення збіжності до оптимального розв’язку даний алгоритм використовує деякі евристики;

– FSS може використовувати декілька видів руху риб, таких як індивідуальний, колективний та рух до цілі, що дозволяє особинам більш ефективно відшукувати оптимальні розв’язки;

– FSS може бути модифікований для вирішення різноманітних задач оптимізації;

– алгоритм FSS є глобально збіжним алгоритмом, що означає що він може знайти глобальний оптимум в просторі параметрів, а не тільки локальні;

– FSS є здатним до самоконтролю, тобто автономним.

На наступному рисунку 2.3 показано структурну блок-схему алгоритму FSS.

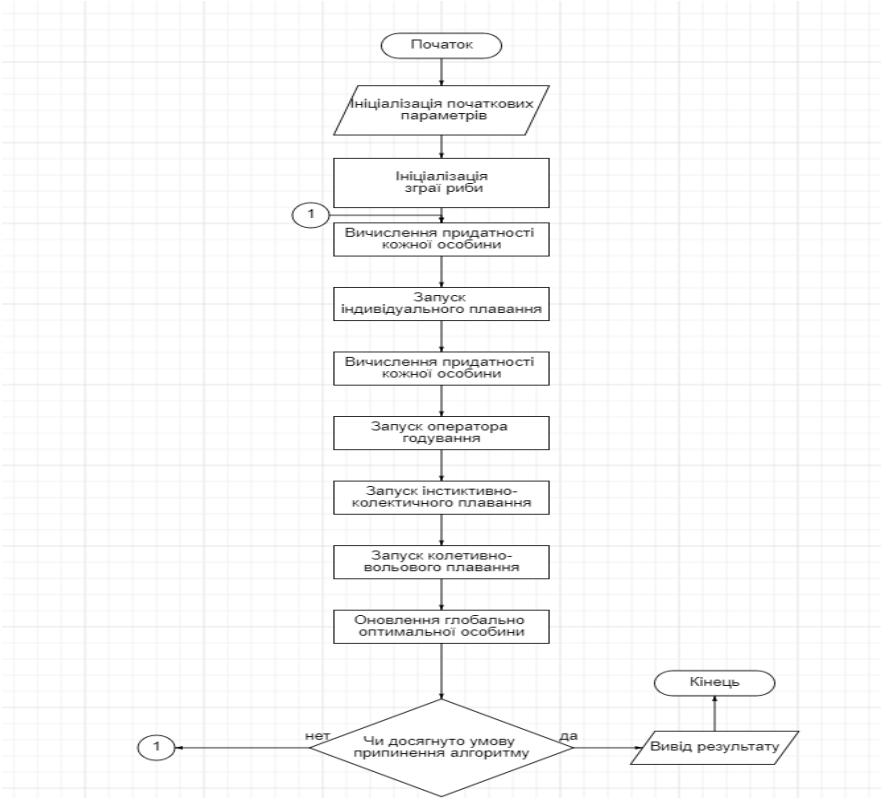

Рисунок 2.3 – Блок-схема алгоритму FSS

FSS – це алгоритм глобального пошуку інформації, який був заснований на основі поведінки кожної риби в зграї, які рухаються в пошуку їжі. При використанні методів еволюційної оптимізації, які по суті є методами оптимізації нульового порядку, передбачається, [14] що при знаходженні екстремумів деякої функції $f^x(x)$ використовується популяція агентів, які можуть діяти або незалежно, або в взаємодії з іншими, з переміщенням q^{th} агенту $(q = 1,2, \dots, Q)$ та i^{th} ітерації та записується як:

$$x_q^i = x_q^{i-1} + n_q^i Dir_q^i,\ q = 1,2, \dots, Q, \tag{2.1}$$

де $x_q^i = (x_{q1}^i, x_{q2}^i, \dots, x_{qn}^i)^T$, та Dir_q^i є вектором, який специфікується на направленні пересування q^{th} агенту в I^{th} пошуковій ітерації.

Оператор годування відповідає за вагу кожної з рибок як елемента зграї риби. Чим риба важче, тим ближче вона до екстремуму(максимуму) функції. Вага кожної рибки w_q вираховується по формулі

$$w_q^i = w_q^{i-1} + \frac{f^x(x_q^i) - f^x(x_q^{i-1})}{max\left\{f^x(x_q^i)_p - f^x(x_q^{i-1})\right\}}, \forall q = 1,2, \dots, Q, \qquad (2.2)$$

де $0 < w_q^i < w_{max}$, $w_i^0 = 0{,}5w_{max}$.

Оператор плавання описує в цілому і індивідуальний рух і колективний рух цілої зграї риби. Насамперед, в даному алгоритму розглядається три види руху: індивідуальний, колективно-вольовий, та інстиктивно-колективний.

Індивідуальний рух описаний формулою:

$$x_{qi}^i \begin{cases} x_{qi}^i + n_q^i Rand\{0,1\}, if\ f^x\left(x_q^i\right) > f^x\left(x_q^{i-1}\right) \\ x_q^{i-1}, \text{ інше} \end{cases}, \qquad (2.3)$$

де $Rand\{0,1\}$ – рівномірно розподілене в інтервалі $(0,1)$ випадкове число.

По суті дана функція є локальним випадковим пошуком з поверненням, яке було запроваджено [15] Л. Растригіним. По суті дана процедура проводить «зондування» функції навколо точки, також тут є можливість використання будь-якого іншого алгоритму випадкового пошуку.

На основі зондування функції щільності за допомогою індивідуального руху можна реалізувати інстинктивно-колективного руху в напрямку зростання даної функції за допомогою функції:

$$x_q^i = x_q^{i-1} + \frac{(\sum_{p-1}^{Q}(x_p^i - x_p^{i-1}))(f^x(x_q^i) - f^x(x_q^{i-1}))}{\sum_{p-1}^{Q}(f^x(x_p^i) - f^x(x_p^{i-1}))}. \qquad (2.4)$$

Вже на цьому етапі з урахуванням успішності рухів кожної з риб відбувається збалансоване усереднення окремих рухів. При колективно-вольовому русі, коли, наприклад, всі агенти-риби зграї «збираються» до зваженого центру ваги коли нахил доходить до крайності зваженого центру, та «тікають» під час руху популяції в неправильному напрямку.

Враховуючи зважений центр ваги зграї риби:

$$Bar^i = \frac{\sum_{p-1}^{Q} x_p^i w_p^i}{\sum_{p-1}^{Q} w_p^i}, \tag{2.5}$$

можна записати переміщення даною формулою:

$$x_q^i \begin{cases} x_q^i - n_q^i Rand\{0,1\} \frac{x_q^{i-1} - Bar^{i-1}}{\|x_q^{i-1} - Bar^{i-1}\|}, & \text{якщо } \sum_{p-1}^{Q} w_p^i > \sum_{p-1}^{Q} w_p^i \\ x_q^i + n_q^i Rand\{0,1\} \frac{x_q^{i-1} - Bar^{i-1}}{\|x_q^{i-1} - Bar^{i-1}\|}, & \text{якщо } \sum_{p-1}^{Q} w_p^i > \sum_{p-1}^{Q} w_p^i \end{cases}. \tag{2.6}$$

Також, для збільшення ефективності алгоритму FSS можна ввести додатковий оператор розмноження, який буде дозволяти створення нових особин популяції, які б могли мати покращені характеристики в порівнянні з іншими представниками зграї. Для цього використовується ідея еволюційних операцій, серед яких з ефективності обчислюваної точки зору та ефективної достовірності знаходження екстремумів можна використати симплексний метод та його модифікації.

Можна сформувати зграю, яка буде містити деяку кількість $Q = n + 1$ риб-агентів, але дана популяція буде залишатись незмінною під час процесу пошуку, тобто, популяція буде генеруватись випадковим чином $x_1^0, x_2^0, \dots, x_Q^0$. В даній популяції буде знайдене «найгірша» x_{qworst}^0, яка буде мати найменшу масу x_{qmin}^0, також, буде знайдена «найкраща» рибка в популяції x_{qbest}^0 та з кращою вагою w_{qmax}^0. За основну операцію симплекс-руху беремо

відображення через центр тяжіння рибок, який не буде включати «найгірші», та записується у вигляді формули:

$$\overline{x}^0 = \frac{1}{n}\sum_{q-1}^{1}(x_q^0 - x_{qworst}^0). \quad (2.7)$$

В результаті даної формули виводиться нова рибка за формулою:

$$x_q^{1*} = \overline{x}^0 + \alpha(\overline{x}^0 - x_{qworst}^0), \quad (2.8)$$

де описується найгірший агент x_{qworst}^0.

Дані формули генерують нову популяцію виду $x_1^1, x_2^1, \dots, x_Q^1$.

Ось $0{,}5 \le \alpha \le 2$ – параметр, який має контролювати форму школи (симплекс в процесі оптимізації). В тому випадку, коли $\alpha = 1$, відображення симплекса через центр тяжіння реалізується у випадку, коли $f^x\left(x_q^{1*}\right) > f^x(x_{best}^0)$ прийняте $\alpha = 2$, тобто зграя «розтягується» у правильному сприятливому напрямку, але якщо $f^x\left(x_q^{1*}\right) > f^x(x_{worst}^0)$, $\alpha = 1$, то симплекс стискається відносно невдало.

Рух рибок може бути описаний за допомогою:

$$\begin{cases} \overline{x}^{i-1} = \frac{1}{n}\sum_{q-1}^{Q}\left(x_q^{i-1} - x_{qworst}^{i-1}\right) \\ \overline{x}_q^i = \overline{x}_q^{i-1} + \alpha(x_q^{i-1} - x_{qworst}^{i-1}) \end{cases}. \quad (2.9)$$

По суті, це алгоритм оптимізації [16] Недлера-Міда, де в процесі пошуку екстремумів найгірші рибки з меншою вагою видаляються та утворюються агенти з більшою вагою. При знаходженні будь-якого екстремуму з вихідної вибірки спостереження, де включаються екстремуми, які розташовані виключаються. Після цього видалення процедура повторюється допоки не будуть знайдені всі екстремуми-центроїди.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

3.1 Вибір мови програмування

Для розробки системи мною було обрано мову програмування C#, чому посприяла велика кількість факторів, які будуть послідовно надані мною в наступних пунктах.

3.1.1 Мова C#

Мабуть, з усіх мов програмування, мова C# [17-20] одна з найкращих. Ця мова досить проста для вивчення, об’єктно орієнтована, багатопарадигменна та універсальна.

C# – це об’єктно-орієнтована та орієнтована на компоненти мова програмування. Вона була розроблена компанією Microsoft у 2000 році. Вона була розроблена як одна з мов програмування для розробки на платформі Microsoft .NET Framework. Основними методами даної мови є безпека типів, ефективність, та підтримка розробки гнучких інтерфейсів та інтегроване програмування. Одна з найбільших переваг мови C# є те, що ця мова має високий рівень абстракції, що дозволяє програмістам за допомогою легко зрозумілого програмного коду. Крім того, мова C# підтримує автоматичне управління пам’яттю, що можуть виникнути під час розробки складних систем.

Станом на січень 2023 року C# зайняла 5-е місце в індексі популярності мов програмування TIOBE, відразу після мов Python, C, C++, Java. Також вже в другому індексі популярності мов програмування PYPL на січень 2023 мова зайняла 4 місце, відразу після мов Python, Java, JavaScript.

На рисунку 3.1 наглядно можна побачити те, які мови програмування наряду з C# займають перші 10 місць за даними TIOBE [19].

Янв 2023	Янв 2022	Изменение	Язык программирования	Рейтинг	Изменение
1	1		Python	16.36%	+2.78%
2	2		C	16.26%	+3.82%
3	4	^	C++	12.91%	+4.62%
4	3	v	Java	12.21%	+1.55%
5	5		C#	5.73%	+0.05%
6	6		Visual Basic	4.64%	-0.10%
7	7		JavaScript	2.87%	+0.78%
8	9	^	SQL	2.50%	+0.70%
9	8	v	Язык ассемблера	1.60%	-0.25%
10	11	^	PHP	1.39%	-0.00%

Рисунок 3.1 – Популярність мов програмування за даними TIOBE

На рисунку 3.2 наглядно можна побачити те, які мови програмування наряду з C# займають перші 10 місць за даними PYPL [20].

Место	Изменение	Язык программирования	Доля	Тенденция
1		Python	27.93 %	-0.9 %
2		Java	16.78 %	-1.3 %
3		JavaScript	9.63 %	+0.5 %
4	^	C#	6.99 %	-0.3 %
5	v	C/C++	6.9 %	-0.5 %
6		PHP	5.29 %	-0.8 %
7		R	4.03 %	-0.2 %
8	^^^	TypeScript	2.79 %	+1.0 %
9		Swift	2.23 %	+0.3 %
10	vv	Objective-C	2.2 %	-0.1 %

Рисунок 3.2 – Популярність мов програмування за даними PYPL

Так як і інші мови програмування, C# використовується для створення різних програм по типу свого призначення. C# – це об’єктно-орієнтована мова програмування, яка створювалась в період з 1998 року по 2002 рік інженерами Microsoft під керівництвом Андерса Хейлсберга та Скота Вільтаумона. Дана мова входить в сім’ю C-подібних мов з синтаксисом, наближеним до Java та

C++ та має такі основні особистості як статистична типізація, поліморфізм, навантаження операторів, також в даній мові доступна делегація, події, узагальнені типи та анонімні функції.

На рисунку 3.3 можна наглядно розглянути динаміку популярності мов програмування за останні 17 років за даними PYPL [20].

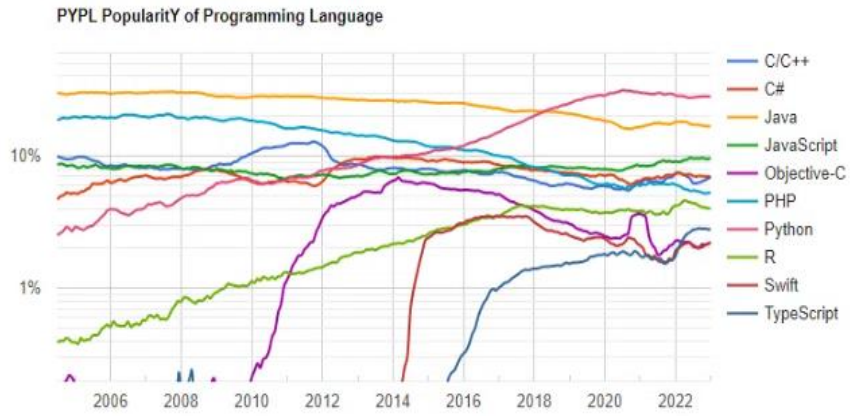

Рисунок 3.3 – Динаміка популярності мов програмування по даним PYPL

3.1.2 Переваги та недоліки мови C#

Переваги мови C# [21]:

– об’єктно-орієнтована мова: дана мова розроблена як об’єктно-орієнтована мова програмування, що допомагає в розробці багатомодульних програм з високою модульністю та масштабованістю;

– велика спільнота розробників: C# має велику спільноту розробників тому, що має велику документацію та спільноту розробників;

– висока продуктивність: C# компілюється до проміжного коду .NET, що забезпечує високу продуктивність;

– безпека: В мови C# є вбудовані механізми безпеки, такі як контроль доступу, забезпечення безпеки типів даних;

– кросплатформенність: додатки, які написані на мові C#, можуть запуститись як на Windows, Linux, так і на macOS.

Недоліки мови C#:

– обмежена підтримка платформ: мова C# підтримувана тільки на платформі .NET, та її не можна використовувати для програмування в окремих середовищах;

– низький рівень контролю пам’яті: мова C# має автоматичне управління пам’яттю, що може забезпечити більш високий рівень безпеки, але обмежує можливість контролювання пам’яті;

– швидкість виконання: хоча C# – це високопродуктивна мова програмування, вона в деяких випадках може бути менш продуктивною, ніж мови, які компілюються відразу в машинний код;

– перевантаження операторів: в мові C# оператори можна перевантажувати, що спрощує код та робить його більш зрозумілим, але часто це приводить до неочікуваної роботи програми;

– висока складність: C# є мовою зі складним синтаксисом, що робить її використання складним для початківців.

3.1.3 Підсумок щодо вибору мови

Узагальнивши всі плюси та мінуси щодо використання мови C# в даному проєкті, було зроблено висновок що дана мова відповідає вимогам майбутньої програми і програма може бути написана швидко та зрозумілою мовою.

3.2 Технології та бібліотеки

Далі описані бібліотеки та технології, які будуть використанні в розробці програми.

3.2.1 .NET Framework 4.7.2

.NET Framework [22-24] – це платформа для розробки програмного забезпечення, яка розроблена компанією Microsoft для створення та запуску програм на операційній системі Windows Даний фреймворк складається з інструментів для розробника, мов програмування та бібліотек для створення настільних та веб-додатків. Фреймворк .NET вперше був випущений в 2002 році, перша версія називалась .NET Framework 1.0. З того часу платформа .NET Framework пройшла довгий шлях розвитку та поточною версією є .NET Framework 4.8.1. Проєкти на основі даної платформи використовуються як для створення програм на основі форм (Windows Forms), так і ігор та веб-застосунків.

Існують різні варіації реалізації програм на платформі .NET. Кожна з цих реалізацій позволяє виконувати .NET-код в різних операційних системах: Linux, macOS, Windows, iOS, Android та багато інших.

В даному проєкті була використана реалізація .NET Framework, яка являється оригінальною реалізацією .NET. вона підтримує роботу вебсайтів, служб, бібліотек класів, настільних програм для операційної системи Windows, тощо. Окрім даної реалізації є такі реалізації, як:

– .NET Core – це міжплатформенна реалізація для запуску вебсайтів, служб, консольних програм в Windows, OS та macOS. Також .NET Core є розробкою з відкритим кодом на GitHub;

– Xamarin – це реалізація .NET для написання та запуску програм на всіх мобільних операційних системах, наприклад, iOS, Android, тощо.

3.2.2 Windows Forms

Windows Forms [25] – це набір керованих бібліотек .NET Framework, які призначені для розробки багатофункціональних клієнтських програм. Це

графічний API для відображення даних і керування взаємодією користувачів з спрощеним розгортанням та кращою безпекою в клієнтських програмах.

Windows Forms пропонує розширену клієнтську бібліотеку, ка забезпечує інтерфейс для доступу до рідних елементів графічного інтерфейсу Windows та графіки з керованого коду. Windows Forms схожа на бібліотеку Microsoft Foundation Class (MFS) у розробці клієнтських програм. Дана технологія надає оболонку, яка складається з набору класів C++ для розробки Windows програм, однак Windows Forms не забезпечує стандартну структуру додатків, як MFS.

3.2.3 Regular Expressions

Регулярні вирази [26-28], насамперед, являються частиною технічної області програми і широко використовуються багатьма програмістами в різних програмах. У C# регулярний вираз це шаблон, який використовується для аналізу даних та перевірки відповідності введеного тексту чи числових значень заданому шаблону. В мові C# регулярні вирази називають Regex. .NET Framework надає деякий механізм регулярних виразів, який надає можливість зіставляти шаблони, які можуть складатися з будь-яких символьних літералів, операторів, або конструкторів. Мова C# надає клас під назвою Regex, який знаходиться в просторі імен System.Text.RegularExpression.

Даний клас виконує дві дії:

- розбір введеного тексту для шаблону регулярного виразу;
- визначення шаблону регулярного виразу в поданому тексті.

3.2.4 ZedGraph

ZedGraph – це бібліотека графічної візуалізації даних для мов програмування .NET, що дозволяє створювати різноманітні графіки і діаграми,

такі як лінійні графіки, стовбчасті діаграми тощо. За допомогою ZedGraph можна створювати професійні графіки для відображення в багатьох областях, таких як фінанси, медецина, інженерія та інші.

ZedGraph ожна використовувати в програмах, які написані на мовах програмування, таких як C#, Visual Basic та інші. Бібліотека має широкі можлиості налаштування графіків, включаючи розмір, кольори, шрифти, маркери, легенди та інше. ZedGraph також підтримує відображення множини даних на одному графіку, що дозволяє порівнювати дані між собою.

В загальному, ZedGraph є потужним інструментом для візуалізації даних в програмах .NET, який дозволяє створювати якісні графіки та діаграми з різноманітною настройкою та можливостями інтерактивної роботи з графіками. Також, бібліотека має відкритий код та безкоштовна для використання [29–31].

3.2.5 Net Charting

Net Charting [32] – це бібліотека компонентів графічного інтерфейсу користувача для створення динамічних діаграм та графіків на платформі.NET. Бібліотека містить набор інструментів для відображення різноманітних типів даних у вигляді лінійних графіків, кругових діаграм, стовпчикових діаграм та інших типів графіки. Net Charting надає можливість налаштувати вигляд та поведінку діаграм з використанням різних настройок, також вона має підтримку анімації та взаємодії з користувачем, що дозволяє створювати динамічні та інтерактивні діаграми. Також Net Charting дозволяє підключати бази даних до графіку, тобто, проводити роботу з різними джерелами інформації. Бібліотека також має можливість відображення даних в реальному часі з використанням AJAX-технологій, що робить її ідеальним вибором для створення моніторингових та аналітичних додатків.

Net Charting є потужним інструментом для створення професійних та ефективних діаграм на платформі .NET. Завдяки широкому набору функціональних можливостей та легкою інтеграцією Net Charting став популярним рішенням серед розробників програмного забезпечення, які потребують відображення даних у вигляді графіків.

3.3 Програмна реалізація

Після дослідження матеріалів та обраного способу реалізації я можу перейти до головної фази розробки програмного застосунку. Для створення легко настоюваної, гнучкої системи для пошуку екстремумів багатовимірних функцій, яка основана роботі алгоритмів Fish Swarm School (FSS) та Genetic Algorithm (GA). В даному розділі мною буде описано роботу даних алгоритмів та програмну реалізацію, також буде розглянута порівнювальна схема цих алгоритмів та взаємодія функціональних блоків програми.

3.4 Розробка програмних компонентів генетичного алгоритму (GA)

Далі буде описано основні класи, з яких складається генетичний алгоритм (GA).

3.4.1 Клас BaseSpecies

BaseSpecies – це абстрактний клас, який містить в собі статичні методи, які використовуються для схрещення та мутації. Також, дані методи повинні бути типу protected.

Клас *BaseSpecies<TSpeciesFunc>* – це клас, всі види популяції повинні бути похідними цього класу. *TSpeciesFunc* – це конкретний тип для даного класу та даний тип є похідним типом з класу *BaseSpecies<TSpeciesFunc>*.

Розгляд методів даного класу можна почати з методів, які будуть реалізовані в похідних класах.

Першим з даних методів є метод *abstract public void TestChromosomes()*. Даний метод проводить перевірку всіх хромосом на те, чи попали хромосоми в задані інтервали функції. Даний метод реалізує булеву статичну змінну Dead, яка показує чи особина мертва або жива. Дана змінна повертає параметр m_Dead, який повертає або значення true, якщо значення хромосом не підпадають в обмеження функції, в іншому випадку він повертає значення *true*.

Наступним методом є метод, в якому проводиться створення нових видів(коли одна особина одного виду схрещується з іншою особою цього ж виду) *abstract public TSpeciesfunc Cross(TSpeciesfunc Species)*. Самий сенс схрещування полягає в тому, що беруться дві особини виду з різними хромосомами, в обох хромосомах особин вибирається точка розриву та з лівої частини одної хромосоми та правої частини іншої створюється нова хромосома вже нової особини виду. Також, клас *BaseSpecies* має публічні статичні методи, які потрібні для схрещування хромосом трьох типів, а саме: *Double, Int64, Int32*. Перші два метода мають дві точки перетину, при цьому, хромосоми спочатку схрещуються побітово без урахування знаку, а потім береться випадковим чином знак однієї з хромосом. Для забезпечення правильної роботи методу треба спочатку перетворити хромосому типу Double перетворити в *Int64* для того, щоб їх можна було без проблем схрещувати побітово. Вже після схрещування, коли числа вже мають тип *Int64*, знаки не враховуються. Потім число переводиться назад у тип *Double* і знак випадково береться від однієї з двох хромосом. Програмна реалізація схрещування хромосом типу *Double* зображена на рисунку 3.4.

```
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;
}
```

Рисунок 3.4 – Схрещування хромосом типу Double

Можна розглянути процес схрещування *Int64*. Спочатку хромосоми схрещуються без врахування їхнього знаку, а потім обирається знак однієї з хромосом. Зокрема, знак обраної хромосоми порівнюється зі значенням результату, і якщо знаки не співпадають, знак результату змінюється на протилежний. Код даного схрещування показано на рисунку 3.5.

```
Ссылок: 2
static protected Int64 BitCross(Int64 x, Int64 y)
{
    // Число бит, оставшееся слева от точки пересечения хромосо
    int Count = m_Rnd.Next(62) + 1;
    Int64 mask = ~0;
    mask = mask << (64 - Count);
    return (x & mask) | (y & ~mask);
}

Ссылок: 0
static protected Int64 Cross(Int64 x, Int64 y)
{
    // Число бит, оставшееся слева от точки пересечения хромосом
    Int64 res = BitCross(x, y);
    if (m_Rnd.Next() % 2 == 0)
    {
        if (x * res < 0)
        {
            res = -res;
        }
    }
    else
    {
        if (y * res < 0)
        {
            res = -res;
        }
    }
    return res;
}
```

Рисунок 3.5 – Програмна реалізація методу схрещення типу *Int64*

Можна також розглянути наступний метод з назвою *static protected double Mutation(double val)*. Даний метод можна порівняти з методом схрещування, оскільки він також змінює хромосому. Теоретично, під час мутації можуть відбуватись будь-які зміни, але в моїй реалізації мутація може

змінювати лише один біт в слові. Мутації зазвичай відбуваються дуже рідко, зазвичай, їх ймовірність становить 1-10%, але іноді може бути ймовірність аж до 30-40%. Якщо робота ведеться з дрібними числами, то краще використовувати генератор випадкових чисел, який надає значення в потрібному інтервалі, аніж спеціальну функцію мутації. У цьому випадку для дрібних чисел використаний генератор випадкових чисел, а для цілих хромосом треба використовувати функцію, код якої неведений на рисунку 3.6 нижче.

```
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>
/// Мутация типа Int64
/// </summary>
/// <param name="val">Мутируемое значение</param>
/// <returns>Промутирующее значение</returns>
Ссылок: 0
static protected Int64 Mutation(Int64 val)
{
    Int64 mask = 1;
    mask <<= m_Rnd.Next(63);
    return val ^ mask;
}
```

Рисунок 3.6 – Програмна реалізація методу мутації

3.4.2 Клас Population

Клас Population – це публічний клас, який має generic-параметр «TSpeciesfunc», який повинен бути похідним від *BaseSpecies<TSpeciesfunc>*. Цей клас є класом популяції, де проживають, розмножуються та вмирають різні види особин. Можна додавати власні види до цього класу, або точніше до масиву видів, який знаходиться в даному класі.

Перед використанням даного алгоритму треба налаштувати певні властивості даного алгоритму:

– *Maxsize(Int32)* – це максимальна кількість видів, яку може мати дана популяція. Ця кількість обмежується після схрещування. За замовчуванням дана властивість має значення 500 особин;

– *Cross-Possibility(double)* – це ймовірність схрещування. Значення цієї властивості повинне бути в межах (від 0,0 до 1,0). Якщо це не так, то виникає виняток *ArgumentOfRangeException*. За замовчуванням дана властивість встановлена на значення 0,95;

– *MutationPossibility(double)* – це ймовірність мутації. Значення даної властивості повинно бути в межах від 0,0 до 1,0. Якщо це не так, то виникає виняток *ArgumentOfRangeException*.

Можна розглянути публічні функції, які потрібно викликати.

Для додавання нового виду в популяцію треба викликати функцію *public void Add(TSpeciesfunc species)*. Перед запуском алгоритму необхідно додати хоча б два види вручну, а кількість видів в популяції може також бути менше, ніж максимально допустиме значення, яке характеризується змінною *MaxSize*. Якщо після схрещування розмір популяції менше значення *MaxSize*, то найгірші види не видаляються навіть якщо вони мертві.

Для отримання наступної популяції треба викликати метод *GetNextGen()*. Даний метод може зайняти велику кількість часу, тому краще треба викликати його з окремого потоку. Дана функція генерує нову популяцію на основі поточної популяції та виконує всі необхідні операції, такі як мутація та схрещування. Робота даного методу показана в фрагменті коду на рисунку 3.7.

```
Ссылок: 3
virtual public void GetNextGen()
{
    if (m_Generation == 0 && m_Species.Count == 0)
    {
        throw new ZeroSizePopulationException();
    }
    //спаривание
    Cross();
    foreach (TSpeciesfunc species in m_Species)
    {
        // Если будет мутация с учетом вероятности
        if (m_Rnd.NextDouble() <= m_MutationPossibility)
        {
            species.Mutation();
        }
        species.TestChromosomes();
    }
    // Отбор самых живучих видов
    m_Species.Sort();
    Selection();
    m_Generation++;
}
```

Рисунок 3.7 – Реалізація методу *GetNextGen()*

Після застосування даного методу треба застосувати метод схрещування двох особин *protected void Cross()*. Під час схрещування проводиться перебір усіх видів та кожен вид випадково схрещується з іншими видами з заданою ймовірністю. Після цього проводиться мутація кожного виду та тестуються його хромосоми. Наступним кроком є сортування видів за значенням їх цільової функції за допомогою методу Sort, який визначено в класі *BaseSpecies*. Реалізація методу схрещування зображена на рисунку 3.8. Реалізація методу відбору популяції *Selection()* зображена на рисунку 3.9.

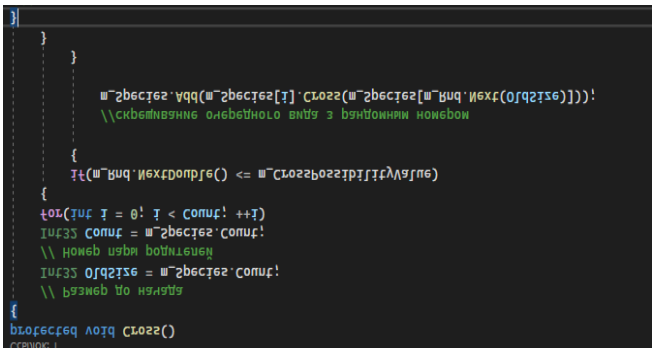

Рисунок 3.8 – Програмна реалізація методу *Cross*

```
Ссылок: 1
protected void Selection()
{
    // кол-во видов для удаления
    Int32 Count = m_Species.Count - m_Maxsize;
    for (Int32 i = 0; i < Count; ++i)
    {
        m_Species.RemoveAt(m_Species.Count - 1);
    }
}
```

Рисунок 3.9 – Програмна реалізація методу *Selection()*

3.4.3 Клас BaseDoubleSpecies

Базовий клас *BaseSpecies* було розроблено з метою максимальної абстракції від конкретних реалізацій особин та представлення хромосом в

будь-якій формі, включаючи дробові та цілі числа, масиви та екземпляри інших класів.

Однак, застосовано генетичний алгоритм (GA) для мінімізації математичної функції, яка залежна від деякої кількості змінних дрібного типу. У цьому випадку, хромосоми зручніше представляти у вигляді масиву дрібних чисел, де кожен елемент масиву відповідає одній хромосомі. При цьому, розмір масиву буде постійним для всіх особин на протязі роботи алгоритму.

Ось тільки для цього було створена клас *BaseDoubleSpecies*. Цей клас має досить заплутане оголошення, проте завдяки йому можна суттєво зменшити код, необхідний для реалізації алгоритму. Оголошення даного класу зображено на рисунку 3.10.

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

Рисунок 3.10 – Оголошення класу *BaseDoubleSpecies*

Тут, так як і раніше, *TSpeciesfunc* використана для позначення певного типу особин.

Клас *BaseDoubleSpecies* – це абстрактний клас, який реалізує всі методи, що необхідні для класу особин, і додає свій абстрактний метод, в якому відбувається розрахунок цільової функції. Цей метод називається *FinalFuncCalculation()* і повертає значення типу double. Щоб уникнути повторного обчислення значення цільової функції при порівнянні особин, вона обчислюється лише один раз у конструкторі особини, а результат зберігається в змінній *m_FuncVal*.

Клас *BaseDoubleSpecies* містить також два масиви: *m_Chromosomes*, який містить хромосоми, та *m_Intervals*, який є статичним масивом класу *Interval* і задає інтервали між кожною хромосомою. Для доступу до даного масиву передбачено властивість *Chromosomes*. Розмір масиву *m_Intervals*

повинен завжди дорівнювати розміру масиву *m_Chromosomes*. Крім того. При створенні особини з випадковими значеннями хромосом, розмір масиву m_Intervals визначає кількість генів в кожній хромосомі. Якщо якась хромосома потрапляє в свій інтервал, то особина вважається мертвою. Цю властивість m_Intervals потрібно встановлювати до заповнення популяції.

Клас *BaseDoubleSpecies* наслідує функціонал базового класу *BaseSpecies*, та також додатково реалізовує етапи генетичного алгоритму, які відсутні у базовому класі. Клас *BaseDoubleSpecies* містить два конструктори:

– *public BaseDoubleSpecies()* – це конструктор, який створює нову особину з випадковою хромосомою, яка може бути корисна для початкового заповнення популяції;

– *public BaseDoubleSpecies(double[] chromosomes)* – це конструктор, який використовується всередині класу *BaseDoubleSpecies* для виконання операцій мутації та схрещування.

3.5 Розробка програмних компонентів алгоритму зграї риб (FSS)

Далі будуть описані основні класи алгоритму FSS та робота, виконувана класами.

3.5.1 Клас ThePointOfFish

Клас *ThePointOfFish* – це клас, який містить в собі статистичні методи, робота яких полягає в знаходженні евклідової відстані між агентами алгоритму, та методи, які виконують основні операції над агентами.

Величини, які відповідають за стан агентів – це структурні типи *n*-мірної точки. Для даних величин є такі операції як: додавання, або віднімання двох точок; додавання або віднімання одної точки та числа; множення або ділення одної точки та числа та операції порівняння точок. Дані величини прийняті як векторі типи даних.

Детальний розбір деяких методів класу *ThePointOfFish*:

– *ThePointOfFish(params double[] cords)* – це конструктор, в який передаються координати агента;

– *ThePointOfFish(int SizeOfDimention)* – це конструктор, в який передається кількість агентів в популяції;

– *GetCoords()* – це метод, в який йде передача координат агентів, які храняться в векторі, який оголошено в конструкторі *ThePointOfFish(int SizeOfDimention)*;

– *EuclidDistance(ThePointOfFish p1, ThePointOfFish p2)* – це статистичний метод, який вичисляє евклідову відстань між агентами FSS, та пізніше, на стадії плавання він також знадобиться;

– *ThePointOfFish operator -(ThePointOfFish p1, ThePointOfFish p2)* – це статистичний метод, в якому виконується операція віднімання векторів, які являють собою агентів;

– *ThePointOfFish operator +(ThePointOfFish p1, double number)* – це статистичний метод, який виконує операцію додавання векторів, які являють собою особин зграї (риб);

– *ThePointOfFish operator *(ThePointOfFish p1, double factor)* – це статичний метод, який виконує собою операцію множення вектора(рибки) на змінну *factor*;

– *InRegion(ThePointOfFish loverBound, ThePointOfFish higherBound)* – це метод, який, чкщо особина знаходиться в межі пошуку буде повертати true, та якщо особина не лежить в межах пошуку, то метод повертає *false*.

3.5.2 Клас FishUniversalWork

ThePointOfFish – це основний клас програми, який реалізує основну логіку алгоритму FSS. В даному класі реалізовані ті методи, в яких реалізовано всі стадії переміщення рибок-агентів, пошук максимального значення дельта-

функції, годування рибок, пошуку сумарної ваги всіх рибок в зграї та інші параметри.

Агенти в даному класі пересуваються поітераційно та в кожній ітерації виконуються оператори, які відповідають за плавання (забезпечення руху агентів в межах акваріума), та годування (фіксація дослідження одної, чи другої області акваріума). Тим самим можна перейти до введення параметрів акваріума (області пошуку) та агентів. Даним параметрам будуть присвоєні наступні змінні:

– *PopSize* – це змінна, яка відповідає за кількість агентів в популяції;

– *IterationCount* – це змінна, яка відповідає за кількість ітерацій;

– *lowerBoundPoint, higherBoundPoint* – це верхня та нижня границя пошуку;

– *individStepStart, individStepFinal* – це межі кроків переміщення агентів в індивідуальному плаванні;

– *weightScale* – змінна, яка відповідає за максимальну вагу даного агента.

Дані параметри вводяться користувачем програмного застосунку на формі. За допомогою цих даних регулюється точність та час роботи алгоритму.

Ось саме дві змінні, які характеризують агентів:

– *swimStatePoint* – це позиція агента в акваріумі;

– *weight* – це вага агента.

Метод *InitializeData()* відповідний за ініціалізацію агентів та наведений на рисунку 3.11 нижче.

Положення кожного агенту популяції вибирається випадково для кожного агенту.

Далі мною будуть наведені основні дані, які потрібні для ітерацій:

– *individStep* – це значення індивідуального кроку агенту під час даної ітерації;

– *indivieDeltaFitnessMax* – це максимальне значення фітнес-функції;

– *instinctAverage* – це середнє зважене після інстинктивної стадії;

– *instinctSumWeightOld* – це старе значення ваги риб в минулій ітерації;

– *willStep* – це шаг вольового переміщення, який зависить від індивідуального кроку переміщення;

– *barycenter* – це центр тяжіння риб, який буде використаний в вольовому переміщенні;

– *individDeltaPoint* – це зміщення агенту після індивідуальної стадії плавання;

– *individDeltaFitnes* – це зміна фітнес-функції після здійснення індивідуального плавання;

weight – це вага одного агенту функції, який за замовчуванням рівний *weightScale/2*.

```
Ссылок: 1
public static void InitializeData(TheFuncOfUser.FuncTemplate _function, int _dimensionSize, out FishUniversalWork[] _fishes, int _popsize, int _iterationCount,
    ThePointOfFish _lowerBound, ThePointOfFish _higherBound, double _individStepStart = 0.5, double _individStepFinish = 0.1, double _weightScale = 10)
{
    funcTemplate = _function;
    dimensionSize = _dimensionSize;
    popSize = _popsize;
    iterationCount = _iterationCount;
    lowerBoundPoint = _lowerBound;
    higherBoundPoint = _higherBound;
    individStepStart = _individStepStart;
    individStepFinal = _individStepFinish;
    individStep = individStepStart;
    weightScale = _weightScale;
    instinctSumWeightOld = popSize * weightScale / 2;
    _fishes = new FishUniversalWork[popSize];
    for (int fishIndex = 0; fishIndex < _fishes.Length; fishIndex++)
    {
        _fishes[fishIndex] = new FishUniversalWork();

        for (int swimState = 0; swimState < 4; swimState++)
            _fishes[fishIndex].swimStatePoint[swimState] = new ThePointOfFish(dimensionSize);
        for (int dimensionIndex = 0; dimensionIndex < dimensionSize; dimensionIndex++)
            _fishes[fishIndex].swimStatePoint[0][dimensionIndex] = lowerBoundPoint[dimensionIndex] +
                (higherBoundPoint[dimensionIndex] - lowerBoundPoint[dimensionIndex]) * randGen.NextDouble();
    }

}
```

Рисунок 3.11 – Програмна реалізація методу *InitializeData()*

Також, індивідуальні дані повинні десь зберігатися, отже було реалізовано масив *swimStatePoint = new ThePointOfFish[4]*, де зберігається під значеннями масиву:

0 – це початок індивідуального плавання;

1 – це місцеположення після індивідуального плавання;

2 – це місцеположення після інстинктивного плавання;

3 – це кінцева позиція ітерації.

В першу чергу алгоритмом проходить ініціалізація всіх вхідних даних та ініціалізація популяції. Далі проводиться робота основного циклу алгоритму, де агенти шукають шлях до місцезнаходження їжі. В даному випадку, мною за критерій зупинки було взято кінцеву кількість ітерацій.

Потім відбувається стадія індивідуального переміщення агентів. Вона реалізована в методі *IndividSwim()*. Під час роботи даного метода, агентам дається можливість пересуватися, та кожен агент популяції шукає коло себе їжу (краще рішення функції). Якщо агент знаходить краще значення, то краще значення записується, а якщо краще значення не знаходиться, то вважається що агент не проводив пошук. Після завершення пошуку проводиться запис кращої ваги риби-агенту.

Після індивідуального переміщення, агенти здійснюють інстинктивно-колективно стадію переміщення. Дана стадія переміщення реалізована в методі *InstinctSwim()*. Для всієї популяції агентів проходить вираховування загального кроку переміщення (*swimStatePoint*). Популяція впливає на кожного агенту, та має загальний вплив на окремих агентів, який пропорційний успіхам даного агенту в індивідуальній стадії переміщення. Після цього вся популяція агентів переміщується на вирахувану величину *instinctAverage*.

Потім проводиться вирахування центру тяжіння популяції, яке відображене в методі *SearchBarycentre(Fishuniversal[] fishes)*, де йде розрахування на основі положення популяції та ваги кожного агенту популяції.

Останньою частиною, точніше останньою стадією алгоритму є колективно-вольове переміщення, яке реалізовано методом *CollectiveSwim()*.

В даному методі потрібно узнати як сильно була змінена вага зграї в порівнянні з минулою ітерацією. Якщо вага збільшилась, то зграя близиться до пошуку мінімуму функції.

А якщо вага зменшується, то треба змінити напрямок руху зграї, що значить те, що функція зшукається не в тому місці.

Також, величина *swimStatePoint[3]* буде відповідати за крок вольового зміщення, а оператор *euclidD* вичислює відстані між двома точками в евклідовому просторі.

3.6 Дослідження ефективності алгоритму FSS

При дослідженні впливу розміру популяції щодо часу виконання алгоритму було здійснене на прикладі пошуку екстремуму функції $y = 0{,}2 * * (x^3 + z^3) + 20$.

Всі значення досліду було представлено на таблиці 3.1.

Дослідження впливу розміру зграї щодо часу виконання алгоритму було здійснено на прикладі пошуку екстремуму функції $y = 0{,}2 * (x^3 + z^3) + 20$, дані обчислень представлено в таблиці 3.1, звідси видно що оптимальна кількість агентів це 25 агентів, яка дає найменший час виконанні алгоритму в 0,07 секунд.

Можна відмітити те, що під час дослідження час виконання збільшується при збільшенні кількості агентів, а при зменшенні числа агентів, введеного користувачем, час зменшується.

Також, мною було проведене дослідження щодо визначення швидкості виконання алгоритму залежно від маси агентів, з якого можна зробити висновок, що швидкість виконання залежить також від маси агенту.

Таблиця 3.1 – Вплив розміру популяції щодо часу виконання алгоритму

№ експерименту	**Розмір популяції**	**Час виконання алгоритму, (секунди)**
1	5	0,025
2	10	0,13
3	25	0,07
4	50	0,11
5	75	0,13
6	100	0,26
7	200	0,4
8	500	0,083
9	1000	0,175

3.7 Дослідження генетичного алгоритму

Дослідження проводилося для вивчення впливу розміру популяції на час виконання час виконання генетичного алгоритму (GA) у контексті пошуку екстремуму функції $y = 0{,}2 * (x^3 + z^3) + 20$. Таблиця 3.2 містить найефективніші значення. Після проведення дослідження щодо роботи генетичного алгоритму, мені стало зрозуміло те, що ефективність роботи даного алгоритму залежить від введених параметрів користувачем так як і в алгоритмі FSS, звідки можна зробити висновок, що чим кращі параметри буде введено користувачем, тим точнішим, кращим, та швидшим буде результат роботи алгоритму.

Дане дослідження показало те, що час виконання генетичного алгоритму (GA) у контексті пошуку екстремуму функції залежить від розміру популяції. Чим більша популяція особин, тим алгоритм буде довше обчислювати дані, які були введені користувачем.

Таблиця 3.2 – Вивчення впливу розміру популяції на час виконання час виконання генетичного алгоритму (GA)

№ експерименту	Розмір популяції	Час виконання алгоритму, (секунди)
1	5	0,018
2	10	0,013
3	25	0,063
4	50	0,0335
5	75	0,856
6	100	0,018
7	200	0,557
8	500	0,083
9	1000	0,175

3.8 Розгляд та аналіз інтерфейсу програмного застосунку

Після дослідження роботи алгоритмів FSS та GA можна перейти до розгляду та аналізу інтерфейсу користувача, який був розроблений в програмі.

Інтерфейс с користувача складається з одної форми, яка розбивається на дві сторінки, які зображені на рисунку 3.12, рисунку 3.13.

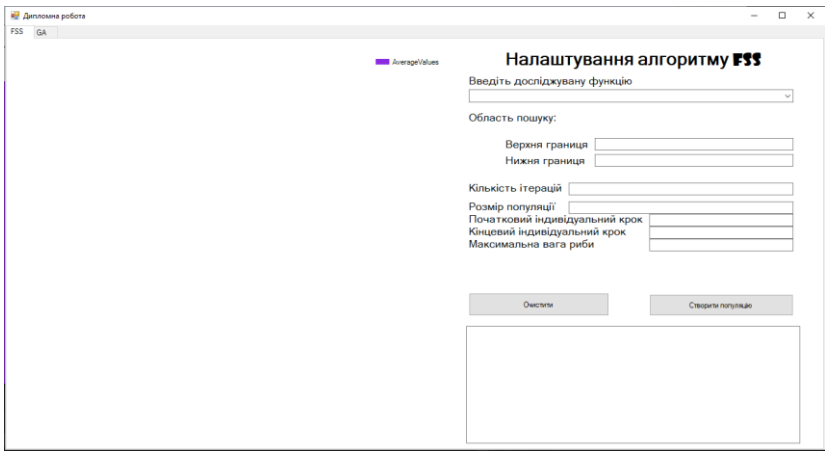

Рисунок 3.12 – Інтерфейс користувача (перша сторінка)

На першій сторінці форми під назвою FSS розташовано графік, на якому виводяться значення мінімумів на кожній з ітерацій алгоритму. Також на даній сторінці реалізовано кнопки та текст бокси.

«Введіть досліджувану функцію» – це графа для введення користувачем досліджуваної функції. «Верхня границя» – це *textBox* для введення користувачем верхньої межи пошуку. «Нижня границя» – це *textBox* для введення користувачем нижньої межи пошуку. «Кількість ітерацій» – це *textBox* для введення користувачем кількості ітерацій алгоритму. «Розмір популяції» – це *textBox* для введення користувачем розміру популяції зграї рибок (агентів). «Початковий індивідуальний крок» – *textBox* для введення користувачем початкового кроку, з яким будуть рухатись агенти на перших ітераціях. «Кінцевий індивідуальний крок» – це *textBox* для введення користувачем кінцевого індивідуального кроку, з яким будуть рухатись агенти на кінцевих ітераціях. «Максимальна вага риби» – це *textBox* для введення користувачем максимальної ваги агенту популяції зграї риби. Також на даній сторінці є textbox для виведення інформації щодо роботи алгоритму.

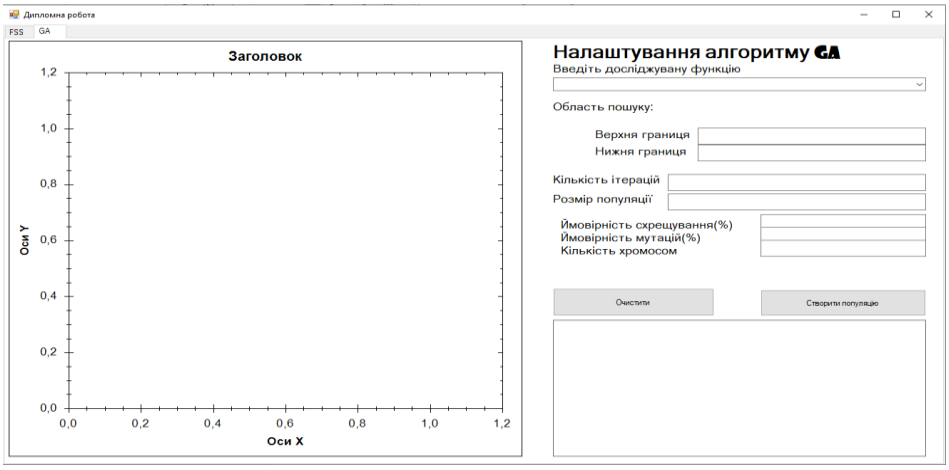

Рисунок 3.13 – Інтерфейс користувача (друга сторінка)

На другій сторінці (рис. 3.13) форми під назвою GA розташовано графік, на якому виводяться значення мінімумів на кожній з ітерацій алгоритму. Також на даній сторінці реалізовано деякі графи та кнопки.

«Введіть досліджувану функцію» – це графа для введення користувачем досліджуваної функції. «Верхня границя» – це *textBox* для введення користувачем верхньої межи пошуку. «Нижня границя» – це *textBox* для введення користувачем нижньої межи пошуку. «Кількість ітерацій» – це *textBox* для введення користувачем кількості ітерацій алгоритму. «Ймовірність схрещування(%)» – це *textBox* для введення користувачем ймовірності проходження схрещування особин. «Ймовірність мутацій(%)» – це *textBox* для введення користувачем ймовірності мутацій хромосом особин після схрещування. «Кількість хромосом» – це *textBox* для введення користувачем значення кількості змінних функції. Також на даній сторінці є textbox для виведення інформації щодо роботи алгоритму.

Також, в інтерфейсі користувача введена перевірка на валідацію введених даних. Якщо користувач неправильно вводить дані, то вся графа становиться червоною та в textbox для виведення інформації виводиться червоним кольором «неправильно введені дані». Це продемонстровані на рисунку 3.14.

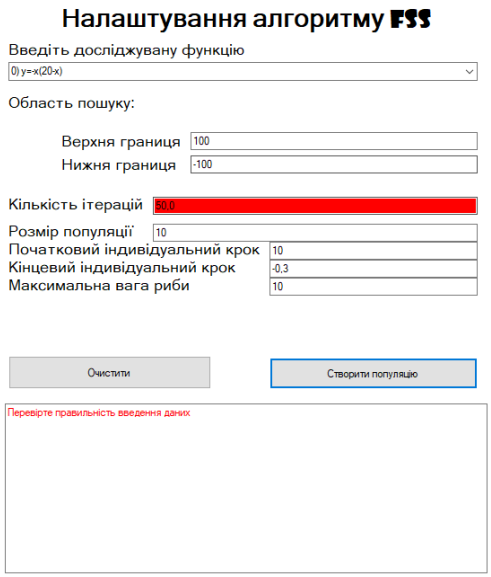

Рисунок 3.14 – Перевірка на валідацію введених даних

Тепер можна перейти до перевірки роботи програми, перевірка роботи програми показана на рисунку 3.15 та рисунку 3.16.

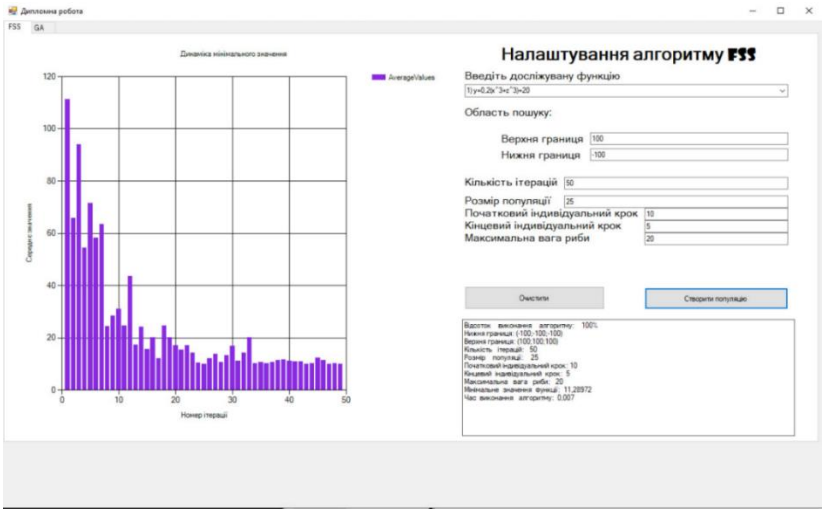

Рисунок 3.15 – Приклад роботи алгоритму зграї риб

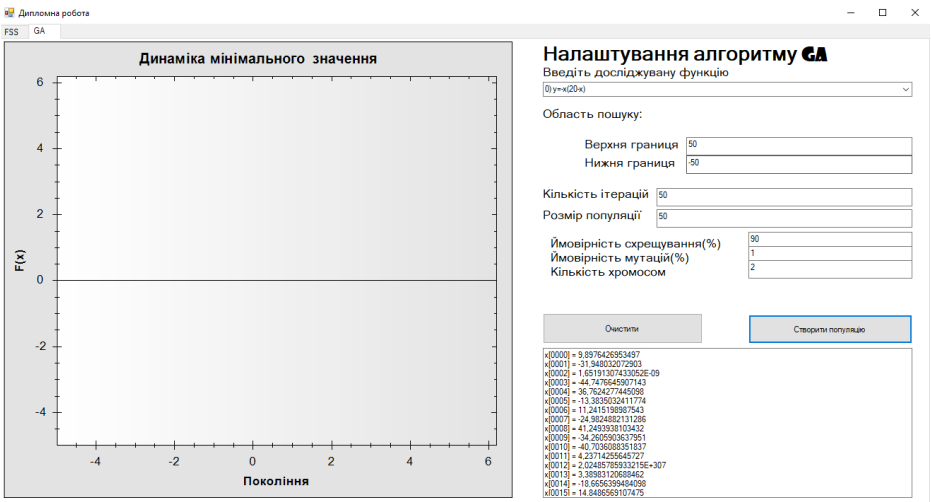

Рисунок 3.16 – Робота генетичного алгоритму

3.9 Рекомендації щодо вдосконалення та застосування застосунку

Розроблений мною програмний застосунок може бути використаний для глобального аналізу. У майбутньому, для розвинення мого програмного застосунку можна додати деякі зміни:

– можливість пошуку не тільки глобального мінімуму, а й максимуму;

– додавання можливості пошуку глобальних екстремумів;

– введення нових алгоритмів(в подальшому в програмний застосунок можна доповнити іншими еволюційними алгоритмами);

ВИСНОВКИ

У рамках кваліфікаційної роботи була розглянута система для пошуку екстремумів багатовимірних функцій на основі еволюційних алгоритмів таких, як генетичний алгоритм та алгоритм зграї риб. Програмний застосунок надає можливість користувачу за допомогою зрозумілого інтерфейсу знаходити глобальні екстремуми багатовимірних функцій.

Робота ділиться на три розділи, в першому з яких було розглянуто різноманітні еволюційні алгоритми та описано їх роботу, такі як: генетичний алгоритм, алгоритм рою частинок, алгоритм випалення, алгоритм поведінки бджіл та алгоритм зграї риб. Також була розглянута схема роботи цих алгоритмів. Також були розглянуті їх переваги та недоліки.

В другому розділі детально були описані ті еволюційні алгоритми, які використані для написання системи. А саме це алгоритми GA та FSS. Повністю був описаний принцип роботи даних алгоритмів та формули, якими вони користуються.

В третьому розділі було проаналізовано програмне середовище, платформи, фреймворки, та технології, які були використанні при написанні програмного застосунку. Потім було розглянуто деякі класи, алгоритмів та детально описані їх атрибути та змінні. Також, в даному розділі було проведено аналіз роботи розробленого застосунку, та надано рекомендації щодо подальшого розвитку проєкту.

Проведені експерименти підтвердили дієвість запропонованих методів достовірної нечіткої кластеризації даних і дозволяють рекомендувати його для використання на практиці для вирішення задач автоматичної кластеризації даних різної природи. Запропонований метод призначений для використання в гібридних системах обчислювального інтелекту і, насамперед, у проблемах навчання штучних нейронних мереж, нейро-нечітких систем, а також у проблемах кластеризації та класифікації.

Отримані результати мають важливе практичне значення для створення систем відновлення та кластеризації даних, які надходять на обробку послідовно, в реальному часі. У ході досліджень отримано такі результати:

– запропонований метод достовірної нечіткої кластеризації даних, який дозволяє вирішувати задачу кластеризації у таблиці «об’єкт-властивість», що містять апріорі невідому кількість даних, а також забезпечує високу швидкодію і простоту чисельної реалізації;

– проведено імітаційне моделювання метода, виконана експериментальна оцінка похибок та якості кластеризації даних в онлайн режимі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Глобальна оптимізація. URL: https://uk.wikipedia.org /wiki/Глобальна_оптимізація (дата звернення 15.04.2023).

2. Shafronenko, A. U., Volkova, V. V., & Bodianskiy, E. V. (2011). АДАПТИВНА КЛАСТЕРИЗАЦІЯ ДАНИХ С ПРОПУЩЕНИМИ ЗНАЧЕННЯМИ. *Radio Electronics, Computer Science, Control*, (2).

3. Жалдак, М. І., & Триус, Ю. В. (2005). Основи теорії і методів оптимізації: навчальний посібник. *Черкаси: Брама-Україна*, *608 с*.

4. Генетичні алгоритми. Ключові поняття і методи реалізації. URL: http://www.znannya.org/?view=ga_general (дата звернення 20.04.2023).

5. Venter, G., & Sobieszczanski-Sobieski, J. (2003). Particle swarm optimization. *AIAA journal*, *41*(8), 1583-1589.

6. Остапчук, В. М. КОМБІНОВАНИЙ АЛГОРИТМ НАВЧАННЯ НЕЙРОННИХ МЕРЕЖ ПРЯМОГО ПОШИРЕННЯ. *Редакційна колегія*, 31 с.

7. Bertsimas, D., & Tsitsiklis, J. (1993). Simulated annealing. *Statistical science*, *8*(1), pp. 10-15.

8. Wasim, M. S., Amjad, M., Habib, S., Abbasi, M. A., Bhatti, A. R., & Muyeen, S. M. (2022). A critical review and performance comparisons of swarm-based optimization algorithms in maximum power point tracking of photovoltaic systems under partial shading conditions. *Energy Reports*, *8*, pp. 4871-4898.

9. Cai, Y. (2010). Artificial fish school algorithm applied in a combinatorial optimization problem. *International Journal of Intelligent Systems and Applications*, *2*(1), p. 37.

10. Mirjalili, S., & Mirjalili, S. (2019). Genetic algorithm. Evolutionary Algorithms and Neural Networks: Theory and Applications, pp. 43-55.

11. Генетичні алгоритми. Ключові поняття і методи реалізації. URL: http://www.znannya.org/?view=ga_general (дата звернення 05.05.2023).

12. Introduction to Genetic Algorithms – Including Example Code. URL: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3 (дата звернення 06.05.2023).

13. Fish School Search. URL: https://fbln.me/fss/ (дата звернення 07.05.2023).

14. Bodyanskiy, Y., Shafronenko, A., & Pliss, I. (2022). Clusterization of vector and matrix data arrays using the combined evolutionary method of fish schools. *Системні дослідження та інформаційні технології: міжнародний науково-технічний журнал, № 4*.

15. Rastrigin, L. A. (1986). Random search in adaptation problems of stochastic systems. *IFAC Proceedings Volumes*, *19*(5), 195-196.

16. Nedler-Mead method. URL: https://en.wikipedia.org/wiki/Nelder–Mead_method (дата звернення 08.05.2023).

17. A tour of the C# language. URL: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ (дата звернення 08.05.2023).

18. C#: Що це за мова та де її використовують? URL: https://robotdreams.cc/uk/blog/284-s-chto-eto-za-yazyk-i-gde-ego-ispolzuyut (дата звернення до ресурсу 07.05.2023).

19. TIOBE Index for may 2023. URL: https://www.tiobe.com/tiobe-index/ (дата звернення 10.05.2023).

20. PYPL PopularitY of Programming Language – 2023. URL: https://pypl.github.io/PYPL.html (дата звернення 09.05.2023).

21. C Sharp – Features, Advantages and Disadvantages. URL: https://urbannaturale.com/c-sharp-features-advantages-and-disadvantages/ (дата звернення 10.05.2023).

22. Що таке NetFramework і навіщо він потрібен? URL: https://intelserv.net.ua/blog/material/id/247 (дата звернення 09.05.2023).

23. What is .NET Framework? URL: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet-framework (дата звернення 10.05.2023).

24. What is .NET Framework? Explain Architecture & Components. URL: https://www.guru99.com/net-framework.html (дата звернення 11.05.2023).

25. Desktop Guide (Windows Forms .NET). URL: https://learn.microsoft.com/enus/dotnet/desktop/winforms/overview/?view=netdesktop-7.0&viewFallbackFrom=netdesktop-5.0 (дата звернення 11.05.2023).

26. .NET regular expressions. URL: https://learn.microsoft.com/en-us/dotnet/standard/base-types/regular-expressions (дата звернення 10.05.2023).

27. Regular Expressions (Regex) Tutorial. URL: https://www.regular-expressions.info (дата звернення 13.05.2023).

28. How to write Regular Expressions? – GeekforGeeks. URL: https://www.geeksforgeeks.org/write-regular-expressions/ (дата звернення 13.05.2023).

29. Regular expression. URL: https://en.wikipedia.org/wiki/ Regular_expression#:~:text=A%20regular%20expression%20shortened%20as,strings%2C%20or%20for%20input%20validation. (дата звернення 13.05.2023).

30. A flexible charting library for .NET. URL: https://www.codeproject.com/Articles/5431/A-flexible-charting-library-for-NET (дата звернення 11.05.2023).

31. Plot Data with ZedGraph – SWHarden.com. URL: https://swharden.com/csdv/plotting-free/zedgraph/ (дата звернення 11.05.2023).

32. Getting started with .NET Charts. URL: https://www.i-programmer.info/programming/uiux/2756-getting-started-with-net-charts.html (дата звернення 12.05.2023).