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 другий (магістерський)

ДОСЛІДЖЕННЯ ТА РЕАЛІЗАЦІЯ ЕВОЛЮЦІЙНОГО МЕТОДУ КЛАСТЕРИЗАЦІЇ ДАНИХ
(тема)

Виконав:
студент 2 курсу, групи ІНФм-23-2
Фалько М.К.
(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)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Фальку Михайлу Костянтиновичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та реалізація еволюційного методу кластеризації даних

затверджена наказом по університету від 25 листопада 2024 року № 1246Ст

2. Термін подання студентом роботи до екзаменаційної комісії 23 грудня 2024 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані Інтернет-джерел та відомих наукових проєктів, документація Python, набір синтетичних даних для тренування та тестування системи, бібліотеки для наукових обчислень та візуалізації (NumPy, Plotly), MongoDB для зберігання параметрів і результатів.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз предметної галузі.
2. Постановка задачі.
3. Опис програмного рішення.
4. Опис результатів досліджень.

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, еволюційний метод кластеризації даних, 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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	25.11.2024	
2	Аналіз завдання, підбір літератури	26.11.24-28.11.24	
3	Аналіз літератури з досліджуваної проблеми	29.11.24-02.12.24	
4	Аналіз технічних засобів	03.12.24-05.12.24	
5	Розробка еволюційного методу кластеризації даних	06.12.24-07.12.24	
6	Програмна реалізація	08.12.24-12.12.24	
7	Оформлення пояснювальної записки	12.12.24-14.12.24	
8	Перевірка на плагіат	15.12.2024	
9	Рецензування	18.12.2024	
10	Підготовка презентації та доповіді	23.12.2024	
11	Занесення роботи в електронний архів	02.01.2025	
12	Попередній захист кваліфікаційної роботи	02.01.2025	

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

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

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 71 с., 2 табл., 35 рис., 39 джерел.

ГІБРИДНИЙ МЕТОД, ПРАВДОПОДІБНА НЕЧІТКА КЛАСТЕРИЗАЦІЯ, ЕВОЛЮЦІЙНИЙ МЕТОД, ПРОЦЕДУРА ОПТИМІЗАЦІЇ, ДАНІ, ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ.

Об’єктом дослідження є кластеризація даних, що надходять на обробку послідовно, в онлайн режимі.

Метою роботи є дослідження та реалізація еволюційного методу кластеризації даних, що надходять на обробку послідовно в режимі реального часу (онлайн).

В ході виконання кваліфікаційної роботи проведено теоретичний аналіз принципів нечіткої кластеризації даних на основі еволюційних алгоритмів, а також їх переваг та недоліків. Розроблено еволюційний метод нечіткої кластеризації даних, що надходять на обробку послідовно в режимі реального часу (онлайн). Експериментальні дослідження показали ефективність запропонованого методу на різних типах даних, включаючи великі та складні набори даних.

HYBRID METHOD, CREDIBILISTIC FUZZY CLUSTERING, EVOLUTIONARY METHOD, OPTIMIZATION PROCEDURE, DATA, INTELLIGENT DATA ANALYSIS.

The object of the research is the clustering of data received for processing sequentially online.

The purpose of the research is the research and implementation of the evolutionary method of clustering data that are received for processing sequentially in real-time (online).

In the course of the qualification work, a theoretical analysis of the principles of fuzzy data clustering based on evolutionary algorithms, as well as their advantages and disadvantages, was carried out. An evolutionary method of fuzzy clustering of data received for processing sequentially in real-time mode (online) has been developed. Experimental researchers have shown the effectiveness of the proposed method on various types of data, including large and complex data sets.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних

ШІ – штучний інтелект

AI – Artificial Intelligence (штучний інтелект)

API – Application Programming Interface (інтерфейс програмування застосунків)

CGWO – Chaotic Gray Wolf Optimizer (хаотичний алгоритм сірого вовка)

DB – Database (база даних)

DBSCAN – Density-Based Spatial Clustering of Applications with Noise (просторове кластерування застосувань з шумом на основі густини)

GWO – Gray Wolf Optimizer (алгоритм сірого вовка)

HTML – HyperText Markup Language (мова розмітки гіпертексту)

PSO – Particle Swarm Optimization (оптимізація рою частинок)

ВСТУП

У сучасному світі аналіз та обробка даних мають вирішальне значення для різних галузей, таких як наука, бізнес, медицина та інформаційні технології. Одним із ключових аспектів аналізу даних є класифікація та групування інформації за спільними ознаками або характеристиками. У цьому контексті кластерний аналіз слугує потужним методом для ідентифікації подібних шаблонів чи груп у даних. Однак, попри успіхи існуючих методів кластеризації, вони мають деякі обмеження, які впливають на їхню точність. Одним із таких обмежень є труднощі у визначенні оптимальної кількості кластерів, а також недостатня стійкість до шумів та викидів у даних.

Для вирішення проблем, пов’язаних із кластеризацією, необхідно впроваджувати нові підходи та методи, які б підвищували якість кластеризації і були менш чутливими до шуму. Одним з перспективних рішень є гібридні методи, що поєднують переваги різних алгоритмів кластеризації. Еволюційні алгоритми, завдяки своїй здатності знаходити оптимальні рішення в складних просторах, можуть відігравати важливу роль у вдосконаленні процесу кластеризації шляхом оптимізації параметрів або пошуку нових структур кластерів [1-15].

У цьому дослідженні планується детально розглянути основні підходи до кластерного аналізу, а також застосування еволюційних алгоритмів для підвищення ефективності кластеризації. Окрім цього, буде проведено огляд існуючих гібридних методів, що дозволяють комбінувати різні підходи та компенсувати їхні недоліки, що призводить до кращих результатів кластеризації.

Однією з найбільших проблем класичних методів є вибір оптимальної кількості кластерів [2, 4-5]. Це питання стає особливо складним при роботі з великими обсягами даних або з їхньою неоднорідною структурою. Багато алгоритмів вимагають визначення кількості кластерів заздалегідь, що може призвести до неповних або неточних результатів.

Ключову роль у кластеризації також відіграє попередня обробка даних. Цей етап включає видалення викидів, нормалізацію ознак, вибір найбільш репрезентативних характеристик тощо. Від якості підготовки даних залежить стабільність і точність кінцевих результатів. Гібридні методи можуть використовувати кілька технік обробки даних для досягнення найкращого результату [15, 16].

Центральне місце в цьому дослідженні займає розробка гібридного методу кластеризації на основі еволюційного алгоритму. Генетичні алгоритми та інші еволюційні підходи добре підходять для оптимізації складних проблем, оскільки здатні адаптуватися до змінних умов і складних структур даних. Особлива увага приділятиметься налаштуванню параметрів таких алгоритмів, як розмір популяції, ймовірність мутацій та кросоверу, що дозволить підвищити ефективність і швидкість роботи методу.

Такий гібридний підхід може значно підвищити ефективність вирішення проблем кластеризації, зокрема вибору оптимальної кількості кластерів і зменшення чутливості до шуму в даних. Його застосування може бути корисним у різних сферах: аналіз соціальних мереж – для виявлення спільнот, аналізу поведінкових моделей користувачів та їх взаємодій; медична діагностика – для кластеризації пацієнтів на основі схожих симптомів або результатів медичних досліджень, що дозволяє точніше ставити діагнози; фінансовий аналіз – для сегментації ринків, виявлення шахрайських дій або оцінки кредитоспроможності.

Розробка такого методу може суттєво вплинути на розвиток кластерного аналізу, розширюючи його можливості в умовах зростання обсягів та складності даних. Впровадження нових підходів до кластеризації є актуальним завданням для різних сфер, що потребують обробки великих масивів даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Методи кластеризації

У сфері аналізу даних існує велика кількість алгоритмів кластеризації, які використовуються для групування схожих об’єктів. Ось кілька найбільш поширених:

– метод *k*-середніх (*k*-means) – це один з найпопулярніших методів кластеризації, який працює за принципом визначення центрів кластерів та мінімізації відстані об’єктів до цих центрів. Його перевагою є швидкість роботи з великими наборами даних, проте недоліком є необхідність заздалегідь визначати кількість кластерів, що може бути проблемою для складних або неоднорідних даних;

– ієрархічна кластеризація. Цей підхід формує дерево подібності (дендрограму), що дозволяє побачити всі можливі рівні кластеризації та вибрати оптимальну кількість кластерів постфактум. Ієрархічний метод не потребує попереднього визначення кількості кластерів, але може бути занадто повільним для великих наборів даних через високу обчислювальну складність;

– DBSCAN (Density-Based Spatial Clustering of Applications with Noise). Цей метод добре підходить для даних зі складними структурами та неоднорідною густотою кластерів. Він здатен автоматично визначати кількість кластерів та розпізнавати шумові точки, що робить його менш залежним від обмежень, як у методі *k*-середніх. Однак, DBSCAN може погано працювати на даних із змінною щільністю;

– агломеративна кластеризація – один з підходів ієрархічної кластеризації, де кожен об’єкт спочатку вважається окремим кластером, а потім поступово об’єднується з іншими. Цей метод також не потребує визначення кількості кластерів наперед, але може бути досить вимогливим до обчислювальних ресурсів;

– спектральна кластеризація. Застосовується до графів та може ефективно виявляти складні структури в даних. Цей метод використовує властивості спектра (власних значень) матриці подібностей даних. Спектральна кластеризація дає добрі результати на складних даних, але потребує точного визначення методу побудови матриці подібностей.

При розробці гібридного методу кластеризації на основі еволюційних алгоритмів важливо враховувати переваги й недоліки кожного з цих методів. Еволюційні алгоритми, такі як генетичні алгоритми, дозволяють адаптивно налаштовувати параметри кластеризації, що може забезпечити покращення в розв’язанні задач, пов’язаних із вибором кількості кластерів, стійкістю до шуму та іншими аспектами [17-34].

Дослідження та порівняння існуючих методів кластеризації є важливим кроком для вибору оптимальних підходів до аналізу даних. Однак, оскільки завдання та вимоги до обробки даних постійно змінюються, необхідність у розробці нових методів кластеризації стає все більш актуальною. Дійсно, не всі методи однаково ефективні для різних сценаріїв. Наприклад, метод *k*-середніх стикається з труднощами при визначенні кількості кластерів і може бути ненадійним для складних структур або неоднорідної густини даних. Ієрархічна кластеризація, хоч і не потребує попередньої інформації про кількість кластерів, має високу обчислювальну складність, що обмежує її застосування для великих наборів даних.

У зв’язку з цим пошук нових методів, здатних вирішувати ці обмеження, є критично важливим. Одним із перспективних підходів є застосування генетичних алгоритмів та інших еволюційних стратегій. Генетичні алгоритми імітують процеси природного відбору та мутації, що дозволяє ефективно вирішувати оптимізаційні задачі навіть у складних випадках, де традиційні методи кластеризації не працюють. У контексті кластеризації, ці алгоритми можуть автоматично підбирати оптимальну кількість кластерів і їхню внутрішню структуру, адаптуючись до різноманітних даних без необхідності заздалегідь задавати параметри. Це

робить генетичні алгоритми цінним інструментом для вирішення задач, де традиційні методи неефективні або потребують додаткових налаштувань.

Іншим цікавим напрямком є впровадження методів машинного навчання для підвищення ефективності кластеризації. Використання підходів з навчанням, таких як методи передачі навчання, дозволяє адаптувати існуючі моделі кластеризації до нових наборів даних, що знижує необхідність повного перенавчання. Така адаптивність дозволяє зменшити обчислювальні витрати і водночас покращити точність, використовуючи наявне знання про структуру даних.

Комбінація еволюційних алгоритмів з техніками машинного навчання відкриває нові горизонти у створенні гібридних методів кластеризації, які можуть адаптуватися до складних, динамічно змінюваних даних. Такий підхід сприяє підвищенню ефективності кластеризації та розширює можливості її застосування у різних галузях, від аналізу великих даних до медичних і фінансових систем.

1.2 Аналіз еволюційних алгоритмів кластеризації даних

Еволюційні алгоритми представляють собою унікальний підхід до кластеризації даних, використовуючи принципи еволюції та природного відбору для пошуку оптимальних рішень. Основні методи, такі як генетичні алгоритми та еволюційні стратегії, використовують популяцію рішень, де кожен індивідуум представляє можливе рішення задачі кластеризації. При цьому якість кожного рішення визначається на основі певної метрики, яка вимірює ефективність кластеризації (наприклад, сумарна внутрішньокластерна варіація або інші показники).

Однією з головних переваг еволюційних алгоритмів є їхня здатність проводити глобальний пошук у просторі рішень. Це дає їм значну перевагу над традиційними методами, такими як *k*-середніх або ієрархічна кластеризація,

які часто можуть застрягати в локальних мінімумумах. Еволюційні алгоритми дозволяють уникнути цих локальних пасток завдяки використанню механізмів мутацій, схрещування та природного відбору, що забезпечує широкий огляд можливих варіантів рішень [17, 20-25]. Використання еволюційних алгоритмів для кластеризації дійсно супроводжується низкою викликів, які необхідно враховувати для досягнення оптимальних результатів.

Налаштування параметрів алгоритму. Вибір таких параметрів, як розмір популяції, ймовірності мутації та кросоверу, є критично важливим. Неправильні параметри можуть призвести до передчасної конвергенції, коли алгоритм швидко «застряє» в локальних мінімумах, або до надмірної експлорації, яка не приносить суттєвих результатів. Баланс між пошуком нових рішень (експлорація) і поліпшенням поточних (експлуатація) має бути оптимізований.

Обчислювальні витрати. Через необхідність оцінювати пристосованість кожного індивідуума в популяції протягом численних поколінь, еволюційні алгоритми можуть вимагати значних обчислювальних ресурсів, особливо при роботі з великими наборами даних. Це може стати обмеженням, зокрема у випадках, коли швидкість обробки є критично важливою.

Забезпечення різноманітності популяції. Різноманітність популяції важлива для запобігання передчасній конвергенції та забезпечення пошуку нових рішень у просторі рішень. Для цього застосовують методи, такі як нішування або катастрофічний відбір, які допомагають підтримувати популяцію різноманітною і стимулюють пошук нових, перспективних рішень.

Інші виклики:

– передчасна конвергенція. Ця проблема виникає, коли алгоритм знаходить локальні оптимуми, не дослідивши весь простір рішень;

– вибір функції пристосованості. Функція, яка оцінює якість рішень, має бути правильно підібраною до конкретної задачі кластеризації, що може бути викликом через варіативність типів даних та критеріїв;

– адаптація до специфіки даних. Еволюційні алгоритми мають бути

адаптовані до характеристик даних (розподіл, кількість ознак, типи кластерів), щоб забезпечити їхню ефективність;

– інтерпретація результатів. Результати кластеризації повинні бути інтерпретовані з точки зору реальних даних, що може бути складним завданням, якщо кластери мають неочевидну структуру.

Незважаючи на ці виклики, адаптивність еволюційних алгоритмів робить їх перспективними для вирішення складних задач кластеризації, таких як робота з нерівномірною густотою або складними структурами даних. Вони можуть знайти застосування в різних областях, де традиційні методи кластеризації не показують достатньо високої ефективності. Комбінуючи глобальний пошук з можливістю адаптації до різних типів даних, еволюційні алгоритми пропонують гнучкий і потужний інструмент для аналізу даних у сучасному світі.

1.2.1 Види еволюційних алгоритмів

Еволюційні алгоритми – це клас оптимізаційних методів, що імітують принципи природної еволюції. Вони використовуються для розв’язання складних задач, включаючи кластеризацію, маршрутизацію, управління ресурсами та інші. Основні види еволюційних алгоритмів включають:

– генетичні алгоритми (Genetic Algorithms, GA) – це методи оптимізації та пошуку, які імітують процеси природної еволюції, такі як селекція, кросовер і мутація. Цей підхід використовується для розв’язання складних задач, де традиційні алгоритми можуть бути неефективними. Генетичні алгоритми засновані на концепції еволюції Дарвіна: «виживають найсильніші». Основні кроки алгоритму такі:

Крок 1. Ініціалізація популяції. Популяція – це набір можливих рішень задачі, які називають хромосомами. Кожна хромосома – це структура даних, яка представляє одне можливе рішення.

Крок 2. Оцінка функції пристосованості (Fitness Function). Кожне рішення оцінюється за допомогою функції, яка визначає його якість. Мета алгоритму – максимізувати або мінімізувати значення цієї функції.

Крок 3. Селекція (вибір найкращих). Найбільш пристосовані особини (хромосоми) обираються для подальшого розмноження.

Крок 4. Генетичні оператори. Кросовер (схрещування) – об'єднання частин двох хромосом-батьків для створення нових хромосом-нащадків. Мутація – випадкова зміна одного чи кількох генів у хромосомі для додавання різноманітності.

Крок 5. Створення нового покоління. Нове покоління замінює старе, і процес повторюється, поки не досягнуто критерію зупинки.

Перевагами генетичних алгоритмів є глобальний пошук за допомогою якого генетичні алгоритми можуть знаходити рішення в складних і нелінійних просторах; гнучкість – підходять для задач з різними типами даних (числові, дискретні, комбінаторні); простота реалізації – не вимагають знання градієнтів чи інших специфічних властивостей функцій. Генетичні алгоритми – потужний інструмент, особливо для задач, які складно вирішити стандартними методами;

– диференційна еволюція (Differential Evolution, DE) – це метод оптимізації, який належить до класу еволюційних алгоритмів і використовується для глобального пошуку рішень. Його особливістю є простота реалізації та ефективність у задачах, пов'язаних із безперервними функціями та великим простором параметрів;

– алгоритми рою частинок (Particle Swarm Optimization, PSO) – це метод оптимізації, натхненний колективною поведінкою зграї птахів, рою бджіл чи косяка риб. Вони використовуються для вирішення задач оптимізації в багатовимірних просторах і належать до еволюційних алгоритмів. Алгоритми рою частинок є одним із найпопулярніших інструментів для задач оптимізації завдяки своїй простоті та ефективності;

– алгоритми мурашиних колоній (Ant Colony Optimization, ACO) –

це метаевристичний підхід для вирішення задач оптимізації, натхненний поведінкою мурах у природі. Зокрема, алгоритм моделює процес пошуку їжі, під час якого мурахи залишають феромони на шляху, вказуючи іншим оптимальні маршрути. Базуються на поведінці мурах, які знаходять найкоротший шлях до їжі;

– еволюційні стратегії (Evolution Strategies, ES) – це клас еволюційних алгоритмів, призначений для оптимізації реальних функцій. Вони зосереджені на процесах мутації**,** відбору та самоадаптації параметрів, що робить їх ефективними для безперервних задач оптимізації. Сфокусовані на пошуку рішень через мутації та відбір;

– культурні алгоритми (Cultural Algorithms) – це метаевристичний підхід до оптимізації, що моделює не лише еволюцію особин (як у генетичних алгоритмах), а й передачу культурного знання між поколіннями. Цей метод базується на ідеї, що знання, накопичене популяцією, може покращити ефективність пошуку оптимальних рішень. Розширення генетичних алгоритмів, що враховують «культурний шар» – знання, яке накопичується протягом еволюції. Основна ідея полягає в тому, що знання про найкращі рішення передаються між поколіннями;

– гібридні еволюційні алгоритми – це методи оптимізації, які поєднують принципи еволюційних алгоритмів із іншими техніками оптимізації або аналізу, такими як методи градієнтного спуску, локальні пошукові алгоритми, або методи, що використовують інші математичні моделі. Метою таких гібридних підходів є підвищення ефективності та швидкості пошуку оптимальних рішень.

1.2.2 Алгоритм «Сірих вовків»

Алгоритм «Сірих вовків» (Gray Wolf Optimizer, GWO) – це метаевристичний алгоритм оптимізації, натхненний полюванням сірих вовків

у природі. Цей алгоритм був запропонований в 2014 році Хосейном Мірсалаї та іншими авторами і є частиною групи еволюційних алгоритмів, заснованих на поведінці тварин. Основною ідеєю є використання соціальної структури вовчої зграї для знаходження оптимальних рішень у задачах оптимізації.

Сірі вовки в зграї мають чітко визначену ієрархію, що визначає їхню поведінку під час полювання. Ієрархія вовків виглядає так:

– альфа-вовк (Alpha) – це лідер зграї, який приймає основні рішення, у тому числі вибір напрямку для полювання;

– бета-вовки (Beta) – вони є помічниками альфа-вовка і допомагають йому в ухваленні рішень;

– дельта-вовки (Delta) – це досвідчені вовки, які допомагають підтримувати порядок у зграї;

– омега-вовки (Omega) – це вовки, що займають нижчу позицію в ієрархії та виконують основні завдання з пошуку їжі.

Алгоритм складається з декількох основних кроків:

Крок 1. Ініціалізація. Створюється початкова популяція вовків, де кожен вовк представляє кандидатне рішення для оптимізаційної задачі.

Крок 2. Оцінка рішень. Для кожного вовка оцінюється функція вартості або функція об’єктиву (яка може бути мінімізована чи максимізована).

Крок 3. Оновлення позицій вовків. Альфа, бета та дельта вовки визначають напрямок пошуку для інших вовків. Позиції кожного вовка оновлюються за допомогою таких формул:

Коли жертва знайдена, починається ітерація (t=1). Згодом α – , β – та δ – вовки керуватимуть ω, щоб переслідувати здобич і, зрештою, оточити її. Три коефіцієнти A, B і C пропонуються для опису поведінки оточення:

$$C_{\alpha} = \left| B_1 * GW_{\alpha} - X(t) \right|,$$

$$C_{\beta} = \left| B_2 * GW_{\beta} - X(t) \right|,$$

$$C_\delta = \left| B_3 * GW_\delta - X(t) \right|,$$

де t вказує на поточну ітерацію;

GW вектор позиції сірого вовка;

GW_1, GW_2 і GW_3 – є векторами положення α – , β – та δ – вовків, що обчислюється наступним чином:

$$GW_1 = GW_\alpha - A_1 * C_\alpha,$$

$$GW_2 = GW_\beta - A_2 * C_\beta,$$

$$GW_3 = GW_\delta - A_3 * C_\delta,$$

$$GW(t) = \frac{GW_1 + GW_2 + GW_3}{3}.$$

Крок 4. Конвергенція. Алгоритм триває до тих пір, поки не буде досягнута конвергенція до оптимального рішення або не буде досягнута максимальна кількість ітерацій.

Схематично представити роботу алгоритму можна наступним чином (рис. 1.1) [8].

Рисунок 1.1 – Схема роботи алгоритму GWO

Багато алгоритмів ройового інтелекту імітують поведінку тварин під час полювання та пошуку їжі. Проте GWO відрізняється тим, що враховує внутрішню ієрархію в зграї вовків. Позиція найкращого рішення визначається не тільки одним, а трьома рішеннями – альфа, бета та дельта, що дозволяє більш точно оцінити результат. В інших ж алгоритмах ройового інтелекту оптимальне рішення зазвичай базується лише на одному – локальному оптимумі. Блок-схема алгоритму «Сірих вовків» наведена на рисунку 1.2 [8].

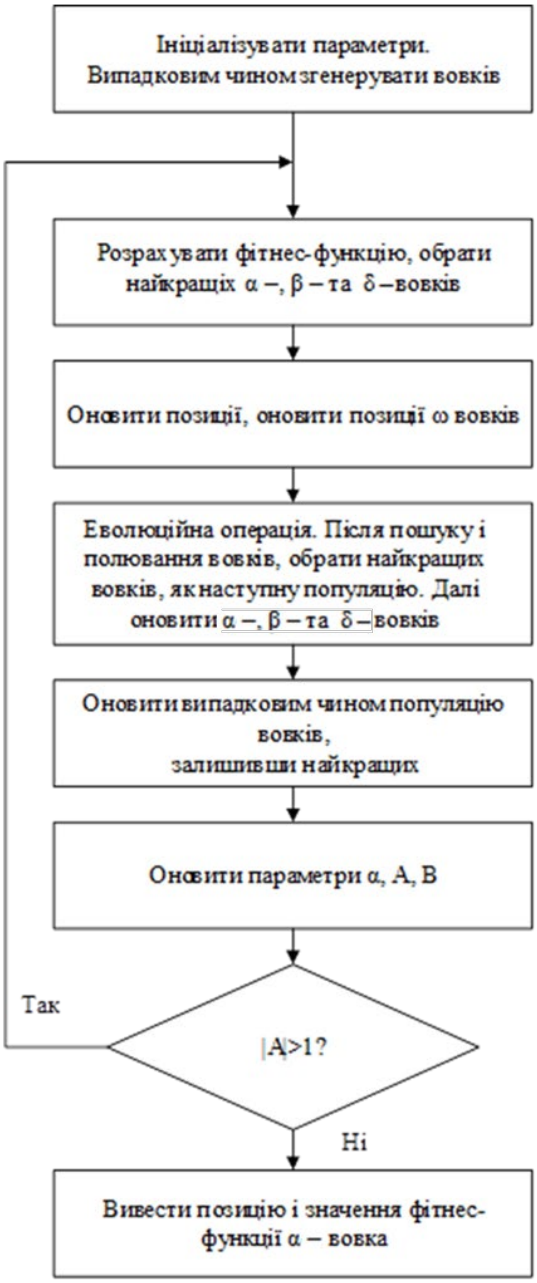

Рисунок 1.2 – Блок-схема алгоритму «Сірих вовків»

1.2.3 Алгоритм «Котячої зграї»

В останні роки в області оптимізації було запропоновано багато алгоритмів, таких як генетичний алгоритм (GA), оптимізація мурашиних колоній (ACO), оптимізація рою часток (PSO) тощо. Деякі з цих алгоритмів базуються на принципах ройового інтелекту. Оптимізація Cat Swarm (CSO), алгоритм є поєднанням ідей з PSO і ACO. Згідно з літературними джерелами, PSO з ваговим коефіцієнтом зазвичай знаходить оптимальні рішення швидше, ніж класичний PSO, але експериментальні результати показують, що оптимізація Cat Swarm (CSO) демонструє ще кращу ефективність. Спостерігаючи за поведінкою різних тварин, можна отримати корисні інсайти для вирішення завдань оптимізації. Наприклад, поведінка мурах була вивчена для розробки алгоритму ACO, а рухи зграї чайок лежать в основі PSO. Вивчаючи поведінку котів, можна розробити алгоритм оптимізації Cat Swarm (CSO) для пошуку оптимальних рішень.

Згідно з класифікацією в біології, в котячому світі налічується близько тридцяти двох різних видів тварин, таких як лев, тигр, леопард, кішка та інші. Хоча ці види можуть мати різні середовища існування, вони демонструють схожі поведінкові характеристики. Мисливські навички не є вродженими для всіх кішок, але їх можна навчити через досвід. Для диких кішок ці навички є необхідними для виживання, тоді як для домашніх кішок вони проявляються у вигляді природного інстинкту – сильної цікавості до будь-яких рухомих об’єктів.

Попри те, що всі кішки мають високу цікавість, більшість з них не діють на імпульс і часто просто спостерігають. Якщо провести деякий час, спостерігаючи за кішками, можна помітити, що більшість часу вони проводять у стані спокою, відпочиваючи або сплячи. Однак їхня пильність дуже висока – навіть під час відпочинку вони залишаються уважними до навколишнього середовища. Зовні кішки можуть виглядати ледачими, лежачи і закривши очі, але насправді вони завжди насторожені, ретельно спостерігаючи за тим, що

відбувається навколо.

Ці спостереження демонструють, що, хоча кішки можуть здаватися пасивними, насправді вони дуже обдумані й обережні. Звісно, якщо уважно спостерігати за їх поведінкою, можна помітити ще багато інших цікавинок, які відображають їхню складну та багатогранну натуру.

В алгоритмі CSO спочатку визначається кількість кішок, які використовуються для вирішення задачі. Кожна кішка має своє власне положення в просторі, яке складається з M вимірів швидкості для кожного виміру. Окрім того, кожна кішка має значення придатності, яке відображає її здатність пристосуватися до функції придатності, а також прапор, що визначає, чи знаходиться кішка в режимі пошуку або відстеження.

Процес оптимізації в CSO полягає в тому, що кожна кішка прагне знайти оптимальне рішення, яке представляється як її найкраща позиція. Алгоритм зберігає найкраще знайдене рішення в процесі ітерацій, і це рішення в кінцевому підсумку стає найбільш оптимальним, оскільки алгоритм використовує його до завершення пошуку. Тому остаточне рішення полягає в тому, щоб вибрати найкращу позицію серед усіх кішок, досягнуту за час виконання алгоритму.

У режимі пошуку в алгоритмі CSO визначаються чотири ключові фактори: пошук пулу пам’яті (SMP), пошук обраного діапазону вимірювання (SRD), кількість вимірювань для зміни (CDC) та розгляд власної позиції (SPC). Кожен з цих факторів має важливу роль у процесі оптимізації, і їх коректне використання забезпечує ефективність пошуку.

Пошук пулу пам’яті (SMP) – це параметр, який визначає розмір пам’яті пошуку для кожної кішки. Він вказує на точки в пошуковому просторі, які шукає кішка. Кішка вибирає точку з пулу пам’яті, згідно з визначеними правилами, що дозволяє їй фокусуватися на найбільш перспективних напрямках.

Пошук обраного діапазону вимірювання (SRD). SRD визначає коефіцієнт мутації для вибраних вимірювань. Якщо певне вимірювання

вибране для зміни, різниця між новим і старим значенням не повинна виходити за межі діапазону, визначеного SRD. Це обмеження дозволяє контролювати процес мутації та зберігати зміни в межах допустимих значень.

Кількість вимірювань для зміни (CDC). CDC визначає, скільки вимірювань буде змінено під час ітерації. Цей параметр дозволяє контролювати ступінь варіації в пошуку, забезпечуючи збалансоване оновлення позицій без занадто великих або малих змін.

Розгляд власної позиції (SPC). SPC – це логічна змінна, яка визначає, чи буде точка, в якій кішка стоїть, одним із кандидатів для переходу. Незалежно від того, чи значення SPC є істинним чи хибним, значення SMP не змінюється.

Опис процесу режиму пошуку в 5 кроках:

Крок 1. Кішка ініціалізує пул пам’яті (SMP), вибираючи точки для пошуку.

Крок 2. Кішка визначає, які вимірювання будуть змінюватися, і встановлює діапазон мутації через SRD.

Крок 3. Вибирається кількість вимірювань для зміни (CDC).

Крок 4. Кішка проводить мутацію у межах визначеного діапазону, забезпечуючи збалансовані зміни.

Крок 5. Кішка перевіряє свою поточну позицію через SPC, визначаючи, чи є вона готовою до переходу.

Цей процес дозволяє кішкам ефективно шукати оптимальні рішення, здійснюючи мутації, при цьому не виходячи за межі дозволених діапазонів і зберігаючи свою адаптацію до функції придатності (рис. 1.3) [18].

Можна зазначити, що розглянутий алгоритм пошуку реалізує по суті покоординатний спуск (метод Гаусса – Зейделя), який потребує багаторазових оцінок значень функції, що оптимізується, і характеризується низькою швидкістю збіжності. У режимі трасування градієнтний пошук реалізовано з великим кроком, що, як правило, не гарантує знаходження глобального екстремуму. У зв’язку з цим видається доцільним модифікувати процедуру оптимізації на основі cat swarm шляхом її рандомізації на основі випадкового

пошуку, що має ряд переваг перед детермінованими процедурами пошуку екстремуму.

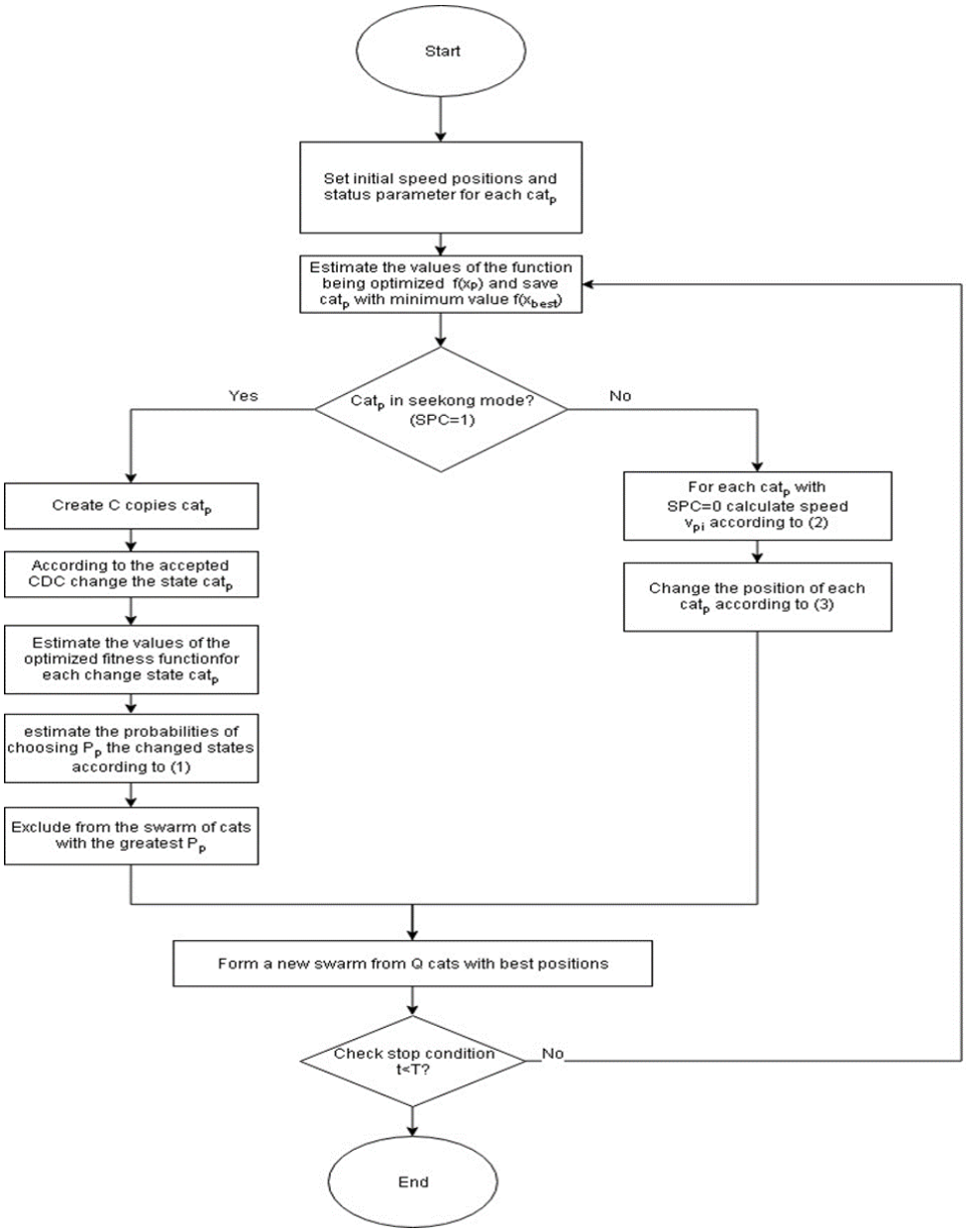

Рисунок 1.3 – Блок-схема алгоритму «Котячих зграй»

1.3 Постановка задачі дослідження

Кластерний аналіз відіграє ключову роль у сучасних підходах до обробки великих обсягів даних, оскільки дозволяє виявляти приховані структури та шаблони в інформації. Однак, існуючі методи кластеризації

мають певні обмеження, такі як труднощі у визначенні кількості кластерів і низька стійкість до шуму.

Однією з найбільш цікавих перспектив у цій галузі є використання гібридних підходів до кластеризації. Гібридні методи можуть поєднувати переваги різних алгоритмів, наприклад, ієрархічної кластеризації та методів *k*-середніх або ж поєднувати машинне навчання з еволюційними підходами. Еволюційні алгоритми, такі як генетичні алгоритми або ройові методи (наприклад, алгоритми частинок), здатні допомогти в оптимізації параметрів кластеризації, автоматично визначати оптимальну кількість кластерів, а також знаходити нетрадиційні структури у даних.

Завдяки своїй природі еволюційні алгоритми мають високу здатність адаптуватися до різних умов, що робить їх корисними при роботі з шумовими даними та викидами. Це дозволяє отримати більш стабільні та точні результати в складних сценаріях, де інші методи можуть не справлятися з поставленими завданнями. Отже, розвиток гібридних підходів та застосування еволюційних алгоритмів у кластерному аналізі відкриває нові можливості для підвищення якості аналізу даних, забезпечуючи кращу адаптацію до реальних умов та складних структур у даних.

Об’єктом дослідження є кластеризація даних, що надходять на обробку послідовно, в онлайн режимі.

Метою роботи є дослідження та реалізація еволюційного методу кластеризації даних, що надходять на обробку послідовно в режимі реального часу (онлайн).

Для досягнення мети необхідно вирішити такі завдання:

– аналіз та опис предметної області;

– постановка завдання;

– розробка методу правдоподібної нечіткої кластеризації на основі еволюційного алгоритму;

– експериментальна перевірка розробленого метода.

2 ЕВОЛЮЦІЙНИЙ МЕТОД ПРАВДОПОДІБНОЇ КЛАСТЕРИЗАЦІЇ ДАНИХ

2.1 Гібридні методи кластеризації

Гібридні методи кластеризації даних поєднують різні підходи та алгоритми для підвищення якості кластеризації, кращої адаптації до специфічних умов задачі та підвищення ефективності обробки даних. Одним із таких методів є кластеризація на основі алгоритму «Fish schools», який імітує поведінку риб у пошуках їжі. Алгоритм працює на основі індивідуальних і групових рухів риб, що допомагає знаходити локальні максимуми функції розподілу густини даних. Поєднання індивідуальних, інстинктивно-колективних та колективно-вольових рухів дозволяє ефективно знаходити глобальні й локальні екстремуми, що прискорює процес кластеризації та покращує її якість [8-12].

Метод кластеризації на основі «Скажених вовків» застосовує модифікований алгоритм сірих вовків, який поєднує еволюційні алгоритми та глобальний випадковий пошук для виявлення глобальних екстремумів цільової функції кластеризації. В цьому алгоритмі вовки імітують поведінку реальних тварин під час полювання, проходячи етапи відстеження, переслідування, оточення та атаки здобичі. Така модель дозволяє ефективно досліджувати простір пошуку та знаходити оптимальні рішення навіть у складних багатоекстремальних функціях, підвищуючи точність і швидкість процесу кластеризації [8].

Метод «Зграї котів» [12-14, 16] поєднує випадковий пошук із еволюційними стратегіями, ґрунтуючись на поведінці котів. У цьому алгоритмі коти демонструють два типи поведінки: активну (полювання), коли вони активно шукають здобич (розв’язок), та пасивну (спостереження), коли вони стежать за ситуацією та збирають інформацію. Така структура алгоритму дозволяє ефективно чергувати активні та пасивні фази пошуку, що

сприяє більш глибокому дослідженню простору можливих рішень і підвищує ймовірність знаходження глобальних екстремумів у процесі кластеризації (рис. 2.1).

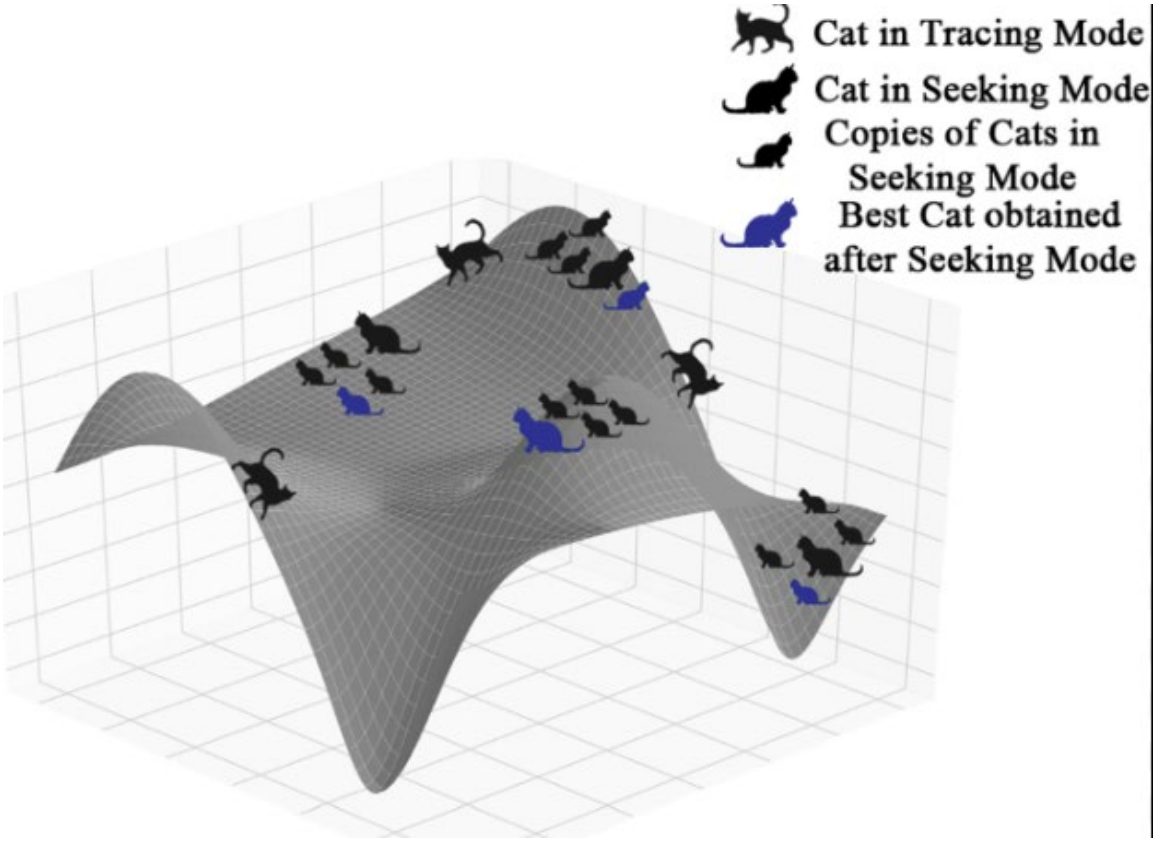

Рисунок 2.1 – Еволюційний метод «Зграї котів» (Cat Swarm)

Цей підхід дозволяє алгоритму ефективно використовувати два режими – дослідження та експлуатацію пошукового простору, що значно покращує його здатність знаходити оптимальні рішення в умовах багатоекстремальної оптимізації. Особливо метод ефективний при роботі з даними, що містять багато локальних оптимумів, забезпечуючи високу точність результатів.

Спільними характеристиками багатьох гібридних методів є їх здатність працювати з великими обсягами даних, гнучкість у налаштуванні параметрів і можливість інтеграції переваг традиційних алгоритмів з сучасними методами машинного навчання.

Основною метою гібридного методу правдоподібної кластеризації,

описаного в дослідженні, є пошук глобальних екстремумів цільової функції. Він заснований на модифікованому алгоритмі сірих вовків, який поєднує еволюційні алгоритми з глобальним випадковим пошуком. Цей підхід дозволяє досягти високої швидкості та надійності в задачах багатоекстремальної фазової кластеризації [35-37].

При використанні гібридного методу, заснованого на комбінованому еволюційному алгоритмі рибних косяків для кластеризації векторних і матричних масивів даних, основною метою є оптимізація функцій розподілу густини цих масивів. Це дозволяє зменшити кількість повторних запусків процедури оптимізації, ефективно знаходити екстремуми у складних багатоекстремальних функціях і забезпечує простоту чисельної реалізації.

Одним із ключових елементів цього гібридного підходу є інтеграція рандомізованого пошуку та еволюційної оптимізації, що дозволяє досягти високої точності й швидкості обробки даних. Такий метод особливо ефективний при роботі з великими, високорозмірними і складними масивами даних, де потрібні швидкі й точні рішення.

2.2 Особливості задачі правдоподібної кластеризації даних

Задача дослідження гібридного методу правдоподібної кластеризації на основі еволюційного алгоритму є комплексною задачею в галузі аналізу даних, що поєднує декілька важливих аспектів: кластеризацію, обробку великих обсягів даних у різних режимах (пакетному та онлайн), а також використання правдоподібного підходу. Основною метою такого дослідження є виявлення глобальних екстремумів цільової функції правдоподібної нечіткої кластеризації за допомогою модифікованого еволюційного алгоритму, що забезпечує більш точну та гнучку кластеризацію [38, 39].

Особливості задачі:

– правдоподібний підхід до кластеризації. Він орієнтований на

максимізацію правдоподібності кластерів і враховує невизначеність, що виникає в реальних даних. У нечіткій кластеризації об’єкти можуть належати до кількох кластерів з певною мірою ймовірності, що дозволяє більш точно моделювати складні структури даних;

– еволюційний алгоритм. У модифікованій версії еволюційного алгоритму, який використовується для цієї задачі, поєднуються класичні елементи еволюційного пошуку (мутація, кросовер, відбір) та глобальний випадковий пошук для підвищення ефективності знаходження глобальних екстремумів. Це важливо, оскільки традиційні методи можуть застрягати в локальних мінімумумах;

– обробка великих масивів даних у різних режимах:

1) пакетний режим. Передбачає опрацювання значних обсягів даних, що надходять заздалегідь. Такий підхід є оптимальним для завдань, де є доступ до повного набору даних на початку процесу аналізу;

2) онлайн-режим. Вимагає обробки даних в реальному часі, де нові дані постійно додаються, і алгоритм повинен адаптуватися до змінних умов. Це корисно для таких завдань, як поточний моніторинг або прогнозування;

3) глобальний пошук. Інтеграція глобального випадкового пошуку в еволюційний алгоритм дозволяє зменшити ризик передчасної конвергенції та поліпшити шанси на знаходження глобальних екстремумів у просторі рішень.

Гібридний метод повинен забезпечити кращу продуктивність порівняно з класичними методами кластеризації завдяки здатності адаптуватися до складних даних, динамічно оновлюватися в онлайн-режимі та ефективно використовувати глобальні механізми пошуку. Це дозволить отримати більш точні й стабільні результати кластеризації навіть у складних, неоднорідних наборах даних [34, 37].

Подальші дослідження і розробка такого методу можуть значно розширити можливості кластерного аналізу та відкрити нові шляхи для

практичного використання в різних сферах, таких як аналіз великих даних, медична діагностика, фінансові системи та багато інших [8, 9].

Ключовим аспектом вирішення задачі правдоподібної нечіткої кластеризації є використання модифікованого алгоритму «сірих вовків» (GWO), який належить до класу швидкодіючих еволюційних алгоритмів. Удосконалення цього методу шляхом додавання випадкових збурень для покращення глобального пошуку демонструє високу ефективність у знаходженні глобальних екстремумів цільової функції. Такий підхід підвищує шанси на отримання оптимальних рішень у складних умовах багатоекстремальної оптимізації, що підтверджується результатами експериментальних досліджень [8, 9, 23-26].

2.3 Правдоподібна кластеризація масивів даних на основі алгоритму «Сірих вовків»

Цільова функція правдопоодібної нечіткої кластеризації може бути записана у вигляді

$$E\left(Cred_q\left(k\right),Cl_q\right)=\sum_{k=1}^{N}\sum_{q=1}^{m}Cred_q^{\beta}\left(k\right)d^2\left(x_k,Cl_q\right) \qquad (2.1)$$

з урахуванням обмежень $0\le Cred_q\left(k\right)\le 1\forall q,k;$ $\sup Cred_q(k)\ge 0,5\forall k;$ $Cred_q\left(k\right)+\sup Cred_l\left(k\right)=1.$

Пакетний алгоритм правдоподібної нечіткої кластеризації можна записати у вигляді

$$\begin{cases} u_q^{(t+1)}(k) = \left(1 + d^2\left(x_k, Cl_q^{(t)}\right)\right)^{-1} \\ u*_q^{(t+1)}(k) = U_q^{(t+1)}(k)\left(\sup u_l^{(t+1)}(k)\right)^{-1} \\ Cred_q^{(t+1)}(k) = \frac{1}{2}\left(u_q^{*(t+1)}(k) + 1 - \sup_{l \neq q} u_l^*(k)\right) \\ Cl_q^{(t+1)} = \sum_{k=1}^{N}\left(Cred_q^{(t+1)}(k)\right)^{\beta} x_k \left(\sum_{k=1}^{N}\left(Cred_q^{t+1}(k)\right)^{\beta}\right)^{-1}. \end{cases} \quad (2.2)$$

Також можна записати цей метод ще й у рекурентній онлайн формі

$$\begin{cases} \sigma_q^2(k+1) = \left(\sum_{\substack{l=1 \\ l \neq q}}^{m} \left\| x_{k+1} - Cl_l(k) \right\|^{-2}\right)^{-1}, \\ u_q(k+1) = \left(1 + \frac{\left\| x_{k+1} - Cl_q(k) \right\|^2}{\sigma_q^2(k+1)}\right)^{-1}, \\ \widehat{u}_{(k+1)} = u_q(k+1)\left(\sup u_l(k+1)\right)^{-1}, \\ Cred_q(k+1) = \frac{1}{2}\left(u_q^*(k+1) + 1 - \sup_{l \neq q} \widehat{u}_l(k+1)\right), \\ Cl_q(k+1) = Cl_q(k) + \eta(k+1) Cred_q^{\beta}(k+1)\left(x_{k+1} - Cl_q(k)\right). \end{cases} \quad (2.3)$$

Для пошуку глобального екстремума функції (2.1), доцільно використовувати біоінспіровані еволюційні алгоритми оптимізації роїв частинок [6], серед яких в якості одного з найбільш швидкодіючих можна відзначити, так званий, алгоритм «Сірого вовка» (GWO) [38].

Згідно алгоритму, запропонованому в [38], сірі вовки живуть разом і полюють групами.

Процес пошуку та полювання можна описати так наступним чином:

$$C_{\alpha} = \left| B_1 * GW_{\alpha} - X(t) \right|,$$

$$C_\beta = \left| B_2 * GW_\beta - X(t) \right|,$$

$$C_\delta = \left| B_3 * GW_\delta - X(t) \right|,$$

якщо видобуток знайдено, вони спочатку вистежують, переслідують і наближаються до неї.

Якщо здобич біжить, тоді сірі вовки переслідують, оточують і спостерігають за здобиччю, поки вона не перестане рухатися, що можна записати у вигляді

$$GW_1 = GW_\alpha - A_1 * C_\alpha,$$

$$GW_2 = GW_\beta - A_2 * C_\beta,$$

$$GW_3 = GW_\delta - A_3 * C_\delta,$$

де GW – функція руху вовка.

Після того, як здобич знайдена, починається атака:

$$GW(t) = \frac{GW_1 + GW_2 + GW_3}{3}.$$

Параметри A та B є комбінаціями керуючого параметра α та випадкових чисел r_1 та r_2 [38]:

$$A = 2\alpha r_1 - \alpha, \tag{2.4}$$

$$B = 2r_2. \tag{2.5}$$

Якщо $|A|>1$, сірі вовки втікають від домінантів, а це означає, що омега-вовки втечуть від здобичі та досліджуватимуть більше простору; якщо $|A|<1$ вони наближаються до домінант, а значить δ – вовки будуть слідувати за домінантами, які наближаються до здобичі. В таблиці 2.1 наведено псевдокод роботи алгоритму.

Таблиця 2.1 – Псевдокод роботи алгоритму «Сірих вовків»

Опис	**Псевдокод**
Параметри налаштування	Вибірка даних Популяція зграї Контрольний параметр Критерій зупину
Ініціалізація	Початкові позиції сірих вовків-домінант
Пошук	Якщо це не критерій зупинки, то обчислення нового значення фітнес-функції Оновлення позиції Обмеження коло позицій вовків Оновити α, β і δ Оновити критерій зупинки Кінець

Щоб досягти належного компромісу між розвідкою та полюванням, пропонується покращений алгоритм «Сірих вовків» (best mode grey wolf optimization – BMGWO).

Зміна позицій вовків описана наступними виразами:

$$C=\left|B*X_P(t)-X(t)\right|, \tag{2.6}$$

$$X\left(t+1\right)=X_P(t)-A*C, \tag{2.7}$$

де X_P – позиція здобичі;

X – позиція вовка.

Вовк у позиції (X, Y) (рис. 2.2) може відтворити свою позицію навколо здобичі згідно з формулами оновлення (2.6) та (2.7).

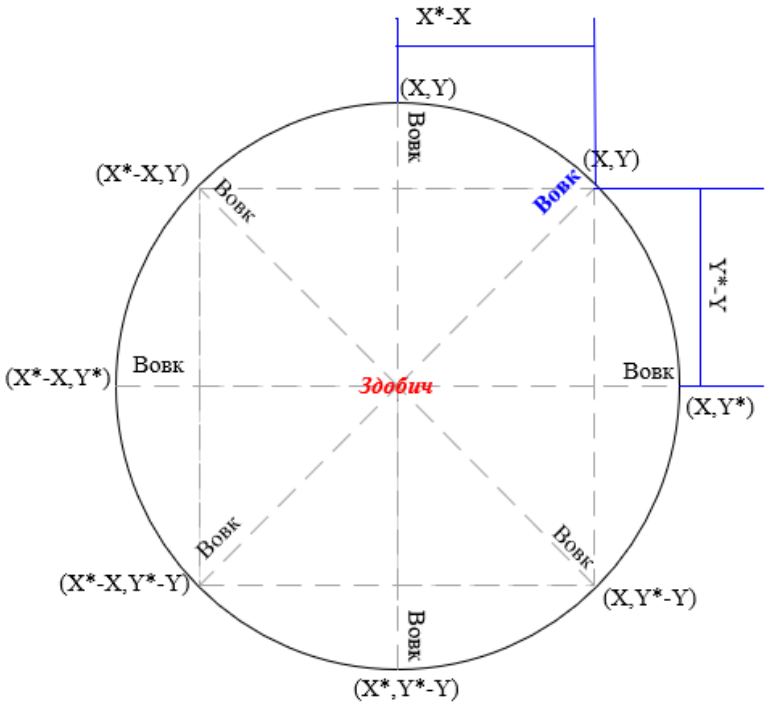

Рисунок 2.2 – Схема роботи алгоритму сірих вовків

Для цього, на етапі параметрів налаштування, задаються початкові позиції вовків – домінант

$$Cl_1 = C_\alpha;$$

$$Cl_2 = C_\beta; \text{ при } t = 0.$$

$$Cl_3 = C_\delta.$$

Беручи за початкові позиції центри кластерів, знайдених за допомогою метода можливісної нечіткої кластеризації.

2.4 Експериментальні дослідження

Експериментальні дослідження методу проводились на основі багатоекстремальної функції Аклі (рис. 2.3).

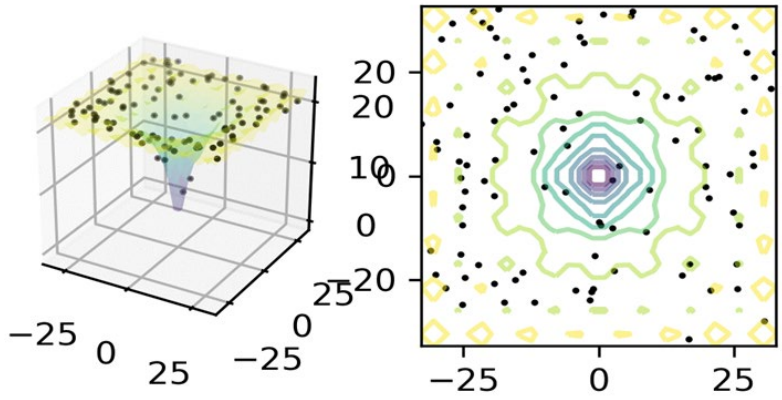

Рисунок 2.3 – Функція Аклі, на якій розташовані агенти-вовки

Порівняльний аналіз проводився з відомими алгоритмами рояового інтелекту, такими як PSO (Particle Swarm Optimization) та GWO (Grey Wolf Optimizer).

В таблиці 2.2 наведений порівняльний аналіз отриманих результатів.

Таблиця 2.2 – Порівняльна статистика методів

Точність	**CGWO**	**GWO**	**PSO**
Найкраща	1,5099e – 017	7,5495e – 013	0,0035
Середня	1,2204e – 016	1,0048e – 012	0,0055
Найгірша	2,2204e – 014	1,4655e – 013	0,0082

В таблиці 2.2 наведено порівняльний аналіз роботи трьох методів рояового інтелекту, які були обрані для проведення симуляційних контрастних експериментів із запропонованим правдоподібним методом нечіткої кластеризації на основі модифікованого алгоритму «Сірих вовків», щоб перевірити його перевагу щодо швидкості конвергенції та точності

пошуку.

Такий підхід, об’єднує в собі два методи правдоподібну нечітку кластеризацію даних та еволюційний алгоритм «Сірих вовків».

Для успішної реалізації гібридного методу правдоподібної кластеризації система повинна відповідати ряду ключових вимог, що забезпечують її ефективність, гнучкість та коректність у роботі:

– обробка великих обсягів даних у реальному часі. Алгоритм повинен бути оптимізованим для швидкої обробки потокових даних з мінімальними затримками, забезпечуючи кластеризацію як у пакетному режимі, так і в режимі онлайн;

– висока точність кластеризації. Система повинна забезпечувати точне виявлення кластерів шляхом використання гібридного підходу, що поєднує правдоподібну нечітку кластеризацію та еволюційні алгоритми. Важливо, щоб вона могла адаптуватися до різних структур даних та ефективно знаходити кластери навіть у складних умовах багатоекстремальності цільової функції;

– налаштування параметрів. Користувач повинен мати можливість налаштовувати основні параметри кластеризації, такі як кількість кластерів, метрики відстані, параметри фазифікації тощо, для оптимізації процесу кластеризації під конкретні завдання;

– візуалізація результатів. Система повинна мати зручні та інтуїтивні інструменти для візуалізації результатів кластеризації, включно з відображенням кластерів і їх центроїдів на різноманітних графіках та діаграмах. Це полегшить аналіз та виявлення закономірностей у даних;

– аналіз якості кластеризації. Система повинна пропонувати розширені інструменти для оцінки якості кластеризації, зокрема метрики, такі як індекс Данна та індекс Девіса-Болдіна. Це дозволить користувачам оцінювати розподіл елементів у кластерах і визначати ступінь їх належності;

– стійкість до неповних і зашумлених даних. Система має бути здатною обробляти неповні або зашумлені дані, зберігаючи стійкість результатів кластеризації до викидів та відсутніх значень. Для цього необхідно впровадити

передові механізми передобробки даних, такі як нормалізація та робастні методи кластеризації.

Загалом, система для реалізації гібридного методу правдоподібної кластеризації повинна бути універсальною, гнучкою і потужною, здатною адаптуватися до різноманітних завдань і забезпечувати користувачам глибокий аналіз кластерів.

2.5 Роль вибірки при реалізації та аналізі методу

Для реалізації гібридного методу правдоподібної кластеризації вибірка даних відіграє вирішальну роль, оскільки саме вона впливає на точність і ефективність кластеризації, а також на інтерпретацію результатів. Правильний вибір типу вибірки залежить від специфіки задачі і визначає, які аспекти кластеризації будуть акцентовані. Ось основні типи вибірок та їх характеристики:

– великі вибірки (Big Data). Для великих обсягів даних важливо забезпечити швидкість і масштабованість алгоритму. Хоча обробка великих вибірок може бути ресурсомісткою, вона забезпечує вищу точність кластеризації, оскільки дані повніше відображають реальну структуру. Однак, при роботі з великими вибірками важливо використовувати оптимізовані алгоритми й ефективні підходи до зменшення вимог до пам’яті;

– малі вибірки. Кластеризація на малих вибірках може бути більш легкою з точки зору ресурсів, але має ризик недостатньої статистичної репрезентативності. Використання малих вибірок може призвести до неточних висновків, якщо дані не достатньо повно відображають структуру явища;

– збалансовані вибірки. Ці вибірки містять однакову кількість елементів для кожного класу або категорії, що полегшує аналіз і забезпечує рівні умови для кластеризації. Однак, не завжди можливо отримати

збалансовану вибірку з реальних даних;

– незбалансовані вибірки. Часто вибірки реальних даних є незбалансованими, де деякі кластери можуть бути представлені більшою кількістю елементів. Це ускладнює завдання кластеризації, оскільки алгоритми можуть зміщуватися в бік більших кластерів. У таких випадках важливо використовувати спеціальні методи для роботи з незбалансованими даними;

– шумні та неповні вибірки. В реальних умовах дані можуть містити викиди, шум або бути неповними. При роботі з такими вибірками необхідно застосовувати методи робастної кластеризації, які здатні зменшувати вплив шуму та коректно обробляти відсутні дані;

– структуровані та неструктуровані дані. Структуровані вибірки, такі як таблиці або бази даних, дозволяють легше застосовувати алгоритми кластеризації. Неструктуровані дані, наприклад текстові або мультимедійні, вимагають попередньої обробки (наприклад, текстової векторизації) для подальшого застосування кластеризаційних методів;

– вибірки зі змішаними типами даних. У випадках, коли вибірка містить як числові, так і категоріальні дані, необхідно адаптувати методи кластеризації, що можуть обробляти такі різнорідні дані (наприклад, методи, що підтримують як метрики відстані для числових даних, так і функції схожості для категоріальних).

Використання реальних датасетів, таких як набори даних з ресурсів UCI Machine Learning Repository або Kaggle, є ефективним підходом для оцінки гібридного методу кластеризації у реальних умовах. Основні переваги цих вибірок полягають у різноманітті наборів даних із різних галузей, що дозволяє тестувати метод на різних типах інформації, таких як текстові, числові, часові ряди тощо. Це допомагає зрозуміти, наскільки добре метод справляється із завданнями в реальному середовищі та оцінити його практичну користь.

2.6 Вибір програмного середовища та мови програмування

Для реалізації методу правдоподібної кластеризації є мова програмування Python. Цей вибір зумовлений кількома ключовими факторами, які роблять Python ідеальним інструментом для розробки комплексних дослідницьких проектів в галузі обробки даних та машинного навчання.

Для реалізації гібридного методу правдоподібної кластеризації існує кілька бібліотек Python, які можуть бути особливо корисними:

– бібліотека NumPy пропонує потужні засоби для ефективної роботи з масивами даних, підтримуючи широкий спектр математичних операцій, необхідних для обробки та аналізу даних. Використання NumPy дозволяє оптимізувати виконання численних операцій, що є критично важливим для обробки великих наборів даних;

– бібліотека SciPy розширює можливості NumPy, надаючи додаткові модулі для оптимізації, статистики та обробки сигналів. Це може бути корисно для реалізації складніших алгоритмічних процедур, пов'язаних з гібридним методом правдоподібної кластеризації;

– для роботи з машинним навчанням і кластеризацією даних велику роль відіграє бібліотека Scikit-learn. Вона включає в себе велику кількість готових до використання алгоритмів кластеризації, класифікації, регресії та зниження розмірності. Scikit-learn має добре документований API та інтегрується з NumPy та SciPy, що робить її зручним вибором для розробки комплексних дослідницьких проєктів;

– для візуалізації результатів кластеризації можна використовувати бібліотеку Matplotlib, яка дозволяє створювати графіки та діаграми високої якості. Вона надає гнучкі засоби для представлення даних в найзручнішому для аналізу вигляді. Крім того, бібліотека Seaborn, яка побудована на базі Matplotlib, надає додаткові можливості для створення статистичних графіків.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

3.1 Програмна реалізація

Програмна реалізація методу здійснена за допомогою мови програмування Python та фреймворку Dash для створення інтерактивного веб-застосунку. Основні компоненти програми включають файл app.py, модулі для роботи з базою даних MongoDB, а також модулі, що реалізують алгоритми.

Файл app.py є головним скриптом, який відповідає за запуск веб-застосунка та забезпечення взаємодії між користувачем і внутрішніми модулями програми. У ньому містяться необхідні імпорти бібліотек, налаштування MongoDB, визначення ключових функцій і створення користувацького інтерфейсу. Нижче (рис. 3.1) наведено фрагмент коду, який демонструє початкове налаштування MongoDB та ініціалізацію основних параметрів програми.

```
import numpy as np
from bson import ObjectId
import plotly.graph_objs as go
import dash
from dash import Dash, dcc, html, Input, Output, State
from gray_wolf_algorithm import CGWOAlgorithm, ObjectiveFunction, create_figure
from datetime import datetime
from pymongo import MongoClient
import json
```

Рисунок 3.1 – Імпорт пакетів

Основні функції, реалізовані у файлі app.py, включають завантаження даних з бази даних (рис. 3.2), налаштування параметрів алгоритмів, запуск обчислювальних процесів та відображення результатів у вигляді графіків і таблиць.

```
# MongoDB setup
client = MongoClient("mongodb://localhost:27017/")
db = client['gwo_database']
params_collection = db['params']
functions_collection = db['functions']
history_collection = db['history']

# Placeholder for algorithm parameters and history
algorithm_parameters = {'iterations': 10, 'wolves': 10, 'param_variations': [], 'objective_functions': []}
history = []
```

Рисунок 3.2 – Підключення до БД

app.py здійснює взаємодію з базою даних MongoDB для зберігання та отримання даних про параметри алгоритмів і результати їх виконання. Використовуючи бібліотеку pymongo, програма підключається до бази даних, виконує запити і обробляє відповіді. Це забезпечує гнучке налаштування алгоритмів, оскільки користувач може змінювати параметри через веб-інтерфейс, і ці зміни автоматично фіксуються в базі даних.

Для реалізації алгоритмів використовуються окремі модулі, наприклад, gray_wolf_algorithm.py, який містить реалізацію алгоритму оптимізації «Сірих вовків». У цьому модулі визначені класи і методи для виконання основної логіки алгоритму та його модифікацій. На рисунках 3.3 та 3.4 наведено приклад коду модуля.

```
class CGWOAlgorithm:
    def __init__(self, objective_function, lb, ub, wolves=10, iterations=100):
        self.objective_function = objective_function
        self.lb = lb
        self.ub = ub
        self.wolves = wolves
        self.iterations = iterations
        self.alpha, self.beta, self.delta = None, None, None
        self.population = self.initialize_population()
```

Рисунок 3.3 – Клас CGWOAlgorithm

```
def initialize_population(self):
    return np.random.uniform(self.lb, self.ub, (self.wolves, 2))

def optimize(self):
    for iter in range(self.iterations):
        for wolf in self.population:
            fitness = self.objective_function(wolf[0], wolf[1])
            # Оновлення альфа, бета та дельта вовків
            # Логіка оптимізації
    return self.alpha, self.beta, self.delta
```

Рисунок 3.4 – Ініціалізація популяції зграї вовків

Веб-застосунок, розроблений з використанням Dash, має кілька вкладок, які надають користувачеві різноманітні функції. Наприклад, вкладка "Test Algorithm" дає можливість користувачу налаштовувати параметри алгоритму та запускати його для виконання обчислень. Нижче наведено приклад коду, який створює інтерфейс цієї вкладки (рис. 3.5).

```
def load_data():
    global algorithm_parameters, history
    algorithm_parameters['param_variations'] = list(params_collection.find())
    algorithm_parameters['objective_functions'] = list(functions_collection.find())
    history.extend(list(history_collection.find()))

    if not algorithm_parameters['param_variations']:
        default_variations = [
            {'name': 'GWO no crazy', 'y': 0, 'd': 0, 'sigma2': 0},
            {'name': 'default1', 'y': 0.5, 'd': 0.1, 'sigma2': 0.5},
            {'name': 'default2', 'y': 0.4, 'd': 0.2, 'sigma2': 0.3}
        ]
```

Рисунок 3.5 – Завантаження вибірки

Завантаження даних із бази даних відбувається за допомогою функції load_data(), яка отримує параметри алгоритмів та функції з колекцій MongoDB і зберігає їх у глобальні змінні. Це дозволяє програмі динамічно змінювати параметри на основі збережених даних, а також зберігати історію виконання алгоритмів для подальшого аналізу.

Для створення інтерфейсу користувача в застосунку використовується бібліотека Dash, яка дозволяє розробляти веб-застосунки з інтерактивною візуалізацією на основі Plotly. Dash – це потужний та гнучкий інструмент, що дає змогу легко і швидко будувати складні графічні інтерфейси з мінімальними витратами часу та зусиль.

Розглянемо детально процес створення інтерфейсу користувача застосунку, звертаючи увагу на організацію вкладок і їх функціональність. Dash надає широкий спектр можливостей для створення елементів інтерфейсу, таких як вкладки, форми для введення даних, графіки та інші інтерактивні компоненти.

В застосунку зроблено кілька вкладок, кожна з яких виконує свою специфічну функцію. Вкладки допомагають структурувати інтерфейс та роблять навігацію зручнішою для користувачів, дозволяючи їм зосереджуватися на конкретних завданнях, які вони хочуть виконати.

Застосунок містить кілька вкладок, кожна з яких виконує свою унікальну функцію:

– Test Algorithm. Ця вкладка дозволяє користувачам тестувати алгоритм, задавати його параметри та запускати процес оптимізації. Інтерфейс містить поля введення для кількості ітерацій і кількості «вовків», а також випадаючі списки для вибору варіантів параметрів та об’єктивної функції (рис. 3.6);

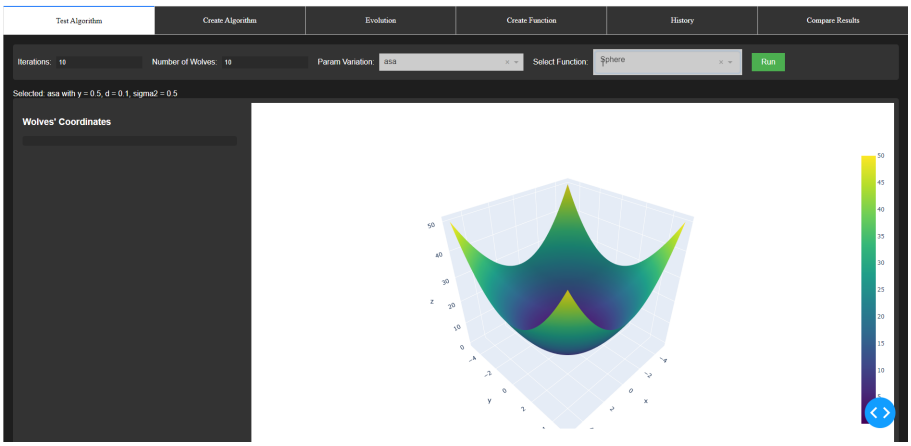

Рисунок 3.6 – Вкладка «Test Algorithm»

– Create Algorithm. У цій вкладці користувачі можуть додавати нові варіації параметрів для алгоритму, зберігати їх у базі даних та переглядати вже створені варіанти (рис. 3.7);

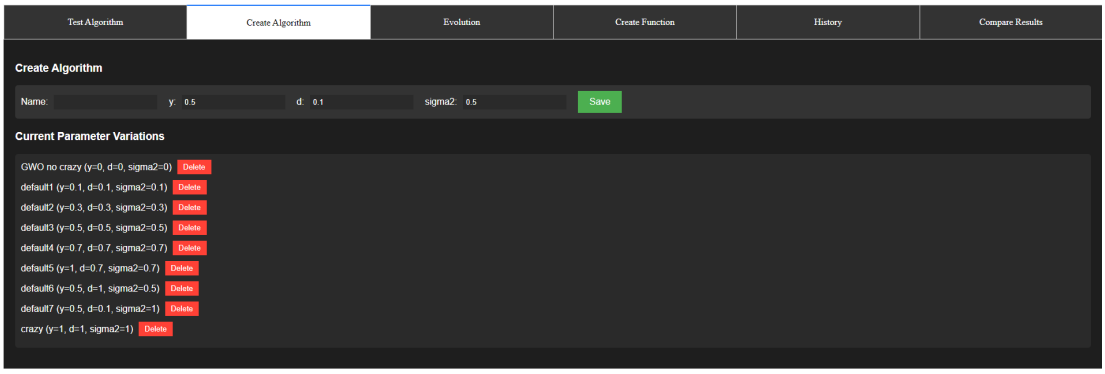

Рисунок 3.7 – Вкладка «Create Algorithm»

– Evolution. Вкладка для виконання еволюційної оптимізації, що дозволяє користувачам налаштовувати параметри, такі як кількість популяції та ітерацій;

– Create Function. Тут користувачі можуть створювати нові об’єктивні функції. Інтерфейс дозволяє задавати код функції, її межі, а також візуалізувати результати;

– History. Ця вкладка відображає історію запусків алгоритму, включаючи деталі параметрів і результати попередніх виконань;

– Compare Results. Вкладка для порівняння результатів різних запусків алгоритму, з візуалізацією даних на графіках для полегшення аналізу.

Такий поділ функціональних можливостей забезпечує зручність для користувачів і дозволяє легко орієнтуватися в інтерфейсі, виконуючи різні завдання.

Після запуску алгоритму користувач має можливість стежити за його прогресом за допомогою динамічних графіків та відображення деталей параметрів у реальному часі. Це дає змогу миттєво оцінювати ефективність обраних параметрів та швидко вносити зміни для досягнення оптимальних

результатів. Такий підхід забезпечує інтуїтивний і зручний спосіб взаємодії з алгоритмом.

Вкладка «Create Algorithm» дозволяє користувачам створювати нові варіації параметрів для алгоритма оптимізації. Інтерфейс цієї вкладки включає форми для введення назви варіації та значень параметрів, таких як *y, d* та *sigma2*. Після введення потрібних значень, користувач може зберегти нову варіацію, яка буде збережена в базі даних і стане доступною для використання в інших вкладках застосунку.

Вкладка «Evolution» (рис. 3.8) надає користувачам можливість виконувати еволюційну оптимізацію. Інтерфейс цієї вкладки містить налаштування для таких параметрів, як розмір популяції, кількість ітерацій та інші ключові параметри, пов’язані з еволюційним процесом. Після того, як користувач задасть необхідні параметри, він може запустити процес еволюції. Цей процес здійснює оптимізацію параметрів алгоритму, проходячи через кілька поколінь, поступово покращуючи результати. Такий підхід дозволяє ефективно досягати оптимальних значень параметрів за допомогою еволюційного підходу.

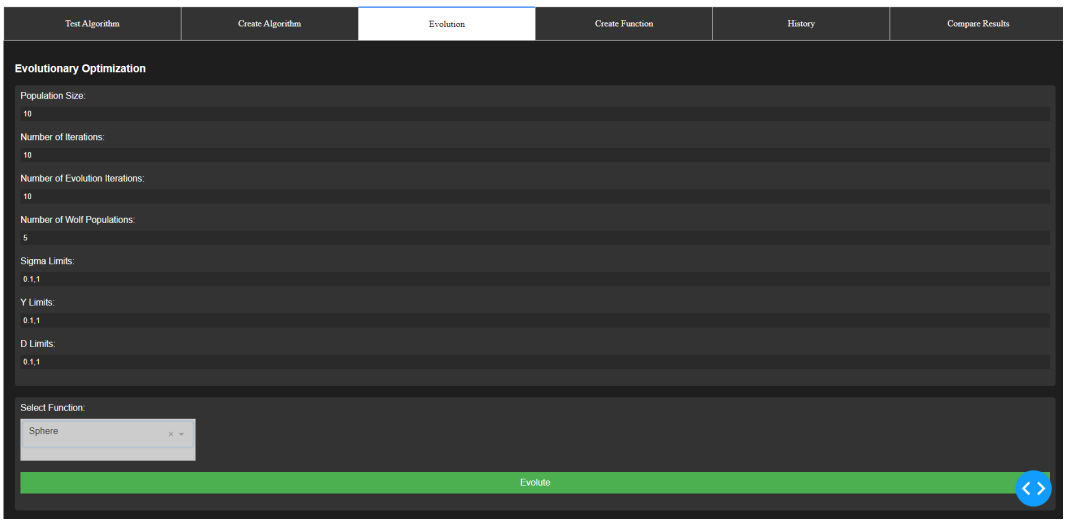

Рисунок 3.8 – Вкладка «Evolution»

Результати еволюційної оптимізації представлені у вигляді графіків та списків, які демонструють найкращі виявлені параметри. Це дозволяє

користувачам порівнювати різні популяції і знаходити найбільш ефективні варіації параметрів для їхніх завдань.

Вкладка «Evolution» надає потужний інструмент для вдосконалення алгоритму через ітеративний процес навчання.

Вкладка «Create Function» (рис. 3.9) пропонує інтерфейс для розробки нових об’єктивних функцій, які можна використовувати для тестування та оптимізації. Користувачі мають можливість ввести назву функції, її математичний вираз і встановити межі значень для параметрів. Після введення цих даних користувачі можуть зберегти функцію, і вона стане доступною для використання в інших розділах програми.

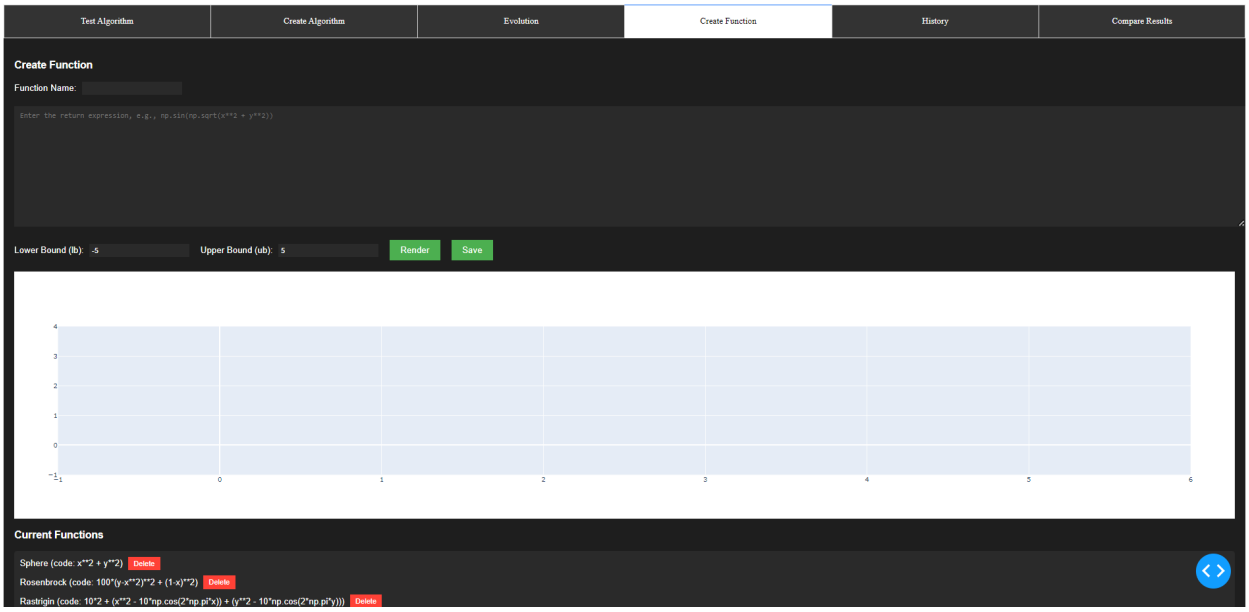

Рисунок 3.9 – Вкладка «Create Function»

Вкладка «History» (рис. 3.10) демонструє історію всіх запусків алгоритму, включаючи деталі параметрів і результати кожного з них. Користувачі можуть переглядати попередні запуски, аналізувати ефективність різних параметрів і порівнювати результати. Ця вкладка сприяє відстеженню прогресу та дозволяє робити висновки на основі минулого досвіду.

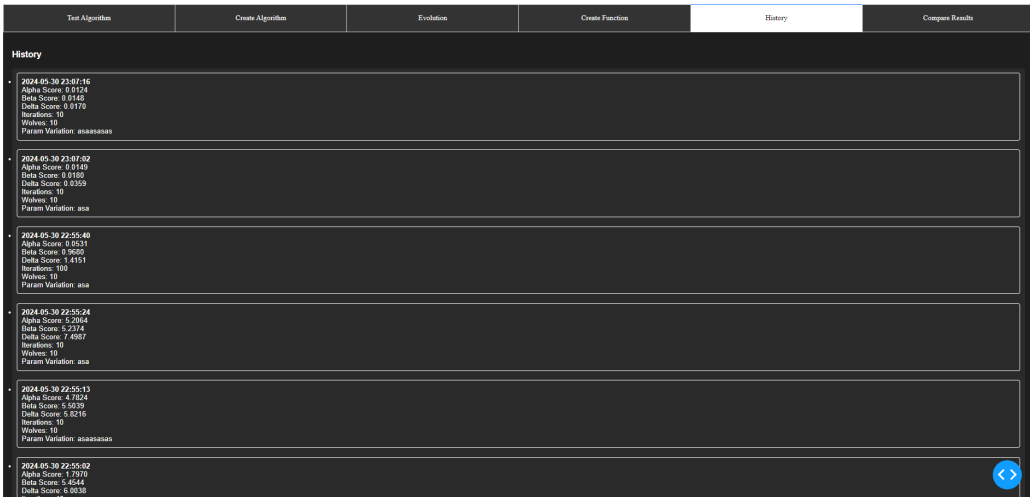

Рисунок 3.10 – Вкладка «History»

Завдяки функціоналу історії, користувачі можуть зберігати результати своїх експериментів і легко повертатися до них у майбутньому. Це особливо корисно для довгострокових проектів, де важливо зберігати інформацію про всі попередні спроби оптимізації та аналізувати їх для подальшого вдосконалення алгоритму.

Вкладка «Compare Results» (рис. 3.11) дозволяє користувачам порівнювати результати різних запусків алгоритму. Інтерфейс включає випадаючі списки для вибору запусків, які потрібно порівняти, а також кнопку для виконання порівняння. Результати відображаються у вигляді графіків, що дає змогу користувачам легко візуалізувати та аналізувати відмінності між різними параметрами.

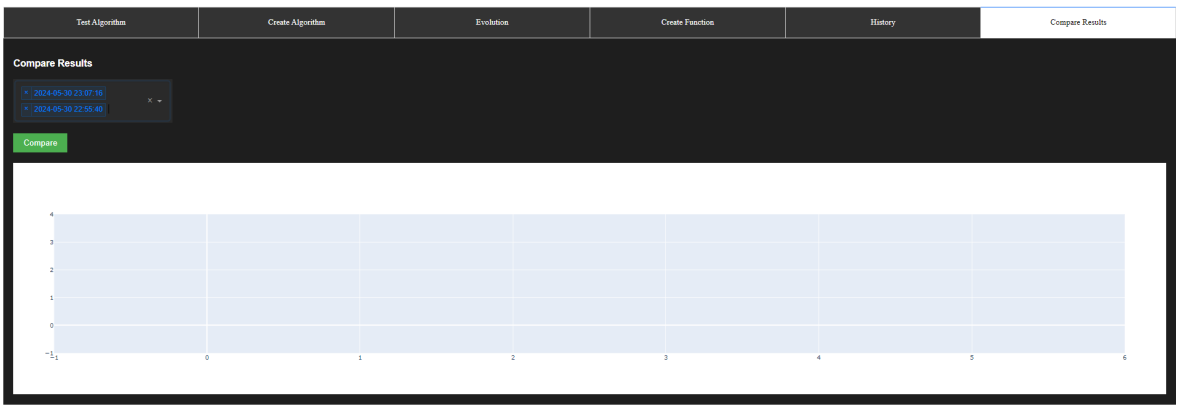

Рисунок 3.11 – Вкладка «Compare Results»

Інтерфейс користувача, розроблений на базі Dash, забезпечує потужні можливості для дослідження та тестування алгоритмів оптимізації. Використовуючи вкладку «Test Algorithm», користувачі можуть експериментувати з різними параметрами алгоритму, спостерігаючи за його поведінкою в реальному часі. Це дозволяє швидко оцінювати ефективність параметрів, вносити корективи та миттєво бачити результати на графіках. Можливість візуалізації процесу оптимізації сприяє кращому розумінню, як змінюються показники ефективності в залежності від введених параметрів.

Вкладка «Evolution» розширює ці можливості, дозволяючи користувачам застосовувати еволюційний підхід для автоматичної оптимізації параметрів. Ця вкладка дозволяє запускати багаторазові ітерації з різними налаштуваннями, зберігаючи результати кожної спроби. Користувачі можуть спостерігати, як алгоритм адаптується та покращує свої параметри з кожним поколінням, що є особливо корисним для складних задач оптимізації, де вручну визначити оптимальні параметри є складно.

Функціональність вкладок «History» та «Compare Results» розширює можливості для аналізу. Користувачі можуть переглядати історію всіх запусків, що дає змогу зберігати та аналізувати великий обсяг даних про попередні експерименти (рис. 3.12). Вкладка «Compare Results» (рис. 3.13) спрощує процес порівняння різних запусків, візуалізуючи їх результати на графіках. Це допомагає виявити найбільш ефективні варіації параметрів та приймати обґрунтовані рішення щодо подальшого вдосконалення алгоритму.

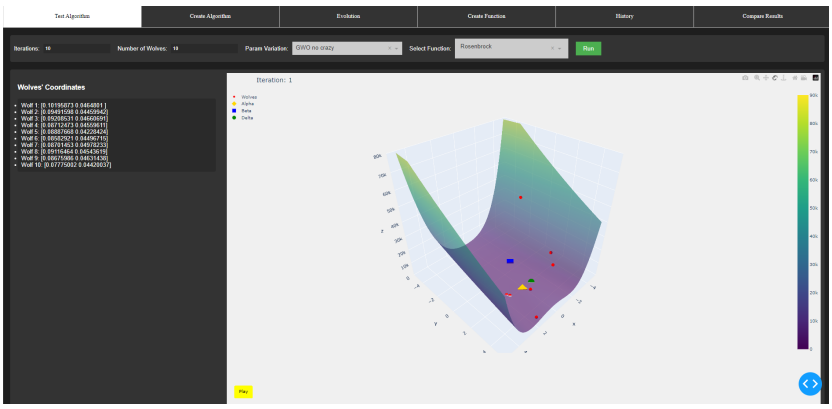

Рисунок 3.12 – Дослідження методу

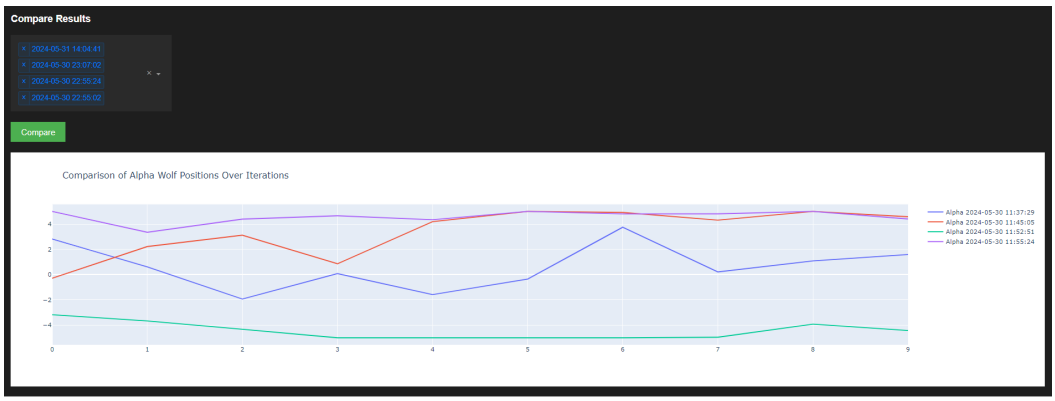

Рисунок 3.13 – Порівняння отриманих результатів

Таким чином, цей інтерфейс надає користувачам інтуїтивно зрозумілі інструменти для глибокого аналізу та оптимізації алгоритмів. Користувачі можуть досліджувати різні підходи, швидко адаптувати параметри, зберігати та порівнювати результати, що значно спрощує процес вдосконалення алгоритмів і робить його більш ефективним і наочним.

3.2 Реалізація основних функцій методу

Основна функція для завантаження даних з MongoDB в застосунку називається load_data. Ця функція завантажує параметри алгоритму, об’єктивні функції та історію з відповідних колекцій бази даних MongoDB. Взаємодія з колекціями params, functions і history дозволяє зберігати та отримувати дані, необхідні для налаштування і виконання алгоритму (рис. 3.14).

```
client = MongoClient("mongodb://localhost:27017/")
db = client['gwo_database']
params_collection = db['params']
functions_collection = db['functions']
history_collection = db['history']

# Placeholder for algorithm parameters and history
algorithm_parameters = {'iterations': 10, 'wolves': 10, 'param_variations': [], 'objective_functions': []}
history = []
```

Рисунок 3.14 – З’єднання з MongoDB

Функція load_data ініціалізує з’єднання з MongoDB (рис. 3.15) і виконує запити до колекцій, щоб отримати актуальні варіації параметрів та об’єктивних функцій. Якщо жодних варіацій не знайдено, функція додає стандартні варіації та функції у відповідні колекції.

Цей код забезпечує завантаження параметрів та функцій з MongoDB і збереження їх у глобальні змінні. Також передбачено додавання стандартних значень, якщо в базі даних немає записів. Це дозволяє зберігати та використовувати параметри і функції між різними сесіями роботи з застосунком.

```
# Load saved data from MongoDB
def load_data():
    global algorithm_parameters, history
    algorithm_parameters['param_variations'] = list(params_collection.find())
    algorithm_parameters['objective_functions'] = list(functions_collection.find())
    history.extend(list(history_collection.find()))

    # Add default parameter variations if none are found in the database
    if not algorithm_parameters['param_variations']:
        default_variations = [
            {'name': 'GWO no crazy', 'y': 0, 'd': 0, 'sigma2': 0},
            {'name': 'default1', 'y': 0.5, 'd': 0.1, 'sigma2': 0.5},
```

Рисунок 3.15 – Функція load_data

Основні обчислювальні функції, пов’язані з алгоритмом сірого вовка (GWO), реалізовані в окремих модулях (рис. 3.16). Алгоритм сірого вовка є методикою оптимізації, яка імітує стратегії полювання та соціальну ієрархію справжніх сірих вовків. Модифікація цього алгоритму, відома як Crazy GWO (CGWO), вводить додатковий фактор випадковості, що допомагає уникати локальних мінімумів і підвищує здатність алгоритму до глобальної оптимізації.

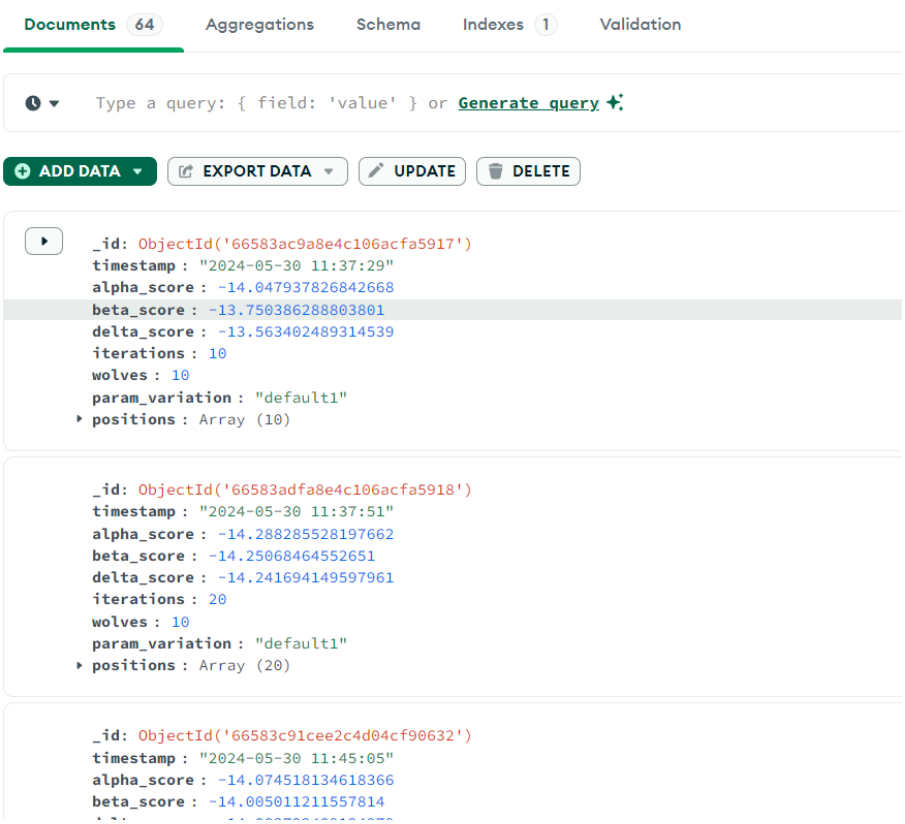

Рисунок 3.16 – Записи в MongoDB за допомогою Mongo Compass

Основні функції для CGWO включають обчислення об’єктивної функції, оновлення позицій вовків та механізм випадкової збуреності для покращення пошуку (рис. 3.17).

```
class CGWOAlgorithm:
    def calculate_new_position(self, a, Xp_t, X_t):
        r1, r2 = np.random.rand(2)
        A = 2 * a * r1 - a
        B = 2 * r2
        C = abs(B * Xp_t - X_t)
        return Xp_t - A * C
    def update_wolves_positions_crazy(self, X_t, Xp_alpha, Xp_beta, Xp_delta, n_iterations, t, y=0, d=0, sigma
        n_wolves, dim = X_t.shape
        a = 2.0 - t * (2.0 / n_iterations)
        E_r = np.zeros_like(X_t)
        for i in range(n_wolves):
```

Рисунок 3.17 – Клас CGWOAlgorithm

Головна відмінність CGWO від звичайного GWO полягає в тому, що додаткові параметри *y*, *d* та σ^2 використовуються для введення випадкових

змін до позицій вовків, що дозволяє алгоритму краще досліджувати простір пошуку.

Налаштування параметрів алгоритму є критично важливим етапом для забезпечення його ефективності. У застосунку параметри можуть бути збережені та завантажені з MongoDB, що дозволяє користувачам зберігати свої експерименти та швидко відновлювати налаштування.

Збережені варіанти параметрів включають кількість ітерацій, кількість вовків, а також специфічні параметри, такі як *y, d* та σ^2, які визначають ступінь випадковості у CGWO. Користувачі можуть зберігати різні варіанти налаштувань, експериментувати з ними та аналізувати результати для вибору найбільш ефективних.

Функція load_data забезпечує завантаження цих параметрів з бази даних, а інтерфейс дозволяє користувачам легко обирати та застосовувати збережені варіанти. Завдяки такій організації параметрів, користувачі можуть зберігати успішні конфігурації та швидко їх використовувати для подальших експериментів, що значно спрощує процес оптимізації та підвищує його ефективність.

Функції, які використовуються для оптимізації в алгоритмі, також зберігаються в MongoDB. Це дозволяє зберігати різні варіанти функцій і швидко їх використовувати в подальших експериментах. Кожна функція має назву, код для обчислення значення, а також нижню і верхню межі значень, в яких проводиться оптимізація.

Функції зберігаються в колекції functions у MongoDB. Коли додаток запускається, функція load_data завантажує всі наявні функції з цієї колекції та зберігає їх у глобальній змінній algorithm_parameters['objective_functions']. Це дозволяє користувачам обирати потрібну функцію з випадаючого списку у вкладці «Test Algorithm».

Клас ObjectiveFunction є ключовим компонентом для обчислення значень функції під час оптимізації. Цей клас приймає на вхід код функції та

її межі і дозволяє виконувати обчислення на основі переданих параметрів. На рисунку 3.18 наведено приклад реалізації цього класу.

```
class ObjectiveFunction:
    def __init__(self, name, code, lower_bound, upper_bound):
        self.name = name
        self.code = code
        self.lower_bound = lower_bound
        self.upper_bound = upper_bound

    def evaluate(self, *args):
        # Перевірка, чи параметри знаходяться в межах
        if all(self.lower_bound <= arg <= self.upper_bound for arg in args):
            return eval(self.code)
        else:
            raise ValueError("Parameters out of bounds.")
```

Рисунок 3.18 – Клас ObjectiveFunction

Цей клас приймає рядок коду функції, наприклад '*x***2 + *y***2', і виконує цей код для створення об'єктивної функції. Метод evaluate дозволяє обчислювати значення функції для заданих значень *x* та *y*.

Однією з збережених функцій у базі даних є функція сфери (Sphere). Ця функція використовується для тестування алгоритмів оптимізації завдяки своїй простоті та гладкості. Запис цієї функції в MongoDB виглядає наступним чином (рис. 3.19).

```
{
    "name": "Sphere",
    "code": "x**2 + y**2",
    "lb": 0,
    "ub": 50
}
```

Рисунок 3.19 – Запис функції в MongoDB

Коли ця функція завантажується в застосунок, вона перетворюється на об’єкт класу ObjectiveFunction, який використовується для обчислення значень під час оптимізації. Цей процес дозволяє застосунку використовувати функцію в оптимізаційних алгоритмах, зокрема, викликаючи метод evaluate, щоб обчислити значення для заданих параметрів.

Наприклад, після завантаження функції з MongoDB, код для створення об’єкта класу ObjectiveFunction може виглядати наступним чином (рис. 3.20).

```
# Припустимо, що дані з MongoDB вже завантажено в змінну sphere_function_data
sphere_function_data = {
    "name": "Sphere",
    "code": "x**2 + y**2",
    "lb": 0,
    "ub": 50
}

# Створення об'єкта ObjectiveFunction
sphere_function = ObjectiveFunction(
    name=sphere_function_data["name"],
    code=sphere_function_data["code"],
    lower_bound=sphere_function_data["lb"],
    upper_bound=sphere_function_data["ub"]
)

# Приклад обчислення значення функції для x=3 та y=4
result = sphere_function.evaluate(3, 4)  # Значення буде 25
```

Рисунок 3.20 – Приклад використання ObjectiveFunction

Користувачі можуть створювати нові функції через вкладку «Create Function», де вони вводять назву функції, її код та межі. Після збереження ці функції записуються в MongoDB і стають доступними для вибору у вкладці «Test Algorithm». Це забезпечує простий спосіб додавання нових функцій для тестування та оптимізації, дозволяючи при цьому зберігати попередні налаштування для подальшого використання (рис. 3.21).

Рисунок 3.21 – Функції в MongoDB за допомогою Mongo Compass

Модифікований алгоритм сірого вовка (Crazy GWO) вводить додаткову випадковість, що дозволяє алгоритму краще обстежувати простір пошуку та уникати локальних мінімумів. Це досягається завдяки додаванню випадкових збурень до позицій вовків під час кожної ітерації. Параметри, такі як *y, d* та σ^2, використовуються для контроля величини цих збурень. Наприклад, *y* визначає масштаб збурень, *d* відповідає за випадковість з нормального розподілу, а σ^2 контролює дисперсію.

Ця випадковість допомагає алгоритму уникати застрягання в локальних мінімумів і підвищує ймовірність знаходження глобального оптимуму, що робить CGWO більш ефективним у порівнянні зі звичайним GWO. Завдяки цій модифікації алгоритм стає більш гнучким та здатним до адаптації у складних середовищах оптимізації.

Таким чином, застосунок забезпечує потужний і гнучкий інтерфейс для роботи з різними варіантами параметрів та функцій, що дозволяє користувачам ефективно досліджувати та оптимізувати алгоритми на основі модифікованого CGWO.

3.3 Візуалізація результатів

Для візуалізації результатів роботи алгоритмів використано бібліотеку Plotly, яка дозволяє створювати інтерактивні графіки високої якості. Основним компонентом для відображення графіків у Dash є dcc.Graph. Цей компонент забезпечує зручну інтеграцію графіків у веб-застосунок та надає можливість взаємодії з ними.

Функція create_figure відповідає за створення графіків для візуалізації процесу оптимізації. Ця функція генерує тривимірні графіки, які відображають позиції вовків та значення об'єктивної функції на кожній ітерації. Для цього вона використовує клас ObjectiveFunction, що дозволяє створювати сітку значень для візуалізації функції.

Лістинг 3.1 Код створення графіків:

```
import plotly.graph_objects as go
import numpy as np

def create_figure(objective_function, x_range, y_range, wolves_positions):
    # Створення сітки значень для візуалізації
    x = np.linspace(x_range[0], x_range[1], 100)
    y = np.linspace(y_range[0], y_range[1], 100)
    X, Y = np.meshgrid(x, y)
    Z = np.array([[objective_function.evaluate(xi, yi) for xi in x] for yi in y])

    # Створення тривимірного графіка
    fig = go.Figure(data=[go.Surface(z=Z, x=X, y=Y, colorscale='Viridis', opacity=0.9)])

    # Додавання позицій вовків на графік
    for pos in wolves_positions:
```

```
            fig.add_trace(go.Scatter3d(x=[pos[0]], y=[pos[1]],
z=[objective_function.evaluate(pos[0], pos[1])],
                                mode='markers', marker=dict(size=5, color='red')))

            fig.update_layout(title='Optimization Process', scene=dict(xaxis_title='X',
yaxis_title='Y', zaxis_title='Objective Value'))
            return fig
```

Функція create_figure використовує дані про позиції вовків на кожній ітерації для створення анімації, яка демонструє, як змінюються позиції та значення об’єктивної функції в процесі оптимізації. Це дозволяє користувачам спостерігати за динамікою роботи алгоритму, наочно бачачи, як вдосконалюються позиції вовків та результати функції в ході ітерацій (рис. 3.22).

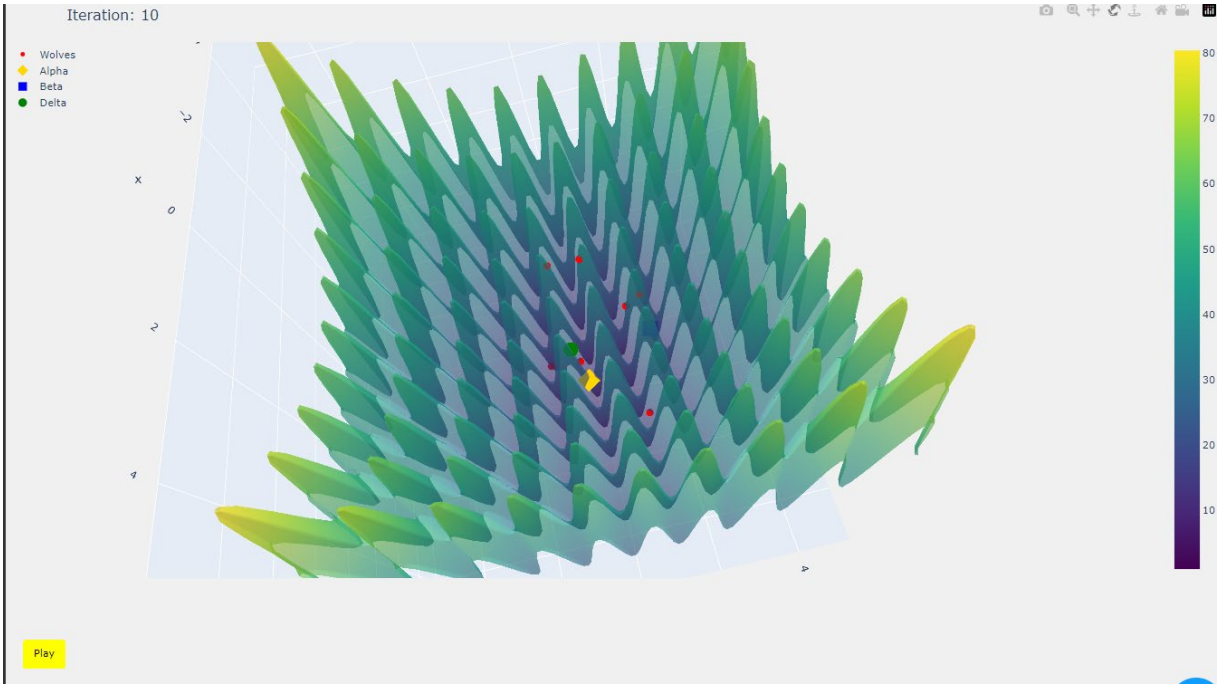

Рисунок 3.22 – Тривимірний графік позицій вовків

Результати обчислень та роботи алгоритмів відображаються у вкладці «Test Algorithm». Ця вкладка містить компонент dcc.Graph, який

використовується для візуалізації графіків, створених функцією create_figure. Коли користувач запускає алгоритм, позиції вовків та значення об'єктивної функції автоматично оновлюються на графіку, що дозволяє спостерігати результати в реальному часі.

Окрім графіків, у вкладці «Test Algorithm» також відображаються важливі дані, такі як координати вовків і статистика результатів (рис. 3.23). Це дозволяє користувачам отримати повне уявлення про результати оптимізації без необхідності переходити на інші вкладки.

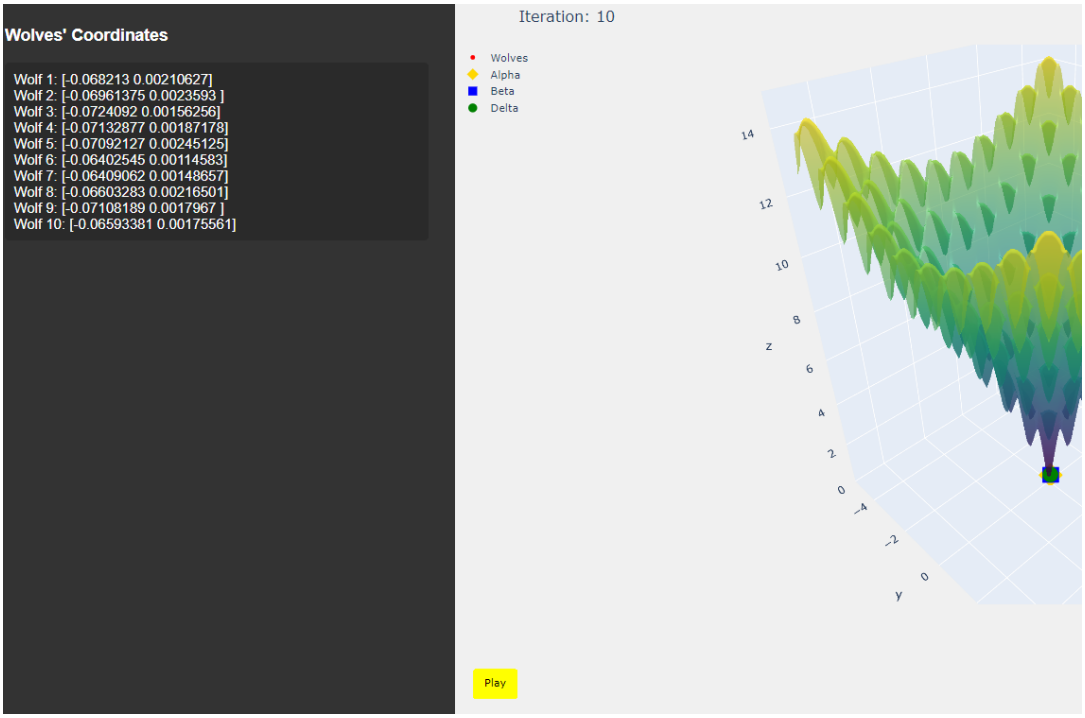

Рисунок 3.23 – Інтерфейс вкладки «Test Algorithm» з графіком і статистикою

Завдяки візуалізації результатів у реальному часі, користувачі можуть швидко аналізувати ефективність алгоритму, спостерігати за його поведінкою та оперативно вносити корективи до параметрів. Це робить процес оптимізації більш інтерактивним і зрозумілим, що є особливо важливим для складних алгоритмів, таких як Crazy GWO.

Процес створення графіків у розробленому застосунку реалізується за допомогою бібліотеки Plotly, що дозволяє створювати інтерактивні та інформативні візуалізації. Основна функція для побудови графіків називається

create_figure. Вона відповідає за генерацію тривимірного графіку, що відображає позиції вовків і значення об’єктивної функції на кожній ітерації.

Процес починається з ініціалізації даних для графіка. Функція отримує масив позицій вовків на кожній ітерації та об’єктивну функцію, яку потрібно візуалізувати. Крім того, вона створює сітку значень для обчислення та відображення поверхні об’єктивної функції. Це дає змогу візуально оцінити ефективність і поведінку алгоритму під час оптимізації.

Потім функція create_figure генерує початковий графік, що містить поверхню об’єктивної функції та позиції вовків на першій ітерації. Це здійснюється шляхом додавання різних об’єктів до графіка: поверхні для об’єктивної функції, точок для відображення позицій вовків, а також маркерів для позначення найкращих вовків (альфа, бета, дельта).

Для динамічного відображення процесу оптимізації функція створює анімаційні кадри. Кожен кадр представляє стан алгоритму на певній ітерації, демонструючи нові позиції вовків і оновлені значення об’єктивної функції. Ці кадри додаються до графіка, що дозволяє користувачам переглядати процес оптимізації у вигляді анімації.

Функція також створює меню управління анімацією, яке дозволяє користувачам запускати, зупиняти та контролювати відтворення анімації. Це реалізується за допомогою компонентів Plotly, які інтегруються в макет графіка. Меню управління забезпечує зручність та інтуїтивність використання, дозволяючи легко взаємодіяти з графіком.

Крім того, функція налаштовує параметри відображення графіка, такі як діапазони осей, кольорова шкала, розташування легенди та інші елементи макету. Це забезпечує зручне та зрозуміле представлення даних, що полегшує аналіз результатів.

Результати обчислень та роботи алгоритмів відображаються у вкладці «Test Algorithm». Вона містить інтерактивний графік, що оновлюється в реальному часі під час виконання алгоритму. Користувачі можуть спостерігати за змінами позицій вовків та значеннями об’єктивної функції на

графіку, що допомагає їм аналізувати ефективність параметрів та алгоритму в цілому.

Графік доповнюється іншими важливими даними, такими як координати вовків і статистика результатів (рис. 3.24). Це дозволяє користувачам отримувати повну картину роботи алгоритму та приймати обґрунтовані рішення щодо його налаштування та подальшого вдосконалення.

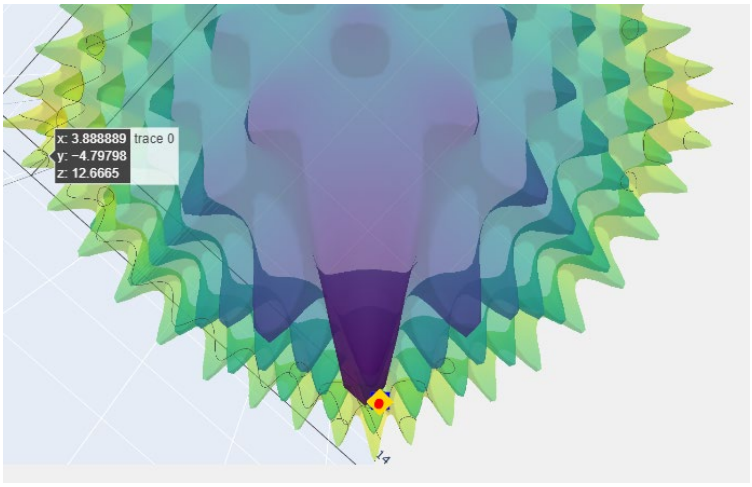

Рисунок 3.24 – Інтерфейс вкладки «Test Algorithm» з інтерактивним графіком

Завдяки інтерактивним графікам користувачі можуть детальніше аналізувати процес оптимізації та отримувати цінну інформацію про роботу алгоритму. Це робить дослідження та налаштування алгоритму більш зручними та ефективними, дозволяючи швидко виявляти й виправляти недоліки, а також знаходити оптимальні параметри для різних задач.

3.4 Експериментальні дослідження

Для тестування гібридного методу кластеризації, реалізованого на основі алгоритму сірих вовків, було обрано синтетичні дані. Синтетичні дані забезпечують гнучкість у контролюванні всіх параметрів і характеристик, що дозволяє краще оцінити ефективність алгоритму в різних умовах.

Використання синтетичних даних дає змогу швидко генерувати великі обсяги даних з відомими характеристиками, що є важливим для налаштування та тестування алгоритмів оптимізації.

В експериментальних дослідженнях використовувалось кілька типів синтетичних функцій, зокрема функції Sphere (рис. 3.25), Rosenbrock, Rastrigin та Ackley. Ці функції добре відомі та широко використовуються для тестування різних методів завдяки своїм різноманітним властивостям, таким як наявність локальних мінімумів, гладкість і складність ландшафту. Всі ці функції були збережені в базі даних MongoDB, що дозволяє легко додавати нові функції та швидко їх використовувати в подальших експериментах.

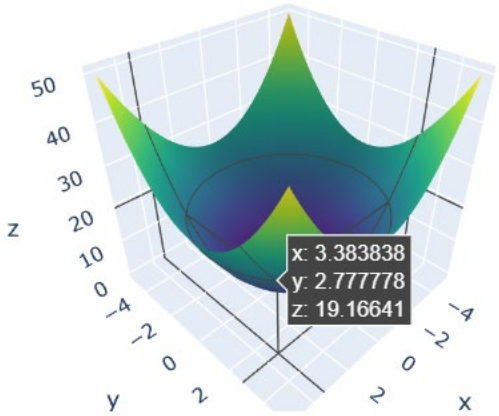

Рисунок 3.25 – Функція Sphere

Поперденьо всі дані були передоброблені для якісної роботи методу: всі дані були нормалізовані для забезпечення однакових масштабів різних параметрів. Нормалізація є важливим кроком, оскільки різні масштаби параметрів можуть негативно впливати на процес оптимізації та призводити до некоректних результатів. Крім того, синтетичні дані генерувалися з метою уникнення шуму, оскільки основною метою було оцінити чисту ефективність алгоритму без впливу зовнішніх факторів. Це дозволяє досягати більш точних результатів і краще розуміти, як алгоритм працює з різними типами функцій.

Для генерації даних використовувалися відомі математичні вирази для кожної функції, що забезпечує отримання точних значень для кожної точки в просторі пошуку. Після генерації та нормалізації дані зберігалися в базі даних MongoDB, що забезпечує легкий доступ до них у будь-який час і можливість використання для подальших експериментів. Це спрощує роботу з даними та дозволяє зберігати історію всіх експериментів для подальшого аналізу та порівняння результатів.

Таким чином, підготовка та обробка синтетичних даних створюють надійну основу для тестування та налаштування алгоритму, що дозволяє зосередитися на його ефективності та адаптивності в різних умовах оптимізації.

Для оцінки ефективності різних параметрів алгоритму, а також різних об’єктивних функцій і вибірок даних, було проведено комплексну серію експериментів, під час яких змінювали налаштування параметрів алгоритму та типи об’єктивних функцій. Основою для оцінки результатів стали кілька ключових метрик, серед яких – середнє значення найкращого знайденого рішення (альфа-вовк), стабільність отриманих результатів і час виконання алгоритму.

Параметри алгоритму, такі як y, d і σ^2, виявилися критично важливими для досягнення оптимальної ефективності CGWO. Наприклад, підвищення значення σ^2 веде до збільшення випадковості у поведінці алгоритму, що дозволяє йому ефективніше досліджувати простір пошуку. Однак така зміна може призвести до зниження стабільності результатів. Водночас оптимально підібрані значення цих параметрів можуть забезпечити необхідний баланс між дослідженням нових ділянок простору та експлуатацією вже відомих, що, в свою чергу, дозволяє отримувати більш стабільні та точні результати.

Додатково, порівняння результатів для різних об’єктивних функцій продемонструвало, що CGWO адаптивно реагує на різні типи функцій, зберігаючи високу ефективність у більшості випадків. Наприклад, на простій функції Sphere алгоритм продемонстрував швидке знаходження глобального

мінімуму, тоді як на більш складних функціях, таких як Rosenbrock, він показав не лише високу точність, але й стабільність, що є важливими показниками для алгоритмів оптимізації.

Таким чином, результати проведених експериментальних досліджень підтверджують (рис. 3.26), що CGWO є гнучким і ефективним методом для вирішення задач оптимізації, здатним адаптуватися до змінних умов і різних характеристик об’єктивних функцій.

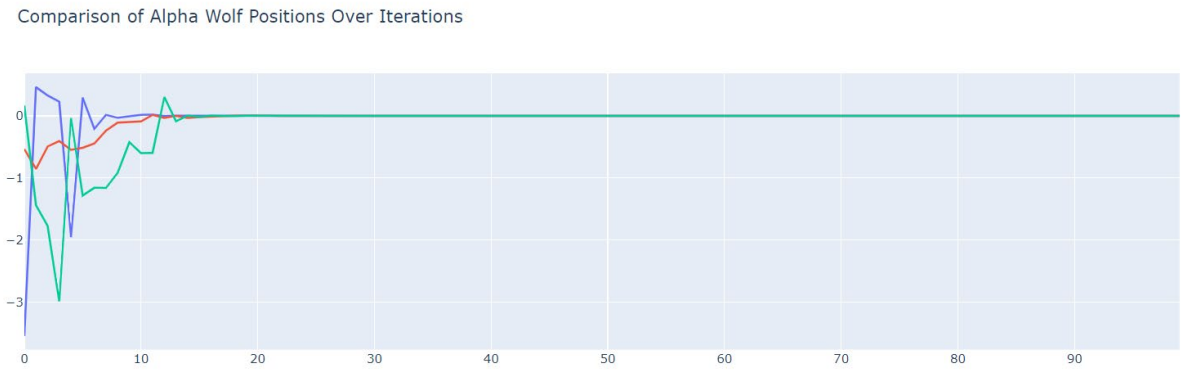

Рисунок 3.26 – Графік порівняння результатів для різних функцій

Порівняння результатів для різних вибірок даних продемонструвало, що CGWO забезпечує стабільні результати незалежно від типу даних, що робить його універсальним інструментом для різноманітних задач оптимізації. Висока точність і стабільність результатів у більшості експериментів підтверджують цю тезу.

Отже, результати проведених експериментів свідчать про вражаючу ефективність запропонованого гібридного методу кластеризації на основі CGWO в порівнянні з традиційними методами, такими як *K*-means і DBSCAN. CGWO показує високу адаптивність, стабільність і точність в різних умовах, що робить його перспективним інструментом для вирішення складних задач оптимізації. Це підкреслює його потенціал у застосуваннях, що вимагають надійних і ефективних рішень в умовах різноманіття даних.

Для візуалізації результатів роботи алгоритму застосовано інтерактивні графіки, створені за допомогою бібліотеки Plotly. Ці графіки забезпечують можливість детально аналізувати розподіл кластерів, відстежувати зміни

значень цільової функції на кожній ітерації, а також спостерігати динаміку процесу оптимізації.

Один із ключових аспектів візуалізації – це графіки розподілу кластерів. На таких графіках відображаються позиції вовків у просторі пошуку на кожній ітерації. Це дозволяє візуально оцінити, як алгоритм досліджує простір і як змінюються позиції найкращих (альфа, бета та дельта) вовків з кожною новою ітерацією. Наприклад, нижче представлено графік, що ілюструє розподіл вовків на функції Sphere після завершення оптимізації (рис. 3.27).

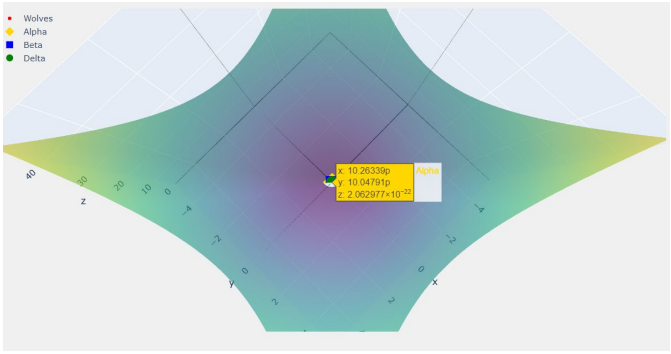

Рисунок 3.27 – Розподіл кластерів на функції Sphere

Крім того, для кожної об’єктивної функції створено графіки, які демонструють зміни значень цільової функції протягом процесу оптимізації. Ці графіки відображають, як змінюються значення цільової функції (значення альфа-вовка) на кожній ітерації. Це надає можливість оцінити ефективність алгоритму та швидкість його збіжності до глобального оптимуму. Наприклад, на рисунку 3.28 представлено графік, зміни значень цільової функції для функції Rosenbrock.

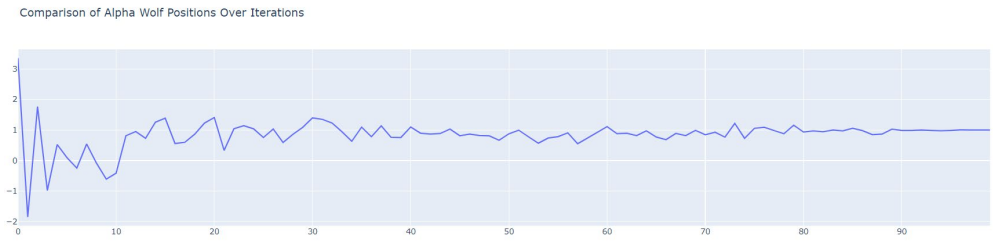

Рисунок 3.28 – Графік зміни значень цільової функції для функції Rosenbrock

Додатковими важливими елементами візуалізації є анімаційні графіки, які ілюструють динаміку процесу оптимізації, показуючи, як змінюються позиції вовків в реальному часі. Такі графіки дозволяють наочно оцінити, як алгоритм досліджує простір пошуку та адаптується до різних умов оптимізації, що є критично важливим для розуміння його роботи та ефективності.

Результати дослідження підтверджують високу ефективність гібридного методу кластеризації на основі CGWO в розв'язанні задач оптимізації з використанням різноманітних об'єктивних функцій. Завдяки додатковій випадковості, яку вводять параметри *y, d* і σ^2, алгоритм має змогу уникати локальних мінімумів, що дозволяє йому досягати глобальних оптимумів із значною точністю.

Аналіз отриманих графіків і діаграм вказує на те, що метод CGWO демонструє стабільні результати під час оптимізації складних об'єктивних функцій, таких як Rastrigin (рис. 3.29) і Rosenbrock. У цих випадках традиційні методи кластеризації, зокрема *K*-means і DBSCAN, часто стикаються з труднощами і застрягають у локальних мінімумів. Ці висновки підкріплюються результатами порівняльних експериментів, які продемонстрували значну перевагу CGWO над іншими алгоритмами в умовах, що характеризуються складними функціями. Таким чином, CGWO не лише виявляється більш ефективним, але й здатним надавати стабільні результати в різноманітних умовах оптимізації.

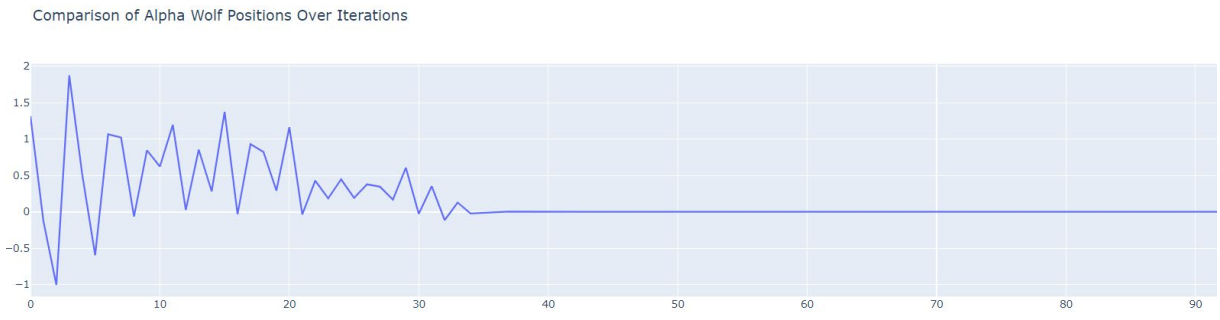

Рисунок 3.29 – Процес оптимізації для функції Rastrigin

Висновки щодо практичного застосування гібридного методу CGWO свідчать про те, що цей підхід є перспективним інструментом для вирішення складних задач оптимізації в різних сферах. Його висока стабільність, адаптивність та точність роблять CGWO ефективним вибором для задач, що вимагають знаходження глобального оптимуму в широких та складних просторах пошуку.

Підсумовуючи, можна відзначити, що розроблений гібридний метод кластеризації на основі CGWO демонструє значну перевагу у порівнянні з традиційними методами кластеризації. Отримані результати підтверджують доцільність і практичність його застосування для вирішення задач оптимізації, що включають різні об’єктивні функції. Це робить CGWO потужним інструментом, який може бути використаний у наукових дослідженнях та промислових задачах, де важливими є висока точність і стабільність результатів.

Для покращення візуалізації процесу оптимізації також створено графіки, які відображають зміну параметрів алгоритму на кожній ітерації. Ці графіки дозволяють спостерігати за варіацією значень параметрів *y, d* і σ^2 в ході виконання алгоритму, а також демонструють, як ці зміни впливають на результати оптимізації. Це надає можливість глибшого розуміння роботи алгоритму та підвищує ефективність його налаштування для досягнення ще кращих результатів.

Результати експериментів, проведених з використанням різних об’єктивних функцій та параметрів, підтверджують високу ефективність і стабільність розробленого методу CGWO. Цей підхід демонструє значні переваги в порівнянні з традиційними методами, особливо в ситуаціях, коли мова йде про складні об’єктивні функції з великою кількістю локальних мінімумів. Висока продуктивність CGWO у таких умовах робить його надійним інструментом для вирішення складних задач оптимізації.

Під час проведення тестування гібридного методу кластеризації, заснованого на модифікованому алгоритмі сірих вовків (CGWO), виявлено

кілька важливих переваг, які підкреслюють його ефективність і корисність у вирішенні складних завдань оптимізації.

По-перше, одним із ключових аспектів методу є його висока точність у знаходженні глобального мінімуму. Це досягається завдяки здатності алгоритму уникати застрягання в локальних мінімумів, що часто є проблемою при оптимізації складних функцій з численними локальними мінімумами. Завдяки введенню параметрів *y, d* та σ, які вносять додаткову випадковість у процес, CGWO має можливість ефективно досліджувати широкий простір пошуку. Це забезпечує алгоритму гнучкість, необхідну для успішного вирішення завдань різної складності.

По-друге, CGWO вражає своєю швидкістю збіжності. Алгоритм ефективно використовує соціальну ієрархію вовків і механізми випадкових збурень, що дозволяє йому швидко адаптуватися до характеристик об’єктивної функції. Це призводить до швидшого знаходження оптимальних рішень у порівнянні з багатьма традиційними методами, такими як *K*-means або DBSCAN. Висока швидкість збіжності не лише економить обчислювальні ресурси, але й дає змогу отримувати результати в коротший термін, що є важливим фактором у динамічних та конкурентних середовищах.

Стабільність результатів є ще однією важливою характеристикою CGWO. Алгоритм демонструє високу повторюваність результатів, що підкреслює його надійність та передбачуваність. Це особливо важливо у прикладних задачах, де результати повинні бути стабільними та точними, адже невизначеність може призвести до значних помилок в ухваленні рішень.

Однак, незважаючи на всі переваги, гібридний метод CGWO має певні обмеження. Одним із них є необхідність ретельного налаштування параметрів *y, d* та σ^2 для досягнення оптимальних результатів. Неправильний вибір цих параметрів може суттєво знизити ефективність алгоритму. Це підкреслює важливість проведення додаткових експериментів для визначення найкращих значень цих параметрів у різних умовах.

Ще одне обмеження пов’язане з можливими труднощами у масштабуванні алгоритму для дуже великих завдань. Хоча CGWO показує відмінну продуктивність на середніх обсягах даних, його ефективність може знижуватися при обробці великих датасетів. Щоб подолати це обмеження, доцільно розглянути можливість впровадження паралельних обчислень або розподілених алгоритмів. Це дозволило б підвищити продуктивність та забезпечити масштабованість методу в умовах обробки великих обсягів інформації.

В аспекті подальшого вдосконалення алгоритму існує кілька цікавих напрямків. По-перше, варто дослідити можливості автоматичного налаштування параметрів *y, d* та σ^2 під час процесу оптимізації. Це дозволило б алгоритму самостійно адаптувати свої параметри до специфічних умов задачі, що зменшить потребу в ручному налаштуванні та підвищить загальну ефективність роботи.

По-друге, інтеграція CGWO з іншими методами оптимізації може стати ще одним напрямком для вдосконалення. Наприклад, поєднання CGWO з методами глибокого навчання або генетичними алгоритмами може призвести до створення нових гібридних алгоритмів, що продемонструють ще більшу ефективність при розв’язанні складних задач оптимізації. Ця комбінація методів відкриває нові можливості для досліджень та практичного застосування у різних галузях, таких як фінанси, медицина, інженерія та багато інших.

Отже, незважаючи на деякі обмеження, гібридний метод кластеризації на основі CGWO показує високу ефективність, стабільність і точність. Подальші дослідження та вдосконалення цього підходу можуть ще більше підвищити його продуктивність і розширити його застосування в різних галузях.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи були досягнуті заплановані цілі та завдання. Досліджено й апробовано гібридний метод правдоподібної кластеризації на основі еволюційного алгоритму сірих вовків, що призвело до покращення якості та кількості показників кластеризації даних. Експериментальні результати підтвердили ефективність цього методу в різних умовах, включаючи складні та багатопікові функції належності.

Розроблений підхід має потенціал для вдосконалення існуючих методів кластеризації даних завдяки поєднанню переваг еволюційних алгоритмів і правдоподібної кластеризації, що дозволяє досягати більш точних і стабільних результатів у порівнянні з традиційними методами.

Науковою навизною даного дослідження є те, що розроблений метод виключає можливість «застрягання» в локальних екстремумах за допомогою подвійної перевірки знаходження вовка-домінанта в екстремумі та порівнянні із заданою похибкою розрахунків, дозволяє скоротити кількість запусків процедури. Метод є достатньо простим у чисельній реалізації, дозволяє знаходити глобальні екстремуми складних функцій, що підтверджується результатами чисельного експерименту.

Таким чином, матеріали кваліфікаційної роботи можуть бути використані не лише для практичного застосування в відповідних сферах, а й для навчальних цілей, забезпечуючи студентів і дослідників інструментами для глибшого аналізу та оптимізації алгоритмів кластеризації. Розроблений метод відкриває нові перспективи для покращення результатів в аналізі даних і може стати основою для подальших наукових досліджень.

Результати роботи апробовано у вигляді тез доповіді під час V міжнародної нукової конференції «Інновації та науковий потенціал світу», м. Умань, Україна [39].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Gorban, A. N., Kégl, B., Wunsch, D. C., & Zinovyev, A. Y. (Eds.). (2008). *Principal manifolds for data visualization and dimension reduction* (Vol. 58, pp. 96-130). Berlin: Springer.

2. Marwala, Tshilidzi, ed. *Computational Intelligence for Missing Data Imputation, Estimation, and Management: Knowledge Optimization Techniques: Knowledge Optimization Techniques*. IGI Global, 2009.

3. Wunsch, Donald C., A. Rossiev, and Alexander N. Gorban. "Neural Network Modeling of Data with Gaps." (2000).

4. Tkacz, Magdalena. "Artificial neural networks in incomplete data sets processing. "*Intelligent Information Processing and Web Mining: Proceedings of the International IIS: IIPWM'05 Conference held in Gdansk, Poland, June 13–16, 2005*. Springer Berlin Heidelberg, 2005.

5. Kohonen, T. (1991). Self-organizing maps: Ophmization approaches. In *Artificial neural networks* (pp. 981-990). North-Holland.

6. Dracopoulos, D. C. Evolutionary Learning Algorithms for Neural Adaptive Control [electronic resource].

7. Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford university press.

8. Shafronenko, A., Bodyanskiy, Y. V., & Pliss, I. (2023). Credibilistic Fuzzy Clustering Method Based on Evolutionary Approach of Crazy Wolfs in Online Mode. In *CMIS* (pp. 141-150).

9. Shafronenko, A. Y., Bodyanskiy, Y. V., & Holovin, O. O. (2023). CLUSTERIZATION OF DATA ARRAYS BASED ON THE MODIFIED GRAY WOLF ALGORITHM. *SCIENCE OF THE TOTAL ENVIRONMENT*, *873*, 73-79.

10. Shafronenko, A., Bodyanskiy, Y., Pliss, I., & Patlan, K. (2019, June). Fuzzy Clusterization of Distorted by Missing Observations Data Sets Using Evolutionary Optimization. In *2019 9th International Conference on Advanced Computer Information Technologies (ACIT)* (pp. 217-220). IEEE.

11. Bodyanskiy, Y., Shafronenko, A., & Pliss, I. (2022). Clusterization of vector and matrix data arrays using the combined evolutionary method of fish schools. *System research and information technologies*, (4), 79-87.

12. Shafronenko, A. Y., Bodyanskiy, Y. V., & Pliss, I. P. (2019, September). The Fast Modification of Evolutionary Bioinspired Cat Swarm Optimization Method. In *2019 IEEE 8th International Conference on Advanced Optoelectronics and Lasers (CAOL)* (pp. 548-552). IEEE.

13. Shafronenko, A., & Bodyanskiy, Y. V. (2020). Adaptive fuzzy clustering approach based on evolutionary cat swarm optimization. In *CMIS* (pp. 832-842).

14. Chu, S. C., Tsai, P. W., & Pan, J. S. (2006). Cat swarm optimization. In *PRICAI 2006: Trends in Artificial Intelligence: 9th Pacific Rim International Conference on Artificial Intelligence Guilin, China, August 7-11, 2006 Proceedings 9* (pp. 854-858). Springer Berlin Heidelberg.

15. Cavalcanti-Júnior, G. M., Bastos-Filho, C. J., Lima-Neto, F. B., & Castro, R. M. (2011). A hybrid algorithm based on fish school search and particle swarm optimization for dynamic problems. In *Advances in Swarm Intelligence: Second International Conference, ICSI 2011, Chongqing, China, June 12-15, 2011, Proceedings, Part II 2* (pp. 543-552). Springer Berlin Heidelberg.

16. Shafronenko, A., & Bodyanskiy, Y. (2019). Online algorithm for possibilitic fuzzy clustering based on evolutionary cat swarm optimization. *Science and Education a New Dimension. Natural and Technical Sciences*, *193*, 86-88.

17. Bodyanskiy, Y. V., Shafronenko, A. Y., & Pliss, I. P. (2021). Credibilistic fuzzy clustering based on evolutionary method of crazy cats. *Системні дослідження та інформаційні технології*, *2021*(3), 110-119.

18. Shafronenko, A., Bodyanskiy, Y., & Rudenko, D. (2020). *Neuro-fuzzy clustering of Distorted Data Using Cat Swarm Optimization*. LAP LAMBERT Academic Publishing.

19. Bodyanskiy, Y. V., Pliss, I. P., & Shafronenko, A. Y. (2022). Кластеризація масивів даних на основі комбінованої оптимізації функцій

щільності розподілу та еволюційного методу котячих зграй. *Radio Electronics, Computer Science, Control*, (4), 61-61.

20. Bezdek, J. C., Keller, J., Krisnapuram, R., & Pal, N. (1999). *Fuzzy models and algorithms for pattern recognition and image processing* (Vol. 4). Springer Science & Business Media.

21. Ribeiro, B. M. (2010). Computational intelligence: Neural networks and kernel methods. *Computational Intelligence: Neural Networks and Kernel Methods*.

22. Fernandez, A., Herrera, F., Cordon, O., del Jesus, M. J., & Marcelloni, F. (2019). Evolutionary fuzzy systems for explainable artificial intelligence: Why, when, what for, and where to?. *IEEE Computational intelligence magazine*, *14*(1), 69-81.

23. Kroll, A. (2013). *Computational Intelligence: Eine Einführung in Probleme, Methoden und technische Anwendungen*. Oldenbourg Wissenschaftsverlag Verlag.

24. Mirjalili, S. (2015). The ant lion optimizer. *Advances in engineering software*, *83*, 80-98.

25. Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey wolf optimizer. *Advances in engineering software*, *69*, 46-61.

26. Yang, X. S., Chien, S. F., & Ting, T. O. (2015). *Bio-inspired computation in telecommunications*. Morgan Kaufmann.

27. Syberfeldt, A., & Lidberg, S. (2012, December). Real-world simulation-based manufacturing optimization using cuckoo search. In *Proceedings of the 2012 winter simulation conference (WSC)* (pp. 1-12). IEEE.

28. dos Santos Coelho, L., & Mariani, V. C. (2013). Improved firefly algorithm approach applied to chiller loading for energy conservation. *Energy and buildings*, *59*, 273-278.

29. Gao, Z. M., & Zhao, J. (2019). An improved grey wolf optimization algorithm with variable weights. *Computational Intelligence and Neuroscience*, *2019*(1), 2981282.

30. James, J. Q., & Li, V. O. (2015). A social spider algorithm for global optimization. *Applied soft computing*, *30*, 614-627.

31. Azizi, R. (2014). Empirical study of artificial fish swarm algorithm. *arXiv preprint arXiv:1405.4138*.

32. Zhang, X., Zhang, X., Ho, S. L., & Fu, W. N. (2014). A modification of artificial bee colony algorithm applied to loudspeaker design problem. *IEEE Transactions on Magnetics*, *50*(2), 737-740.

33. Marichelvam, M. K., Prabaharan, T., & Yang, X. S. (2013). A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems. *IEEE transactions on evolutionary computation*, *18*(2), 301-305.

34. Hathaway, R. J., & Bezdek, J. C. (1995). Optimization of clustering criteria by reformulation. *IEEE transactions on Fuzzy Systems*, *3*(2), 241-245.

35. Pal, N. R., Bezdek, J. C., & Hathaway, R. J. (1996). Sequential competitive learning and the fuzzy c-means clustering algorithms. *Neural networks*, *9*(5), 787-796.

36. Bodyanskiy, Y. V., Pliss, I. P., & Shafronenko, A. Y. (2022). Кластеризація масивів даних на основі комбінованої оптимізації функцій щільності розподілу та еволюційного методу котячих зграй. *Radio Electronics, Computer Science, Control*, (4), 61-61.

37. Bodyanskiy, Y. V., Shafronenko, A. Y., & Pliss, I. P. (2021). Credibilistic fuzzy clustering based on evolutionary method of crazy cats. *Системні дослідження та інформаційні технології*, *2021*(3), 110-119.

38. Shafronenko, A. Y., Bodyanskiy, Y. V., & Holovin, O. O. (2023). CLUSTERIZATION OF DATA ARRAYS BASED ON THE MODIFIED GRAY WOLF ALGORITHM. SCIENCE OF THE TOTAL ENVIRONMENT, 873, 73-79.

39. Шафроненко, А., Фалько, М. (2024). НЕЧІТКИЙ МЕТОД КЛАСТЕРИЗАЦІЇ ДАНИХ З ВИКОРИСТАННЯМ ЕВОЛЮЦІЙНОЇ ПРОЦЕДУРИ. *Інновації та науковий потенціал світу: збірник наукових праць з матеріалами V Міжнародної наукової конференції, м. Умань, 25 жовтня, 2024 р.*, 94-95.