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 МЕТОДУ ЕВОЛЮЦІЙНОЇ ОПТИМІЗАЦІЇ НА ОСНОВІ АЛГОРИТМУ КЛАСТЕРИЗАЦІЇ ЗМІШАНИХ ДАНИХ
(тема)

Виконав:
студент 4 курсу, групи ІНФм-20-1
Свистунов І.О.
(прізвище, ініціали)

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

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

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)

2021 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
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Свистунов_Ілля_Олексійович
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методу еволюційної оптимізації на основі алгоритму кластеризації змішаних даних
затверджена наказом по університету від «22» жовтня 2021 року № 1574Ст.
2. Термін подання студентом роботи до екзаменаційної комісії ___ _____ 2021 р.
3. Вихідні дані до роботи математичні моделі оптимізації алгоритмів, перелік використовуваних програмних засобів: теоретичні відомості про методи кластеризації змішаних даних та алгоритмів оптимізації.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд методів оптимізації.
2. Математичні моделі методів оптимізації.
3. Аналіз актуальності методів оптимізації.
4. Переваги еволюційних алгоритмів.
5. Огляд методологій розробки програмного забезпечення.
6. Способи моделювання розробки програмної системи.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів,

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

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	22.10.2021	
2	Аналіз завдання, підбір літератури	22.10.21-03.11.21	
3	Аналіз літератури з досліджуваної проблеми	03.11.21-09.11.21	
4	Аналіз технічних засобів	09.11.21-11.11.21	
5	Розробка методу	01.11.21-10.11.21	
6	Програмна реалізація	02.11.21-11.11.21	
7	Оформлення пояснювальної записки	08.11.21-15.11.21	
8	Перевірка на плагіат	20.11.2021	
9	Рецензування	28.11.2021	
10	Підготовка презентації та доповіді	29.11.2021	
11	Занесення роботи в електронний архів	06.12.2021	
12	Попередній захист кваліфікаційної роботи	06.12.2021	

Дата видачі завдання _22__жовтня_____2021 р.

Студент _____
(підпис)

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

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 84 с., 22 рис., 41 джерело.

ЕВОЛЮЦІЙНИЙ АЛГОРИТМ, АЛГОРИТМ ОПТИМІЗАЦІІ ЛЕВА, КЛАСТЕРИЗАЦІЯ ДАНИХ, ЗМІШАНІ ДАНІ.

Об’єктом дослідження цієї роботи є еволюційні алгоритми. Багато з цих алгоритмів натхненні різними явищами природи. У цій роботі представлено новий алгоритм на основі популяцій – алгоритм оптимізації Лева (LOA) для змішаних даних.

Метою дослідження є аналіз працездатності еволюційних алгоритмів для кластеризації змішаних даних.

Основною мотивацією для розробки даного алгоритму оптимізації стали особливий спосіб життя левів та особливості їхньої співпраці. Деякі контрольні задачі вибрані з літератури, і рішення запропонованого алгоритму порівняно з рішеннями деяких добре відомих і новітніх мета-евристик для цих задач.

Отримані результати підтверджують високу продуктивність запропонованого алгоритму в порівнянні з іншими алгоритмами, використаними в даній роботі.

Розроблений алгоритм впроваджено у застосунок для оптимізації за допомогою алгоритму оптимізації лева. Застосунок розроблений за допомогою мови програмування Python та пакету для наукових обчислень NumPy.

EVOLUTIONARY ALGORITHM, LION OPTIMIZATION ALGORITHM, DATA CLUSTERING, MIXED DATA.

The object of study of this work is evolutionary algorithms. Many of these algorithms are inspired by various natural phenomena. This paper presents a new algorithm based on populations - the Lion optimization algorithm (LOA) for mixed data.

The aim of the study is to analyze the performance of evolutionary algorithms for clustering mixed data.

The main motivation for the development of this optimization algorithm was the special way of life of lions and the peculiarities of their cooperation. Some control problems are selected from the literature, and the solutions of the proposed algorithm are compared with the solutions of some well-known and newest meta-heuristics for these problems.

The obtained results confirm the high performance of the proposed algorithm in comparison with other algorithms used in this work.

The developed algorithm is implemented in the application for optimization using the lion optimization algorithm. The application is developed using the Python programming language and the NumPy scientific computing package.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

EA – еволюційні алгоритми

NP – Nondeterministic polynomial

TS – Tabu search

LOA – Lion optimization algorithm

SA – Simulated annealing

PSO – Particle swarm optimization

ACO – Ant colony optimization

EM – еволюційний метод

IDE – Integrated development environment

UCI – UC Irvine

ВСТУП

Бізнес сьогоднішніх складних організацій, таких як банки, авіакомпанії та лікарні вимагає збереження та обробку великої кількості даних. Для оптимізованої роботи з великою кількістю об’єктів та подальшого аналізу даних використовують кластеризацію даних, яка розбиває множину об’єктів на кластери схожих об’єктів. Один із видів алгоритмів кластеризації даних є еволюційні алгоритми, які натхнені різними явищами природи.

Еволюційні обчислення становлять один із розділів штучного інтелекту. При побудові систем основна увага приділяється побудові початкової моделі, і правил, якими вона може змінюватися (еволюціонувати). Причому модель може бути складена за різними методами, наприклад, це може бути і нейронна мережа та набір логічних правил. До основних еволюційних методів належать методи відпалу, генетичні, поведінки «натовпу» (PSO), колонії мурах (ACO), генетичного програмування.

На відміну від точних методів математичного програмування ЕМ дозволяють знаходити рішення, близькі до оптимальних, за прийнятний час, а на відміну від інших евристичних методів оптимізації характеризуються істотно меншою залежністю від особливостей додатка і в більшості випадків забезпечують кращий ступінь наближення до раціонального рішення. Універсальність ЕМ визначається також застосовністю до завдань з неметризованим простором керованих змінних (тобто серед керованих змінних можуть бути і лінгвістичні величини, тобто не мають кількісного виразу).

Широке застосування нечітких систем на вирішення проблем автоматичного управління, прогнозування, прийняття рішень змушує фахівців шукати ефективні методи побудови систем, для ідентифікації яких поруч із алгоритмами оптимізації, заснованими на похідних, застосовуються генетичні алгоритми, еволюційні стратегії та нейронні мережі. Еволюційні стратегії разом із еволюційним програмуванням і генетичним алгоритмом

представляють три основних напрями розвитку еволюційного моделювання. Незважаючи на те, що кожен із методів виник незалежно від інших, вони характеризуються низкою кількістю загальних властивостей. Для кожного з них формується вихідна популяція, яка піддається селекції та впливу різних генетичних операторів, що дозволяє шукати кращі рішення. Еволюційна стратегія – це алгоритми, створені як методи вирішення оптимізації завдань і засновані на принципах природної еволюції. Метою роботи є опис класичного алгоритму еволюційних методів оптимізації та реалізація алгоритму оптимізації лева для змішаних даних.

1 ОГЛЯД ОСНОВНИХ МЕТОДІВ ЕВОЛЮЦІЙНОЇ ОПТИМІЗАЦІЇ

1.1 Еволюційні алгоритми

У штучному інтелекті та машинному навчанні еволюційні алгоритми – це розділ еволюційних обчислень, у яких використовуються моделі процесів природного відбору (розмноження, мутація, рекомбінація та відбір) та принципи природної еволюції для вирішення задач оптимізації.

Рішення оптимізаційної задачі розглядаються як особи популяції. Якість кожного рішення оцінюється за допомогою спеціальної функції придатності (fitness function – фітнес-функція), після чого відбувається еволюція популяції. Таким чином, еволюційний алгоритм містить такі кроки:

– формується початкова населення шляхом випадкового відбору (перше покоління);

– оцінюється придатність кожного члена популяції за допомогою фітнес-функції;

– повторюються такі дії (еволюція): відбір – вибір найбільш пристосованих особин для розмноження (батьки); розмноження – формування нових особин шляхом схрещування та мутації, а потім оцінка їхньої придатності; рекомбінація – найменш пристосовані особини попереднього покоління замінюються найбільш пристосованими особами нового покоління.

Еволюційні алгоритми мають такі переваги:

– застосовуються до широкого класу задач оптимізації;

– легко поєднуються з іншими методами;

– дозволяють отримувати результати, що добре інтерпретуються;

– мають інтерактивність – можна включити в популяцію запропоновані користувачем рішення;

– розглядають велику кількість альтернативних рішень.

До недоліків еволюційних алгоритмів можна віднести:

– відсутність гарантії знаходження оптимального рішення за кінцевий час – алгоритм реалізує локальну оптимізацію;

– слабка теоретична основа – алгоритм є евристичним, тобто. точність і строгість постановки приносяться в жертву реалізованості;

– складність у використанні – може знадобитися налаштування параметрів моделі, що оптимізується за допомогою еволюційного алгоритму;

– високі обчислювальні витрати – алгоритм не є масштабованим.

Еволюційні обчислення – це галузь наукових досліджень про розв'язання складних завдань оптимізації за допомогою алгоритмів, натхненних теорією еволюції. Метою оптимізації може бути, наприклад, отримання оптимального розкладу роботи конвеєрів, що мінімізує час простою. Еволюційні обчислення включають різні напрями, що визначаються видом оптимізації: наприклад, розрізняють дискретну і безперервну оптимізацію, також існує багатокритеріальна оптимізація. Варто окремо виділити теорію еволюційних обчислень, що вивчає те, наскільки швидко ті чи інші еволюційні алгоритми можуть знаходити рішення чи складні ті чи інші завдання.

Як правило, спочатку генерується набір випадкових рішень – «особин». Вони можуть бути дуже далекі від оптимальних. Потім запускається процес еволюції: у рішення вносяться невеликі випадкові зміни (мутації), деякі рішення схрещуються між собою та породжують «нащадків». З рішень, що вийшли, в нове покоління відбираються найбільш пристосовані, і процес повторюється. Пристосованість рішень визначає те, що потрібно оптимізувати. Залежно від завдання це може бути час, вартість, потужність сигналу і так далі. Як правило, у кожному наступному поколінні вдається отримати все більш пристосовані, тобто все ближчі до оптимальних, рішення.

Люди визначають те, як обчислюватиметься пристосованість рішень і як ці рішення мають бути представлені в комп'ютері. Іноді розробляються спеціальні оператори мутації і схрещування, що враховують особливості завдання, що розв'язується. Сам процес пошуку оптимального рішення зазвичай виконується без участі людини за допомогою еволюційного алгоритму, реалізованого у вигляді спеціальної комп'ютерної програми. Однак трапляються випадки, коли автоматизувати обчислення дуже складно. Наприклад, як чисельно визначити естетичну цінність згенерованого зображення чи мелодії? У таких випадках може використовуватися так звана інтерактивна еволюція: еволюційний алгоритм під час своєї роботи «просить» людину оцінювати проміжні рішення. В інших випадках для визначення пристосованості може бути потрібним, наприклад, проведення хімічного або фізичного експерименту – тоді еволюційний алгоритм «чекатиме», коли вчені отримають необхідні дані.

Автори еволюційних алгоритмів надихалися теорією еволюції. Сучасні еволюційні алгоритми, на перший погляд, можуть бути досить далекі від початкової ідеї та використовувати складний математичний апарат. Але в їх основі, як правило, лежить відбір, що спирається на пристосованість, як і в основі біологічної еволюції – природний відбір. Використання ідей з біології в розробці нових еволюційних алгоритмів триває й донині: пробують привласнювати рішенням «підлогу» або, наприклад, піддавати їхньому «старінню». Також є дослідження, у яких результати з теорії еволюційних обчислень переносяться назад у біологію. Наприклад, можна спробувати оцінити ефективність методів генної інженерії, спираючись на інформацію про ефективність аналогічних еволюційних алгоритмів.

Більшість цікавих завдань – NP-важкі. З погляду сучасного стану науки це означає, що оптимальне рішення у випадку неможливо знайти за розумний час ніякими методами. Еволюційні алгоритми дозволяють наблизитись до оптимального рішення. Причому, як правило, чим більше часу дати алгоритму, тим ближче вийде підібратися. На практиці зазвичай

формулюються конкретні вимоги: наприклад, пристосованість отриманого рішення повинна бути не нижчою від такої величини. Можна порахувати пристосованість, і якщо результат не влаштовує, запустити алгоритм більш тривалий час, можливо, попередньо внісши в нього якісь корисні зміни.

Сфера застосування еволюційних обчислень дуже широка – вони корисні скрізь, де потрібно щось оптимізувати, і використовуються як в індустрії, наприклад, у промисловому дизайні, логістиці та управлінні, так і в інших галузях науки: біоінформатиці, медицині, робототехніці, машинному навчанні (наприклад, для створення структури нейронних мереж). Активним напрямом у межах еволюційних обчислень є так звана пошукова інженерія програмного забезпечення, у якій вирішуються питання автоматичної генерації, зміни та тестування програмного коду. Також еволюційні обчислення можуть застосовуватися до створення об’єктів мистецтва та генерації рівнів чи персонажів у комп’ютерних іграх.

Існує багато еволюційних методів. Зараз оглянемо основних представників.

1.1.1 Simulated Annealing

У методі відпалу (Simulated Annealing) імітується процес мінімізації потенційної енергії тіла під час відпалу деталей. У поточній точці пошуку відбувається зміна деяких керованих параметрів. Нова точка приймається завжди при поліпшенні цільової функції і лише з певною ймовірністю при її погіршенні.

Алгоритм SA – один з найбільш переважних евристичних методів для вирішення задач оптимізації. Алгоритм SA, симулює процедуру відпалу при обробці металу. Процедура відпалу визначає оптимальне молекулярне розташування металевих частинок, де потенціальна енергія маси зводиться до мінімуму, і означає поступове охолодження металів після нагрівання.

Загалом алгоритм SA приймає ітераційний рух відповідно до змінного температурного параметра, який імітує трансакцію відпалу металів.

Простий алгоритм оптимізації ітеративно порівнює результати роботи цільових функцій із поточною та сусідньою точкою в області, так що, якщо сусідня точка генерує кращий результат, ніж поточна, вона зберігається як базове рішення для наступної ітерації. В іншому випадку алгоритм припиняє процедуру, не шукаючи ширшої області для отримання кращих результатів. Отже, алгоритм схильний потрапляти в пастку локальних мінімумів або максимумів. Натомість алгоритм SA пропонує ефективне вирішення цієї проблеми, включаючи дві ітераційні петлі, які є процедурою охолодження процесу відпалу та критерієм Metropolis. Основна ідея, що стоїть за критерієм Metropolis, полягає в тому, щоб виконати її випадковим чином для додаткового пошуку в околицях рішення кандидата, щоб уникнути потрапляння в пастку локальних крайніх точок. Точніше, розглянемо проблему мінімізації та визначимо цільову функцію $F(X_i)$ відповідно до набору аргументів $X_i = \{X_1, X_2, ..., X_n\}$, де $n \in \mathbb{R}$. Тому, якщо $F(X_{i+1}) < F(X_i)$ беремо X_{i+1} як новий кандидат крайньої точки для перевірки. В іншому випадку визначте:

$$w = \exp[-\{F(X_{i+1}) - F(X_i)\} / T_c],$$

де T_c – поточний параметр температури та генерує випадкове число *s*, яке $0 < s < 1$.

Тоді, якщо відношення $w > s$ вірне, ви також приймаєте X_{i+1} в якості нового кандидату, інакше відхиліть і поверніться до попереднього кроку та генеріть інше *s*. Отже, критерій Metropolis дозволяє певною мірою рухатись поточним кроком, навіть якщо траєкторія цільової функції збігається через потенційну локальну точку мінімуму.

З іншого боку, алгоритм Metropolis пропонує рішення для постійної температури. Так що для більших значень T_c алгоритм потребує ширшої області пошуку. Тому це підвищує ймовірність досягнення глобального мінімуму. Крім того, оскільки ітераційний рух алгоритму ініціалізується випадковим чином, він може пропустити глобальні мінімуми або прийняти неточний підхід через крайню точку. Навпаки, для менших значень T_c, це вимагає меншої площі пошуку, і цей випадок може спричинити потрапляння у локальні мінімуми. Для усунення цього недоліку SA пропонує ітераційне рішення, яке включає вкладені петлі для зміни температурного параметра та точки розв'язання. Рух алгоритму SA починається з більшого значення температури для виконання ітерації для внутрішнього циклу, і негайно точка веде найкращу цільову функцію в поточному внутрішньому циклі, що призначається як новий кандидат на рішення. Потім зовнішній цикл запускається шляхом збільшення температури та оновлення початкової точки. Цей ітераційний процес триває до досягнення найнижчої межі температури або реалізації заздалегідь визначеної кількості ітерацій. Короткий зміст цієї процедури ілюструється на рисунку 1.1.

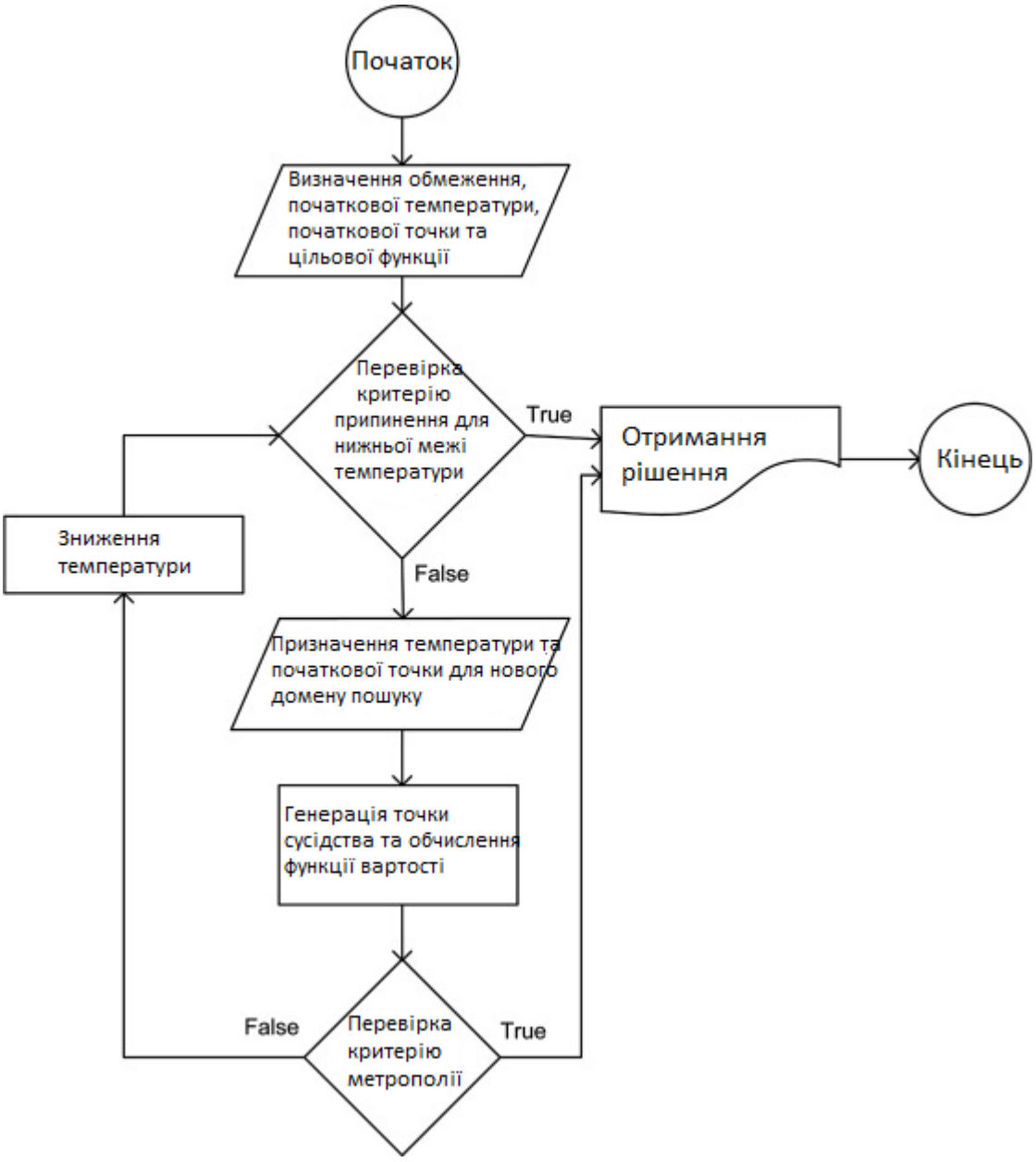

Рисунок 1.1 – Блок-схема методу SA

1.1.2 Ant Colony System

Алгоритм мурашиної колонії (Ant Colony System) – універсальний аналітичний алгоритм, який може бути використаний для вирішення багатьох комбінаторних задач.

Переваги:

– багатогранність – може бути використаний при вирішенні різних завдань і при різних постановках однієї і тієї ж задачі;

– робастність – може застосовуватися до довільних комбінаторним задачам оптимізації при мінімальних змінах структури;

– використовує популяцію рішень – відбувається використання позитивних зворотних зв'язків для пошуку рішення, може використовуватися на паралельних комп'ютерах.

Мурахи живуть і спостерігають світ в одній площині. Вони практично сліпі і глухі. Однак в пошуках їжі вони здатні знаходити найкоротший шлях до джерела їжі. Яким чином це у них виходить? Мурахи виділяють спеціальну речовину – феромон, яке має сильний, стійкий запах. Зазвичай мурахи орієнтуються і переміщуються по запаху залишеному іншими мурахами. Т.ч. виходить, що по шляху, по якому пройшло більшу кількість мурах, залишиться більше сліду (запаху), і нові мурахи підуть цим шляхом.

Приклад. Нехай спочатку мурахи знаходяться в точці А. Їжа знаходиться в точці Е. Відстані BD і DE рівні 1, а BC і CE рівні 0,5. У точці В мурахи розділяться порівну. У той час коли одна частина мурах (15 мурах) пройдуть відстань BD, інша частина встигне прийти BC і CE і піде назад. Коли мурахи, які обрали шлях Все повернутися назад, інші мурахи тільки дійдуть до точки Е, а коли вони повернуться, інші 15 мурах пройдуть шлях Все туди і назад ще раз. Т.ч., на шляху BDE залишать слід (феромон) 15 мурах, а на шляху BCE – 30 мурах (15 мурах пройдуть два рази). Тепер в точці В більша частина мурах вибере шлях BC.

Ідея автокаталитического алгоритму полягає не в імітації поведінки одного мурашки, а використанні ідеї поведінки колонії в цілому. Штучні мурашки живуть в штучному середовищі з дискретним часом. Вони наділені пам'яттю – не є повністю сліпими.

Розглянемо ідею алгоритму ant-cycle на прикладі завдання комівояжера (далі ЗК).

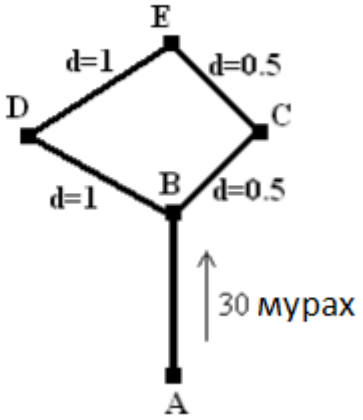

Рисунок 1.2 – Приклад поведінки колонії з 30 мурах

Нехай дано n міст. d_{ij} – відстані між містами (довжини ребер в графі), $d_{ij} \neq d_{ji}$. Необхідно знайти замкнутий обхід всіх міст, що має мінімальну довжину. Кожне місто має бути відвіданий тільки один раз.

Наприклад, (N, E), де N – кількість міст, E – безліч ребер, що з'єднують міста. Позначимо загальна кількість мурах m, $b_i(t)$ – кількість мурах в місті i в момент часу t, тобто:

$$\sum_{i=1}^{m} b_i(t) = m$$

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}$$

$$y_j < T_j$$

$$x : x_{i\min} < x_i < x_{i\max}.$$

Кожна мураха є простим агентом з наступними властивостями:

– вона вибирає місто, в який збирається перейти з ймовірністю, яка є функцією відстані до міста і кількості слідів на з'єднує ребрі;

– переходи в уже відвідані міста заборонені, поки обхід не завершений. Це контролюється за допомогою списку табу;

– коли отримано повний обхід, мураха залишає слід на кожному пройденому ребрі *ij*. $\tau_{ij}(t)$ – інтенсивність сліду на ребрі *ij* в момент часу *t*.

M ходів, виконані *m* мурахами за час *(t, t +1)* називається ітерацією. За n ітерацій всі мурахи здійснять один обхід – цикл алгоритму.

Після завершення циклу:

$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}$, де $0 < \rho < 1$ – коефіцієнт випаровування сліду.

$$\Delta\tau_{ij} = \sum_{i=1}^{m} \Delta\tau_{ij}^{k},$$

де $\Delta\tau_{ij}^{k}$ – кількість сліду, залишеного не одиниці довжини ребра *ij* *k*-ою мурахою.

Для того, щоб виключити необмежене накопичення сліду, кожному мурашки ставиться список ТАБУ. Після обходу, список обнуляється.

Позначимо:

tabu k – список табу у *k*-ого мурашки,

$$\eta_{ij} = \frac{1}{d_{ij}}$$

– видимість для ребра *ij*,

p_{ij}^{k} – ймовірність переходу з точки і в точку *j* для *k*-ого мурашки:

$$p_{ij}^{k} = \begin{cases} \dfrac{\left[\tau_{ij}^{k}(t)\right]^{\alpha} \cdot \left[\eta_{ij}\right]^{\beta}}{\sum_{k \notin tabu_k} \left[\tau_{ij}^{k}(t)\right]^{\alpha} \cdot \left[\eta_{ij}\right]^{\beta}}, j \notin tabu_k \\ 0, j \in tabu_k \end{cases},$$

де α, β – параметри алгоритму, управляють відносною важливістю видимості і сліду.

1.1.3 Particle Swarm Optimization

Ідею алгоритму Particle Swarm Optimization (PSO) вперше сформулювали Дж. Кеннеді і Д. К. Еберхарт в 1995 році. Ідея алгоритму була почерпнута з соціальної поведінки деяких тварин – зграї птахів, стада копитних або косяка риб. PSO аналогічний генетичному алгоритму – він починає зі створення популяції випадковим чином. Але, на відміну від ГА, у PSO немає таких еволюційних операторів, як схрещування і мутація. Рядки в PSO називаються частками, тоді як у ГА це хромосоми. Рядки-частинки являють собою вектор координат точки в просторі оптимізації (дійсних чисел), на відміну від ГА, хромосоми якого є бінарним кодом. Кожна частка пересувається по поверхні графіка функції з якоюсь швидкістю. Частинки змінюють свою швидкість і координати, грунтуючись на власному досвіді і досвіді інших частинок, що виражається формулами:

$$V_{m,n}^{new} = V_{m,n}^{old} + \Gamma_1 * r_1 * \left(p_{m,n}^{local\_best} - p_{m,n}^{old} \right) + \Gamma_2 * r_2 * \left(p_{m,n}^{global\_best} - p_{m,n}^{old} \right),$$

$$p_{m,n}^{new} = p_{m,n}^{old} + V_{m,n}^{new},$$

де $V_{m,n}$ – швидкість частинки, $p_{m,n}$ – координати частинки,

r_1, r_2 – незалежні випадкові числа, розподілені по рівномірному закону,

Γ_1, Γ_2 – коефіцієнти навчання,

$p_{m,n}^{local\_best}$ – кращі координати, коли-небудь знайдені даної часткою,

$p_{m,n}^{global\_best}$ – кращі координати, коли-небудь знайдені всією зграєю.

Алгоритм PSO спочатку оновлює вектор швидкості кожної частинки, а потім додає цей вектор до вектора координат відповідної частки. Оновлення вектора швидкості здійснюється з урахуванням найкращого глобального рішення, відповідного найменшому значенню функції, що мінімізується, коли-небудь знайденому всією зграєю, і з урахуванням найкращого локального рішення, відповідного найменшому значенню функції, коли-небудь знайденому даної часткою популяції. Якщо найкраще локальне рішення має значення функції, менше, ніж значення функції в глобальному найкращому рішенні, то воно стають кращим, коли-небудь знайденим рішенням для всієї зграї. Використання швидкості частинки в якомусь сенсі нагадує про алгоритми локальної оптимізації, що використовують інформацію про похідні, тому що швидкість – це похідна від координат. Константа Г1 називається когнітивним (тобто пізнавальним) параметром і дозволяє враховувати «власний досвід» (історію) частки. Константа Г2 називається соціальним параметром і дозволяє частці «враховувати досвід» всієї зграї. Перевага PSO полягає в простоті реалізації і в наявності малої кількості параметрів, що вимагають настройки. PSO здатний працювати зі складними цільовими функціями, що мають безліч локальних мінімумів.

Зазвичай оптимізація алгоритмом PSO виглядає наступним чином. Частинки, які спочатку рівномірно розподілені по поверхні функції, з плином часу (від покоління до покоління) починають групуватися («збиватися в зграї») близько локальних мінімумів, причому найбільша зграя збирається близько глобального мінімуму. При цьому майже завжди є частки, що знаходяться в стороні від таких зграй, а також частки, що вискакують за кордону допустимої області.

Дослідження ефективності стайного алгоритму проводилось на досить складних тестових функціях двох незалежних змінних – функції Розенброка і функції Гріванка.

Перша функція, хоча і одноекстремальна, має довгих вигнутий яр, що значно ускладнює відшукання точки мінімуму алгоритмам математичного

програмування, як використовують інформацію про похідні, так і задіє детермінований прямий пошук. Друга функція – багатоекстремального з вузькими зонами тяжіння локальних мінімумів, які незначно відрізняються один від одного по глибині, що робить таку функцію винятково складною для оптимізації. Обидві функції входять в стандартний набір тестових функцій для пошукових алгоритмів.

Алгоритм PSO практично не має параметрів, що настроюються, тому основний інтерес викликало питання про розмір і розподіл ресурсу, що приводить до найкращих результатів. Ресурс – це загальна кількість обчислень цільової функції, яка дозволена зробити алгоритму для відшукання оптимуму. Даний ресурс змінювався від 100 до 100 тисяч. Розподіл ресурсу змінювалося від ситуації, коли незначне число частинок еволюціонує тривалий час (наприклад, 10-1000), до ситуації, коли велика кількість частинок еволюціонує незначне по тривалості час (100-10). Проміжні варіанти, зрозуміло, теж розглядалися.

Порівняння ефективності стайного алгоритму і стандартного генетичного алгоритму проводилось на тих же функціях, що і дослідження ефективності алгоритму PSO. Для стандартного ГА заздалегідь були обрані оптимальні настройки (турнірна селекція, рівномірний схрещування, середня мутація). Так як стайня алгоритм такої настройки не вимагає, порівняння алгоритмів проводилося за різними розподілами виділеного ресурсу (10 тис. Обчислень для функції Гріванка і 2 тис. Обчислень для функції Розенброка).

1.2 Критерії оптимальності

Одним з важливих додатків теорії і методів оптимізації є проектування технічних об’єктів. Тому в системах автоматизованого проектування питань оптимізації приділяється значна увага.

Процедури параметричного синтезу в САПР виконуються або людиною в процесі багатоваріантного аналізу (в інтерактивному режимі), або реалізуються на базі формальних методів оптимізації (в автоматичному режимі). В останньому випадку знаходять застосування кілька постановок задач оптимізації.

Найбільш поширеною є детермінована постановка: задані умови працездатності на вихідні параметри і потрібно знайти номінальні значення проектних параметрів, до яких відносяться параметри всіх або частини елементів проектованого об'єкта. Назвемо це завдання оптимізації базової. В окремому випадку, коли вимоги до вихідних параметрів задані нечітко, до числа розраховуються величин можуть бути віднесені також норми вихідних параметрів, які фігурують в їх умовах працездатності.

Якщо проектуються вироби для подальшого серійного виробництва, то важливого значення набуває такий показник, як відсоток випуску придатних виробів в процесі виробництва. Очевидно, що успішне виконання умов працездатності в номінальному режимі не гарантує їх виконання при обліку виробничих похибок, що задаються допусками параметрів елементів. Тому метою оптимізації стає максимізація відсотка виходу придатних, а до результатів рішення задачі оптимізації відносяться не тільки номінальні значення проектних параметрів, а й їх допуски.

Базова задача оптимізації ставиться як задача математичного програмування:

$$extr \quad F(X), \; X \in D_x$$
$$D_x = \left\{ X \,\middle|\, \varphi(X) > 0, \psi(X) = 0 \right\}, \qquad (1.1)$$

де – $F(X)$ цільова функція,

X – вектор керованих (проектних) параметрів,

$\varphi(X)$ і $\psi(X)$ – функції-обмеження,

D_x – допустима область в просторі керованих параметрів.

Запис інтерпретується як завдання пошуку екстремуму цільової функції шляхом варіювання керованих параметрів в межах допустимої області.

Таким чином, для виконання розрахунку номінальних значень параметрів необхідно, по-перше, сформулювати завдання у вигляді, по-друге, вирішити задачу пошуку екстремуму *F(X)*.

Складність постановки оптимізаційних проектних завдань обумовлена наявністю у проектованих об'єктів кількох вихідних параметрів, які можуть бути критеріями оптимальності, але в задачі (1.1) цільова функція повинна бути одна. Іншими словами, проектні завдання є багатокритеріальної, і виникає проблема відомості багатокритеріальної задачі до однокритерійним.

Застосовують кілька способів вибору критерію оптимальності.

1.2.1 Часний критерій оптимальності

У приватному критерії серед вихідних параметрів один вибирають в якості цільової функції, а умови працездатності інших вихідних параметрів відносять до обмежень задачі (1.1). Ця постановка цілком прийнятна, якщо дійсно можна виділити один найбільш критичний вихідний параметр. Але в більшості випадків позначається недолік приватного критерію (рис. 1.3).

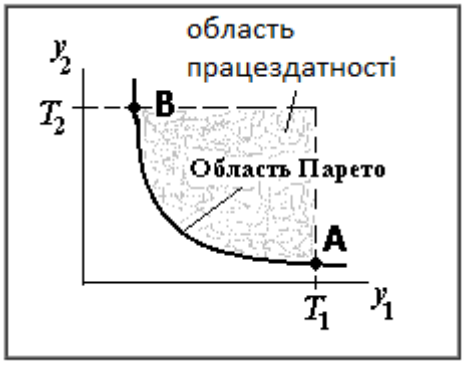

Рисунок 1.3 – Області Парето і працездатності

На цьому малюнку представлено двовимірне простір вихідних параметрів y_1 і y_2, для яких задані умови працездатності $y_1 < T_1$ і $y_2 < T_2$. Крива АВ є кордоном досяжних значень вихідних параметрів. Це обмеження об'єктивне і пов'язане з існуючими фізичними та технологічними умовами виробництва, званими умовами можливості бути реалізованим. Область, в межах якої виконуються всі умови можливості бути реалізованим і працездатності, називають областю працездатності. Безліч точок простору вихідних параметрів, з яких неможливо переміщення, що приводить до поліпшення всіх вихідних параметрів, називають областю компромісів, або областю Парето. Ділянка кривої АВ (див. Рис. 1.3) відноситься до області Парето.

Якщо в якості цільової функції в ситуації рис. 1 вибрати параметр у1, то результатом оптимізації будуть параметри Х, відповідні точці В. Але це межа області працездатності і, отже, при нестабільності внутрішніх і зовнішніх параметрів велика ймовірність виходу за межі області працездатності. Звичайно, результати можна поліпшити, якщо застосовувати так званий метод поступок, при якому в якості обмеження приймають умова працездатності зі скоригованої нормою у вигляді:

$$y_2 < T_2 + \Delta,$$

де Δ – поступка.

Але виникає проблема вибору значень поступок, тобто результати оптимізації матимуть суб'єктивний характер. Очевидно, що ситуація не зміниться, якщо цільовою функцією буде обраний параметр, у2 – оптимізація призведе в точку А.

1.2.2 Адитивний критерій

Адитивний критерій об’єднує (згортає) все вихідні параметри (приватні критерії) в одну цільову функцію, яка була зважену суму приватних критеріїв:

$$F(X) = \sum_{j=1}^{m} \omega_j y_j(X), \tag{1.2}$$

де w_j – ваговий коефіцієнт,

m – число вихідних параметрів.

Функція (2) підлягає мінімізації, при цьому якщо умова працездатності має вигляд $y_j < T_j$, то $w_j < 0$.

Недоліки адитивного критерію – суб’єктивний підхід до вибору вагових коефіцієнтів і неврахування вимог ТЗ. Дійсно в (1.2) не входять норми вихідних параметрів.

1.2.3 Мультиплікативний критерій

Мультиплікативного критерій має аналогічні недоліки з адитивним критерієм, цільова функція якого має вигляд:

$$F(x) = \prod_{j=1}^{m} y_j^{\omega_j}(X), \tag{1.3}$$

Неважко бачити, що якщо прологарифмувати (1.3), то мультиплікативний критерій перетворюється в адитивний.

1.2.4 Максимінний критерій

Більш кращим є максимінний критерій, як цільова функція якого приймають вихідний параметр, найбільш неблагополучний з позицій виконання умов працездатності. Для оцінки ступеня виконання умови працездатності j-го вихідного параметра вводять запас працездатності цього параметра Sj і цей запас можна розглядати як нормований j-й вихідний параметр.

Наприклад (тут і далі для лаконічності викладення передбачається, що всі вихідні параметри приведені до вигляду, при якому умови працездатності стають нерівностями в формі $y_j < T_j$):

$$S_j = \frac{T_j - y_j}{T_j}, \text{ або } S_j = \frac{T_j - y_{номj}}{\delta_j},$$

де $y_{номj}$ – номінальне значення,

а δ_j – деяка характеристика розсіювання *j*-го вихідного параметра, наприклад, тресигмовий допуск.

Тоді цільова функція в максимінному критерії є:

$$F(X) = \min_{j \in [1;m]} S_j(X).$$

Тут запис [1;m] означає множину цілих чисел в діапазоні від 1 до m. Завдання (1.1) при максимінному критерії конкретизується наступним чином:

$$F(X) = \max_{X \in D_X} \min_{j \in [1;m]} S_j(X),$$

де допустима область D_X визначається тільки прямими обмеженнями на керуючі параметри: $x : x_{i\min} < x_i < x_{i\max}$.

1.3 Алгоритми інтелектуального пошуку

Методи інтелектуального пошуку, зазвичай звані пошуком табу, набули широкого поширення і визнання в рішенні практичних оптимізаційних задач. Додатки цих методів стрімко розширюються в таких областях, як управління ресурсами, проектування процесів, логістика, планування і загальна комбінаторна оптимізація. Поєднання з іншими методами, як евристичними, так і алгоритмічними, також дає продуктивні результати.

У цьому параграфі розглядаються основні характеристики пошуку табу, які найбільшою мірою впливають на успішність його застосування, а також наводяться основні відомості, що дозволяють побудувати поліпшені методи інтелектуального пошуку для цілей підтримки прийняття рішення.

Метод пошуку табу (TS) є метаеврістикой, яка управляє процедурою локального евристичного пошуку з метою глобального дослідження простору рішень. В останні роки стрімко розширюється область додатків методу пошуку табу в оптимізації. Процедури TS, що включають основні концепції та гібридні схеми, що поєднують ці концепції з іншими евристичними і алгоритмічними методами, успішно застосовувалися в різних практичних завданнях.

Пошук табу заснований на припущенні, що процес вирішення завдань, що претендує на звання «інтелектуального», повинен включати адаптивну пам’ять і ретельне дослідження ситуації.

Використання адаптивної пам’яті контрастує з підходами до безтями, які прийшли з біології та фізики, і з методами з негнучкою пам’яттю, прикладом якої може служити метод гілок і меж, а також його модифікації. Основною ідеєю в пошуку табу як при детермінованому, так і при стохастичному підході, є припущення, що поганий вибір, заснований на продуманій стратегії, може принести більше інформації, ніж хороший випадковий вибір. У системах, що використовують пам’ять, поганий вибір, заснований на стратегії, може забезпечити придатною до використання інформації про те, як стратегія може бути вигідно змінена. Навіть в просторі зі значним впливом випадковості, яка тим не менш недостатня для того, щоб знищити всі сліди порядку в більшості реальних задач, цілеспрямоване проектування може бути більш успішним в розкритті внутрішньої структури завдання і тим самим дати шанс використовувати умови в тих ситуаціях, коли випадковість не є всеподавляючою. Основні характеристики пошуку табу наведені в таблиці 1.1.

Таблиця 1.1 Основні характеристики пошуку табу

Основні риси пошуку табу
Адаптивна пам’ять
Вибірковість (включаючи стратегічне забування)
Абстракція і декомпозиція (з явною і атрибутивної пам’яттю)
Фактор часу:
новизна подій
частота подій
відмінність між короткими і довгими інтервалами часу
Якість і вплив:

відносна привабливість альтернатив масштаби змін в структурі або обмежують відносинах *Контекст*: регіональна взаємозалежність структурна взаємозалежність послідовна взаємозалежність
Ретельне дослідження ситуації Стратегічно накладаються обмеження і переваги (умови табу і рівні аспірації) Концентрація уваги на хороших регіонах і властивості оптимальних рішень (інтенсифікація процесів) Встановлення і дослідження нових обіцяють областей (диверсифікація процесів) Немонотонні стратегії пошуку (стратегічна осциляція) Інтеграція і розширення рішень (перез'єднання шляхів)

Пошук табу призначений для знаходження нових і більш ефективних шляхів отримання поліпшених рішень за допомогою концепцій, наведених в таблиці 1.1, і для визначення відповідних принципів, які можуть розширити основи інтелектуального пошуку.

Як це зазвичай трапляється, з'являються нові поєднання базових ідей, що призводить до поліпшеним рішенням і найкращим практичних результатів. Це робить TS привабливою областю для досліджень і використання в практичних завданнях. Останнім часом з'явилися гібридні схеми, що поєднують пошук табу не тільки з локальним пошуком, але і з еволюційними алгоритмами і імітацією відпалу, які і самі по собі є глобальними пошуковими процедурами. Застосування в практичних завданнях показує, що ці підходи мають велике майбутнє.

Основні положення методу можуть бути описані в такий спосіб. Задана функція *f(x)*, яка повинна бути оптимізована в безлічі *X*. TS починає як і звичайний локальний пошук, ітеративно обробляючи одну точку (рішення) за одною до тих пір, поки не буде задоволений обраний критерій зупинки. Кожне $x \in X$ має околиця $N(x) \subset X$, і кожне рішення $x' \in N(x)$ можна досягти з *x* за допомогою операції, званої переміщенням.

TS виходить за рамки локального пошуку шляхом застосування стратегії модифікації околиці *N(x)* по ходу пошуку, ефективно замінюючи її на іншу околицю *N*(x)*. Як видно з вищесказаного, ключовий аспект TS полягає у використанні спеціальних структур пам'яті, які служать для визначення *N*(x)*, і в організації траєкторії, по якій досліджується простір оптимізації.

Рішення, що включаються в *N*(x)* цими структурами пам'яті, визначаються декількома способами. Один з них, що дав пошуку табу його назва, визначає рішення, з якими стикається на певному рівні (і, неявно, додаткові пов'язані з ними рішення), і забороняє їм належати *N*(x)*, класифікуючи їх як табу. (Термін «табу» призначений для висловлювання на кшталт обмеження, яке має більше «культурне» значення, тобто таке значення, яке знаходиться під впливом історії і конкретних умов і може бути змінено, якщо умови дозволять).

Процес, за допомогою якого рішення набувають статусу табу, має кілька складових, розроблених для того, щоб забезпечити обґрунтовано сувору перевірку нових точок. Корисним способом організації цього процесу є заміна звичайного оцінювання рішень оцінками табу, які вводять штрафи, щоб значно скоротити вибір рішень, тобто визначити ті з них, які краще виключити з *N*(x)* відповідно до їх залежністю від елементів, що становлять статус табу. Крім того, оцінки табу також періодично включають заохочення, щоб збільшити вибір рішень інших типів в результаті якихось додаткових міркувань про переваги і впливу довготривалих чинників.

Слід підкреслити, що поняття околиці, що застосовується в пошуку табу, відрізняється від застосовуваного в локальному пошуку, охоплюючи типи переміщень, що застосовуються в конструктивних і деструктивних процесах (підстави для таких переміщень називаються, відповідно, конструктивними і деструктивними околицями). Таке використання розширеного поняття околиці зміцнює фундаментальну перспективу TS, яка повинна визначати околиці динамічними способами, щоб мати можливість включати послідовні або паралельні операції з багатьма типами переміщень.

З огляду на те, що TS має кілька найважливіших компонент, і завдання об'єднання їх може здатися на перший погляд надмірно трудомісткою, багато розробки були засновані тільки на перших ідеях, що розглядаються до того ж в самій загальній постановці. Однак треба підкреслити, що число найважливіших компонент не так вже й велика (кожна з декількома основними варіантами), і, один раз систематизовані, вони становлять взаємопов'язану схему, яка виявляється значно ефективнішою, ніж одна-дві компоненти окремо.

Рівні аспірації. Розширюючи визначення умов табу, введемо ще один важливий елемент, що надає пошуку табу додаткову гнучкість, визначивши його як рівень аспірації (перевагу, привабливості). Статус табу для вирішення не є абсолютним, а може бути скасований, якщо з'явилися певні умови, що виражаються у формі рівнів аспірації. Такі рівні аспірації задають певні пороги привабливості, які вказують чи можна розглядати рішення як допустимі, незважаючи на те, що вони класифіковані як табу. Подібний критерій аспірації може бути визначений на підмножині рішень, які належать одному регіону або всі мають специфічні властивості (як деяке приватне значення функції або рівень неприпустимість). Наприклад, один з таких критеріїв аспірації базується на визначенні умовно «найкращих» значень цільової функції, які можуть бути досягнуті за допомогою переміщень, що починаються з деяких інтервалів значень $f(x)$. Потім переміщення вважається

прийнятним, якщо воно може досягти нового кращого значення для інтервалу, з якого воно починається.

Згаданий підхід природно узагальнюється заміною інтервалів значень цільової функції іншими типами інтервалів. У цьому випадку часто краще ставити умову таким чином, щоб переміщення досягало кращого щодо інтервалу значення цільової функції в кінці переміщення, а не на початку. Це більш точно відповідає стандартному табу-обмеження, виключаючи те, що воно використовується для подолання інших обмежень. (Неявно воно відповідає спеціальному типу кон'юнкції.)

Стратегії списку кандидатів. Активність пошуку табу підкріплюється пошуком найкращого доступного переміщення, яке може бути визначено з прийнятними додатковими витратами. Слід пам'ятати, що розуміння найкращого не обмежується значенням цільової функції. (Як уже зазначалося, оцінки табу перебувають під впливом штрафів і стимулів, які визначаються історією пошуку, а також під впливом деяких міркувань про вплив, які будуть охарактеризовані пізніше). Для ситуацій, коли *N(x)* велика або оцінювання її елементів дорого, стратегії списку кандидатів використовуються для обмеження числа рішень, що перевіряються на даній ітерації.

Так як припис пошуку табу відбирати елементи обґрунтовано є виключно важливим, вирішальними для процесу пошуку є ефективні правила для генерування і оцінювання хороших кандидатів. Навіть там, де стратегії списку кандидатів не використовуються явно, структури пам'яті для ефективного поновлення оцінок переміщень від однієї ітерації до іншої і для зменшення витрат на перебування найкращих або близьких до них переміщень, часто інтегровані в розробку TS. Розумне оновлення може значно зменшити час отримання рішення, а використання стратегій списку кандидатів явно для великих завдань може значно підвищити результати.

Подання штрафів як «великого» або «дуже маленького» висловлює пороговий ефект. Раніше ми ставили статус табу як умова типу «все-або-

нічого», але ясно, що можлива диференціація, наприклад, з урахуванням різного числа табу активних атрибутів або з різницею рівнів не закінчилися термінів табу. Статус табу в загальному випадку відповідає такому використанню штрафів, коли в список табу відбираються рішення з дуже поганими оцінками (великими штрафами), і навпаки, служить для збереження зв'язків між дуже хорошими рішеннями, коли потрібно блокувати зміни, що порушують набори атрибутів, стабільно дають високі оцінки.

Якщо всі доступні в даний момент переміщення призводять до рішень, які є табу і повинні б бути виключеними з подальшого розгляду, використання штрафу призводить до вибору рішення, відповідного «найменшому табу». Цей підхід дає можливість уникнути крайніх варіантів, коли або все переміщення можуть бути обраними (звичайний пошук), або небажані переміщення взагалі не можуть бути обрані (пошук табу типу «все або нічого»).

Оцінка табу може бути модифікована також шляхом створення стимулів, заснованих на рівнях аспірації, також як ця оцінка модифікується на основі штрафів, заснованих на статус табу. У цьому сенсі умови табу і умови аспірації можна розглядати як «дзеркальні відображення» один одного.

Довготривала пам'ять. У деяких додатках компоненти короткочасної пам'яті в TS здатні давати рішення дуже високої якості. Однак, в загальному випадку, TS стає значно сильніше при включенні в нього довготривалої пам'яті і пов'язаних з нею стратегій. (Велике число реалізацій TS, що містять тільки короткочасну пам'ять, було надалі значно покращено введенням компонентів довготривалої пам'яті).

При розгляді більш тривалих періодів часу головним аспектом є спеціальні типи пам'яті, заснованої на частоті. Їх робота забезпечується введенням штрафів і стимулів, які визначаються відносним інтервалом часу, протягом якого атрибути належали рішенням, перевіреним при пошуку.

Дозволяється також робити відмінності для різних областей пошуку. Частоти переходів зберігають інформацію про те, як часто атрибути змінювалися, в той час як резидентні частоти дають інформацію про відносні проміжках часу, коли атрибути входили в генеруються рішення. Ці види пам’яті іноді супроводжуються розширеними формами пам’яті, заснованої на недавньому минулому.

Можливо буде несподіваним той факт, що використання довготривалої пам’яті не вимагає тривалої роботи процедури пошуку для того, щоб вигода від неї стала очевидною. Часто її гідності починають проявлятися через відносно короткий проміжок часу, що дозволяє запобігти зайві зусилля за рішенням завдання завдяки тому, що може бути знайдено рішення дуже високої якості за дуже короткий час. Наприклад, найшвидші методи вирішення завдань складання розкладів різного типу засновані на включенні довготривалої пам’яті (як явною так і атрибутивної). З іншого боку, також справедливо, що шанси знайти більш гарне рішення – у разі, коли оптимальне рішення ще не знайдено – з ростом часу підвищуються при використанні довготривалої пам’яті на додаток до короткочасної.

Інтенсифікація та диверсифікація. Два надзвичайно важливих компонента TS – це стратегії інтенсифікації та стратегії диверсифікації. Стратегії інтенсифікації засновані на модифікації правил вибору для заохочення тих комбінацій переміщень і властивостей рішень, які були визнані вдалими протягом попереднього часу. Вони можуть бути застосовані з конструктивними і деструктивними околицями, а також з околицями переходів, шляхом перезапуску пошукових процедур, які включають хороші атрибути в поточну структуру алгоритму з урахуванням вже включених атрибутів. Такі підходи добре працювали при виробленні правил вибору для імовірнісного пошуку табу. Стратегії інтенсифікації можуть також ініціювати повернення до багатообіцяючим регіонах для більш докладного їх дослідження.

Стратегія вибору елітних рішень виключно важлива. Доведено, що три варіанти є виключно вдалими. У деяких роботах представлена міра диверсифікації для забезпечення того, що записані рішення відрізняються один від одного в потрібному ступені, і подальшого очищення короткочасної пам’яті перед записом кращих рішень. Другий варіант зберігає послідовний список обмеженої довжини, в кінець якого додає нове рішення тільки в разі, якщо воно краще за всіх, знайдених раніше. Поточний останній член списку завжди вибирається (і віддаляється) в якості базису для поновлення пошуку. Однак короткочасна пам’ять TS, відповідна цьому рішенню, також зберігається, що забороняє використовувати в якості першого переміщення раніше використане переміщення з цього рішення, так що буде розпочато новий шлях отримання рішення.

Третій варіант працює з впорядкованим списком з k кращих рішень. Через певний число ітерацій починається перебір з гіршого члена списку у напрямку до кращого. Поточний відібраний член породжує новий процес, використовуючи імовірнісний пошук табу як альтернативу простому відновлення колишньої пам’яті. Виконується тільки фіксоване число ітерацій перед запуском нового процесу, проте список оновлюється безперервно. Таким чином, коли знайдено більше гарне рішення ніж наявне найгірше, це нове рішення вставляється в список на відповідне місце, а найгірше видаляється зі списку. Доведено високу ефективність цього методу для задач проектування мереж телекомунікації.

Ці підходи – приклад того, що іноді називається підходом з реструктурування переміщенням, відображаючи той факт, що звичайне безліч переміщень періодично модифікується з тим, щоб дозволити прямий стрибок до вирішення, який лежить за межами поточної околиці. При такому підході зберігається інформація про найкращі неперевірених ще сусідніх рішеннях (включених в список кандидатів) з обмеженням на специфічні типи рішень, типу сусідів локальних оптимумів або сусідів рішень, перевірених безпосередньо перед досягненням таких локальних оптимумів. Незважаючи

на те, що ця стратегія «неперевірених сусідів» ще недостатньо досліджена, було помічено, що схожі стратегії, забезпечували рішення хорошої якості.

Інший тип інтенсифікації – інтенсифікація декомпозицією, коли накладаються обмеження на деякі частини завдання або структури рішення для того, щоб отримати такий спосіб декомпозиції, який би дав можливість більше сконцентруватися на інших частинах структури. Класичний приклад – завдання про комівояжера, коли ребра, які належать перетинанню елітних турів, можуть бути «назавжди» включені в рішення, для того, щоб зосередити увагу на маніпулюванні іншими частинами маршруту. Використання перетинів може бути розглянуто як крайній приклад більш загальної стратегії, яка дозволяє ідентифікувати і обмежувати значення строго визначених і послідовних змінних. У цьому підході інформація про частоти зберігає відомості про змінних, які приймають деякі конкретні значення (або значення з деяких інтервалів) на підмножин елітних рішень. Якість рішень, в яких реалізуються ці значення, і руйнує вплив зміни цих значень і дають ступінь їхньої переваги. Обмеження значень відповідних змінних з використанням подібної інформації може призвести до ідентифікації додаткових змінних, які також слід обмежити з тих же причин, що привносить рекурсивність в цей підхід. Загальний ефект може бути порівняний зі створенням комбінаторного скорочення числа можливостей (по зворотній аналогії з подібним виразом «комбінаторний вибух»), так як обмеження дискретних змінних, наприклад шляхом тимчасової їх фіксації або видалення, призводить в точності до протилежного ефекту в порівнянні з додаванням нових дискретних змінних. Цей тип інтенсифікації був дуже ефективно застосований до задачі складання маршрутів для транспортних засобів.

Інтенсифікація декомпозицією призводить до розгляду і інших типів стратегій, заснованих не тільки на показниках сили і послідовності, але і на можливостях продуктивної взаємодії деяких елементів. Для задач на перестановках, які можуть бути зручно описані в термінах графів (наприклад,

складання розкладів, маршрутизація транспортних засобів, завдання комівояжера), декомпозиція може ґрунтуватися на ідентифікації деяких підланцюгів елітних рішень, де дві або більше підланцюгів можуть бути призначені в загальне безліч, якщо вони містять вузли, які «дуже привабливі», щоб з'єднати їх з вузлами інших підланцюгів безлічі. З ребром, що розчленовує набір ланцюжків, може працювати процес інтенсифікації, який оперує паралельно з кожним безліччю при обмеженні, що початкові і кінцеві вершини підланцюгів залишаються незмінними. В результаті такої декомпозиції можуть бути знайдені нові кращі безлічі підланцюгів, які створять нове рішення. Такий процес може бути застосований для збільшення альтернативних видів декомпозицій в більш широких формах інтенсифікації декомпозиції.

Стратегії диверсифікації. Стратегії диверсифікації TS, як випливає з назви, розробляються для того, щоб направляти пошук в нові області. Часто вони засновані на модифікації правил вибору, які вводять деякі атрибути в рідко використовуються рішення. Інший шлях – вони можуть вводити такі атрибути шляхом часткового або повного перезапуску процесу пошуку рішення. В якості основи для таких процедур можна використовувати ті ж види пам'яті, що були описані раніше, хоча вони повинні працювати з іншими (взагалі кажучи більшими) підмножинами рішень, ніж оброблювані в стратегіях інтенсифікації.

Однак необхідно підкреслити, що при введенні диверсифікації такого типу дуже важливим є правильний вибір моменту часу. Диверсифікація не може застосовуватися довільним чином, а повинна використовуватися тільки в зоні локальних оптимумів. Крім того, кращі переміщення все ще використовуються для управління процесом пошуку, хоча і штрафуються, причому штрафи мають обмежений період дії. Локальний оптимум, який досягається пошуком табу при використанні даного підходу і використовується для початку послідовності кроків диверсифікації, природно, може відрізнятися від справжніх локальних оптимумів з огляду на

те, що правила вибору пошуку табу можуть просто виключити деякі переміщення, які могли б привести до поліпшення.

Успіх цього підходу – заслуга включення варіанту TS, який завжди продовжує рухатися до локального оптимуму, як тільки поліпшує переміщення стає альтернативою, яку можна прийняти, ґрунтуючись на критерії аспірації (привабливості), який активується тільки після виконання поліпшує переміщення. У цьому підході так довго, поки існують поліпшують переміщення, критерій привабливості дозволяє одному з них бути обраним шляхом оцінювання табу, яке штрафує вибори, ґрунтуючись на їх стані табу (загострюючи увагу на покращувати безлічі). Як тільки досягається істинний локальний оптимум, спеціальний критерій привабливості переривається поки стандартними правилами TS НЕ буде вибрано нове поліпшує переміщення. Цей підхід втілює поняття привабливості напрямки пошуку.

Стратегічна осциляція тісно пов’язана з основами TS і забезпечує положення для отримання ефективної взаємодії між прискоренням і уповільненням протягом тривалого проміжку часу. Підхід працює шляхом орієнтації переміщень щодо критичного рівня, який визначається як стан конструкції або обраний інтервал значень функціоналу.

Такий критичний рівень часто представляється точкою, де метод повинен нормально зупинятися. Однак замість того, щоб зупинятися тоді, коли цей рівень досягається, правила для вибору переміщень змінюються для того, щоб вирішувати перетин області, певної критичним рівнем. Метод тоді продовжує роботу до певної глибини поза межами критичного рівня, і його робота поновлюється. Потім метод знову наближається до критичного рівня і перетинає його з протилежного напрямку, і метод переходить до нової позначці розвороту.

Процес неодноразового наближення і перетину критичного рівня з різних напрямків породжує «коливання» стратегії методу, яке і дало методу ім’я. Управління цим поведінкою встановлюється шляхом генерації змінних оцінок і правил руху в залежності від керованої області та напрямки пошуку.

Стандартні механізми TS уникають можливості повторного розгляду попередньої траєкторії.

Такі зміни правил, засновані на напрямку і фазі пошуку, – типові особливості стратегічної осциляції і забезпечують розширену евристику, яка називається «життєздатністю». Додаток різних правил може супроводжуватися перетином кордону до різної глибини на різних сторонах. Ця можливість призначена, щоб наближатися до кордону і відступати від неї без перетину, тобто вибираючи перетин «нульовий глибини». Кажуть про конструктивну і деструктивному типі стратегічної осциляції, коли конструктивні кроки «додають» елементи, а деструктивні кроки ці елементи «видаляють». Як тільки конструктивна фаза, що складається з ряду операцій «додати переміщення», завершується, виконується найбільш привабливе дію – «видалити переміщення». Однак TS структури пам’яті також необхідні, щоб гарантувати, що чергуються фази від дійсної відміни один одного.

При умовної оптимізації осциляція може також функціонувати, збільшуючи і зменшуючи межі порушеності для обмежень. Такий підхід став основою для великого числа ефективних додатків, де межі представляється такими поняттями, як значення цільової функції і допустимість рішення для того, щоб вести пошук і досліджувати на різну глибину пов’язані області.

Корисна інтеграція стратегій прискорення і уповільнення відбувається в підході, званому повторним компонування маршруту. Цей підхід генерує нові рішення, досліджуючи траєкторії, які «з’єднують» елітні рішення – починаючи з одного з цих рішень, званого рішенням ініціалізації, і генеруючи маршрут в просторі околиць, який веде до інших рішень, що викликається керівними рішеннями. Це виконується шляхом вибору переміщень, які представляють атрибути, що містяться в керівних рішеннях.

Підхід може розглядатися як крайній приклад стратегії, яка намагається включити атрибути рішень високої якості, створюючи стимули для перевагу цих атрибутів в обраних переміщеннях. Однак замість того, щоб використовувати спонукання, яке просто заохочує включення таких

атрибутів, підхід повторної компонування маршруту підпорядковує всі інші розгляду мети вибору переміщень, які представляють атрибути керівних рішень для того, щоб створити «хорошу композицію атрибутів» в поточному рішенні. Композиція при кожному кроці визначається шляхом вибору кращого переміщення, використовуючи звичайні критерії вибору з обмеженого набору переміщень, яке включає максимальний кількість (або максимальне зважене значення) атрибутів керівних рішень. Як і в інших додатках TS, критерії привабливості можуть скасовувати це обмеження, щоб дозволяти розгляд інших переміщень особливо високої якості.

1.4 Постановка задачі дослідження

Об’єктом дослідження є процес еволюційної оптимізації на основі кластеризації змішаних даних, зокрема алгоритму оптимізації лева та порівняльний аналіз з відомими методами кластеризації змішаних даних.

Метою дослідження є аналіз працездатності еволюційних алгоритмів для кластеризації змішаних даних на прикладі алгоритму оптимізації лева та деяких реальних даних зі сховища даних UCI.

Для досягнення мети необхідно вирішити такі завдання:

– проаналізувати існуючі еволюційні алгоритми та критерії оптимальності;

– змоделювати математичну модель алгоритму оптимізації лева для кластеризації змішаних даних;

– змоделювати комп’ютерну модель алгоритму для тестування кластеризації існуючих даних;

– зробити порівняльний аналіз результатів роботи еволюційного алгоритму у порівнянні з іншими існуючими алгоритмами кластеризації.

На виході має бути реалізований та досліджений алгоритм оптимізації лева та проаналізований результат його роботи в порівнянні з іншими алгоритмами кластеризації.

2 МАТЕМАТИЧНА МОДЕЛЬ МЕТОДУ ЕВОЛЮЦІЙНОЇ ОПТИМІЗАЦІЇ НА ОСНОВІ АЛГОРИТМУ КЛАСТЕРИЗАЦІЇ ЗМІШАНИХ ДАНИХ НА ПРИКЛАДІ LION OPTIMIZATION

2.1 Алгоритм кластеризації K-Prototype на основі Lion Optimization

K-Prototype Clustering – це ефективний алгоритм для кластеризації змішаних типів наборів даних. Залежність алгоритму від ініціалізації центрів є серйозною проблемою, і він зазвичай застряє в локальному оптимумі. Щоб вирішити цю проблему, поєднуються алгоритми Lion Optimization Algorithm і K-Prototype. У цьому методі процес ініціалізується групою випадкової сукупності (Лев). Алгоритми K-прототипу на основі оптимізації Lion складаються з наступних кроків:

Алгоритм: алгоритм кластеризації K-прототипу на основі оптимізації Lion.

Вхідні дані: змішані дані, параметри *Npop* як населення, *%N* як лев-кочівник, *%S* як самка лева, ітерація *K*, число кластерів *K*, призначте поріг σ.

Вихід: кластер *K*

Процедура:

Крок 1. Розділіть категоріальні дані та числові дані.

Крок 2. Змініть категоричні дані на числові за допомогою *K-Mode* за допомогою:

$$P(W,Q)=\sum_{l=1}^{K}\sum_{i=1}^{n} w_i d_i d_{sim}(x_i,q_l)\,. \qquad (2.1)$$

Крок 3. Попередня обробка даних.

Крок 4. Створення початкової популяції левів *Npop*.

Крок 5. Ініціалізація кочовника і прайду.

– Випадковим чином виберіть $\%N$ як лева -кочівника і розділіть інших левів на прайд.

– У кожному прайді $\%S$ всього населення є самка лева.

Крок 6. Кожну стать лева кочівника сортують за значенням придатності:

$$f_i = \sum_{i=1}^{c} \sum_{d_m \varepsilon c_{cen}} \left\| d_m - c_{cen} \right\|^2 . \tag{2.2}$$

Крок 7. Розташуйте значення придатності від мінімальних до максимальних значень.

Крок 8. Виберіть мінімальне значення зі значення фітнесу.

Крок 9. Повторюйте кроки 4-7, поки не буде виділено скупчення центроїда.

Крок 10. Застосуйте значення придатності в алгоритмі кластеризації K-Prototype.

Крок 11. Обчисліть евклідову відстань за допомогою:

$$d(x, y) = \sqrt{\sum_{i-1}^{n} (x_i - y_i)^2} . \tag{2.3}$$

Крок 12. Перерахуйте центроїд.

Крок 13. Повторюйте кроки 10-12, доки кластеризація не буде змінена.

У цій роботі запропоновано алгоритм кластеризації K-прототипів на основі Lion Optimization, об’єднавши переваги алгоритму Lion Optimization з існуючим алгоритмом K-Prototype, щоб досягти глобального оптимального кластерного рішення. Запропонований алгоритм було перевірено на п’яти контрольних наборах даних, які містять як числові, так і категоріальні атрибути. Доведено, що продуктивність запропонованого алгоритму

перевершує продуктивність звичайних алгоритмів кластеризації K-Means та K-Prototype.

Леви є найбільш соціально схильними з усіх видів диких кішок, які демонструють високий рівень співпраці та антагонізму. Леви представляють особливий інтерес через їх сильний статевий диморфізм як у соціальній поведінці, так і в зовнішньому вигляді. Лев – дика тварина з двома типами соціальної організації: мешканці та кочівники. Мешканці живуть групами, які називають прайдом. Прайд левів зазвичай включає близько п’яти самок, їх дитинчат обох статей і одного або більше дорослих самців. Молоді самці виключаються з гордості при народженні, коли стають статевозрілими. Як згадувалося раніше, друга організаційна поведінка називається кочівниками, які пересуваються спорадично, парами або поодиноко. Пари частіше зустрічаються серед споріднених чоловіків, які були виключені з їхньої материнської гордості. Зверніть увагу, що лев може змінити спосіб життя; жителі можуть стати кочівниками і навпаки.

На відміну від усіх інших кішок, леви зазвичай полюють разом з іншими членами свого прайду. Кілька левиць працюють разом і оточують здобич з різних точок і ловлять жертву швидкою атакою. Узгоджене групове полювання приносить більшу ймовірність успіху в полюванні на левів. Леви та левиці-самці зазвичай залишаються і відпочивають, чекаючи, поки левиці повернуться з полювання. Леви спарюються в будь-який час року, а самки є поліестровими (коли самки, які не виховують своїх дитинчат, сприйнятливі). Левиця може спарюватися з кількома партнерами, коли вона в спеку. У природі самці та жінки левів позначають свою територію та інші місця, що здається хорошим місцем із сечею.

У цій роботі деякі персонажі левів математично моделюються з метою розробки алгоритму оптимізації. У запропонованому алгоритмі Lion Optimization Algorithm (LOA) початкова сукупність формується набором випадково згенерованих рішень, які називаються Левами. Деякі з левів у початковій популяції ($\%N$) вибираються як леви-кочівники, а решта

популяція (постійні леви) випадковим чином розподіляється на Рпіднабори, звані прайдами. Частина членів прайду розглядається як жінка, а решта – як чоловіки, тоді як цей показник (статевий показник (*%S*)) у кочових левів – навпаки.

Для кожного лева найкраще отримане рішення в пройдених ітераціях називається найкращою відвідуваною позицією, і під час процесу оптимізації поступово оновлюється. У LOA це територія гордості, яка складається з кожного члена, який найкраще відвідувати. У кожному прайді деякі самки відбираються випадковим чином. Мисливці рухаються до здобичі, щоб оточити і зловити. Решта самки рухаються до різних позицій території. Самці леви в гордості, бродять по території. Самки inprides спаровуються з одним або кількома постійними самцями. У кожному прайді молоді самці виключаються зі свого материнського прайду і стають кочівниками, коли досягають зрілості, і їхня сила менша, ніж у постійних самців.

Крім того, лев-кочівник (і самець, і жінка) випадковим чином переміщається в просторі пошуку, щоб знайти краще місце (рішення). Якщо сильний самець-кочівник вторгається на постійного самця, кочовий лев виганяє його з прайду. Кочівник стає постійним левом. В процесі еволюції деякі мешканці жіночої статі іммігрують з одного прайду в інший або змінюють свій спосіб життя і стають кочівниками, і навпаки, деякі кочові леви приєднуються до прайдів. Через багато факторів, таких як брак їжі та конкуренція, найслабший лев помре або буде вбитий. Вищезазначений процес триває до тих пір, поки не буде виконана умова зупинки.

Далі буде запропоновано детальний опис кроків алгоритму Lion Optimization:

- ініціалізація;
- полювання;
- рух до безпечного місця;
- роумінг;

- спаровування;
- оборона;
- міграція;
- популяційна рівновага левів;
- конвергенція.

2.2 Ініціалізація

LOA – це мета-евристичний алгоритм на основі сукупності, у якому першим кроком є випадкове генерування сукупності в просторі розв'язків. У цьому алгоритмі кожне рішення називається «Левом». У задачі оптимізації розмірів *Nvar* Лев представлений таким чином: $Lion = [\, x_1;\, x_2;\, x_3;\, \ldots;\, x_{Nvar} \,]$. Вартість (цінність придатності) кожного Лева обчислюється шляхом оцінки функції вартості, як: $fitnessValueOfLion = f(Lion) = f(x_1;\, x_2;\, x_3;\, \ldots;\, x_{Nvar})$.

На першому кроці рішення *Npop* генеруються випадковим чином у просторі пошуку. *%N* згенерованих рішень випадковим чином вибираються як кочові леви. Решта населення буде випадковим чином поділена на P-прайдів. Кожне рішення в цьому алгоритмі має певну стать і залишається незмінним під час процесу оптимізації. Щоб наслідувати цей факт, у кожному прайді *%S (%75–%90)* усієї популяції, сформованої на останньому етапі, відомі як жінки, а решта як чоловіки. Для левів-кочівників це співвідношення є навпаки *%(1S)*. У процесі пошуку кожен лев позначає найкраще місце для відвідування. Відповідно до цих позначених позицій формується територія кожного прайду. Отже, для кожного прайду територію цього прайду утворюють позначені позиції (найкраще відвідувані позиції) його учасниками.

2.3 Полювання

У кожному прайді якась самка шукає здобич у групі, щоб забезпечити їжу для свого прайду. Ці мисливці мають спеціальні стратегії, щоб оточити здобич і зловити її. Загалом леви під час полювання дотримувалися приблизно однакових схем. Стендер розділив левів на сім різних ролей переслідування, показаних на рисунку 2.1, групуючи ці ролі на позиції лівого крила, центру та правого крила. Під час полювання кожна левиця коригує своє положення, виходячи з власного положення і позицій членів групи.

У зв'язку з тим, що під час полювання деякі з цих мисливців оточують здобич і атакують з протилежної позиції, ми використовуємо навчання на основі опозицій (OBL). Доведено, що базова концепція навчання на основі опозицій (OBL) є ефективним методом розв'язання задач оптимізації.

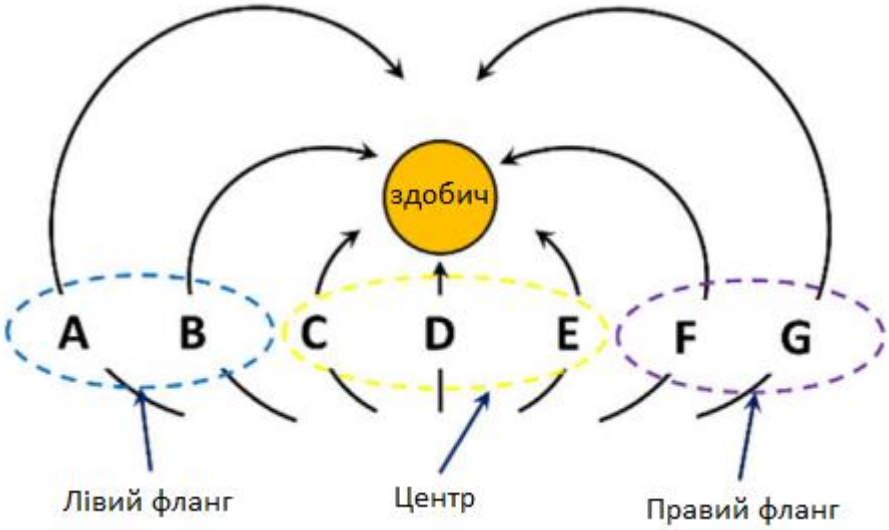

Рисунок 2.1 – Схема узагальненої поведінки полювання на левів

Хай $X(x_1;\ x_2;\ x_3;\ \ldots;\ x_{Nvar})$ буде точкою в *Nvar*-вимірному просторі, де $x_1;\ x_2;\ x_3;\ \ldots;\ x_{Nvar}$ є дійсними числами і $x_i\ [a_i;b_i], i = 1;\ 2;\ 3;\ldots; Nvar$. Протилежна точка точці X виражається як

$\breve{X}(\breve{x}_1;\ \breve{x}_2;\ \breve{x}_3;\ \ldots;\ \breve{x}_{Nvar})$, де $\breve{x}_i = a_i + b_i - x_i; i = 1;\ 2; \ldots; Nvar$. Принцип навчання, орієнтованого на опозиції (OBL) наведено на рисунку 2.2.

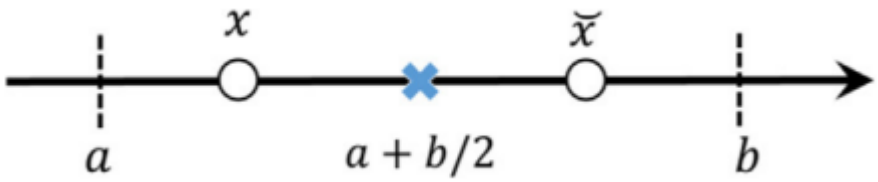

Рисунок 2.2 – Протилежна точка, визначена в області *[a, b]*. *x* є рішенням-кандидатом, а $\breve{x}$ протилежне *x*.

Відповідно до вищезгаданих фактів мисливці випадковим чином поділяються на три підгрупи. Група з найбільшою кумулятивною витонченістю учасників вважається центром, а дві інші групи розглядаються як два крила. У центрі мисливців розглядається фіктивна здобич ($ЗДОБИЧ = \sum мисливців(x_1;\ x_2;\ x_3;\ \ldots;\ x_{Nvar})\ /\ кількість\ мисливців$). Під час полювання мисливці обираються один за одним випадковим чином, і кожен обраний мисливець нападає на фіктивну здобич, що ця процедура буде визначена пізніше відповідно до групи, до якої належить обраний лев. Протягом усього полювання, якщо мисливець покращує власну витонченість, ЗДОБИЧ(PREY) втече від мисливця, і нова позиція ЗДОБИЧИ буде отримана наступним чином:

$$PREY' = PREY + rand(0,1) * PL * (PREY - Hunter),$$

де *PREY* – поточна позиція здобичі,

Hunter – мисливець у новому положенні, який атакує здобич,

а *PL* – відсоток покращення фізичної форми мисливця.

Наступні формули пропонуються для імітації оточення здобичі згаданими групами мисливців. Нові посади мисливців, які належать як лівому, так і правому крилу, генеруються наступним чином:

$$Hunter' = \begin{cases} rand((2 \times PREY - Hunter), PREY), & (2 \times PREY - Hunter) < PREY \\ rand(PREY, (2 \times PREY - Hunter)), & (2 \times PREY - Hunter) > PREY \end{cases},$$

де *PREY* – поточна позиція здобичі,

Hunter – поточна позиція мисливця,

а *Hunter'* – нова позиція мисливця.

Також нові посади мисливців Центру формуються таким чином:

$$Hunter' = \begin{cases} rand(Hunter, PREY), & Hunter < PREY \\ rand(PREY, Hunter), & Hunter > PREY \end{cases},$$

У вищенаведених рівняннях *rand(a, b)* генерує випадкове число між *a* та *b*, де *a* та *b* є верхньою та нижньою межами відповідно. Приклад оточування в LOA центральним левом і криловим левом показаний на рисунку 4. Запропонований механізм полювання має деякі переваги для досягнення кращих рішень. Примітним є те, що ця стратегія забезпечує коло у формі кола навколо здобичі і дозволяє мисливцям наближатися до здобичі з різних сторін. По-друге, ця стратегія дає можливість виходу з локального оптимуму, оскільки деякі мисливці використовують протилежну позицію.

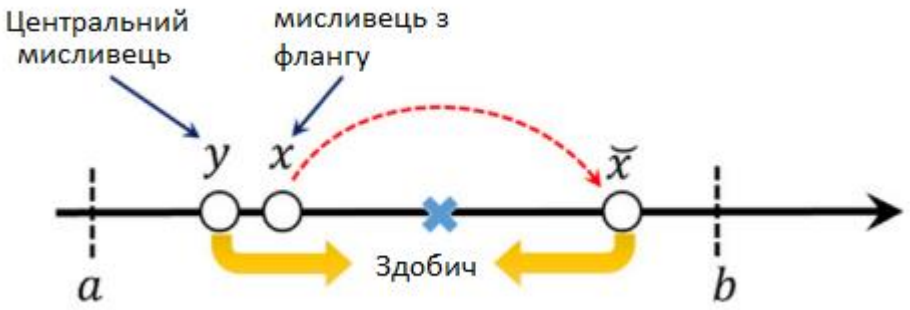

Рисунок 2.3 – Приклад обведення в LOA

2.4 Рух до безпечного місця

Як зазначалося в останньому підрозділі, в кожному прайді деякі самки виходять на полювання. Залишилися самки прямують до однієї з ділянок території. Оскільки територія кожного прайду складається з особистих найкращих позицій кожного члена та допомагає алгоритму Lion Optimization Algorithm (LOA) зберігати найкращі рішення, отримані на даний момент під час ітерації, його можна використовувати як цінну та надійну інформацію для покращення рішень. в LOA. Отже, нова посада для жінки-лева може бути дана так:

$$FemaleLion' = FemaleLion + 2D \times rand(0,1)\{R1\}$$
$$+U(-1,1) \times \tan(\theta) \times D \times \{R2\}$$
$$\{R1\}.\{R2\} = 0, \|\{R2\}\| = 1,$$

де *Female Lion* – це поточна позиція левиці,

D показує відстань між позицією левиці та обраною точкою, обраною в турнірі, серед території прайду.

R1 – вектор, початковою точкою якого є попереднє розташування самки лева, а напрямок – у вибрану позицію.

{R2} перпендикулярно до *{R1}.*

Тепер ми опишемо стратегію нашої турнірної секції. По-перше, ми визначаємо успіх лева, якщо він покращує його або її найкращу позицію на останній ітерації LOA. У групі P успіх лева і на ітерації t визначається як:

$$S(i,t,P) = \begin{cases} 1, & Best_{i,P}^{t} < Best_{i,P}^{t-1} \\ 0, & Best_{t,P}^{t} = Best_{i,P}^{t-1} \end{cases},$$

де $Best_{i,P}^{t}$ – найкраща позиція, знайдена левом *i* до ітерації *t*.

Велика кількість успіхів свідчить про те, що леви зійшлися до точки, яка далека від оптимальної точки. Аналогічно, низька кількість успіхів показує, що леви розмахуються навколо оптимального рішення без істотного покращення. Тому цей фактор можна використовувати як корисний елемент для розміру турніру. Використовуючи значення успіху, $K_j(s)$ обчислюється як:

$$K_j(s) = \sum_{i=1}^{n} S(i,t,P), \;\; j = 1,2,...,P,$$

де *n* – кількість левів у прайді,

а $K_j(s)$ – кількість левів у прайді *j*, які покращили свою придатність на останній ітерації.

Таким чином, розмір турніру в кожному прайді є адаптивним у кожній ітерації. Це означає, що коли значення успіху зменшується, розмір турніру збільшується, і це призводить до збільшення різноманітності. Тому розмір турніру розраховується так:

$$T_j^{Size} = \max\left(2, ceil\left(\frac{K_j(s)}{2}\right)\right), \;\; j = 1,2,...,P,$$

Приклад цієї стратегії наведено на рис. 2.4.

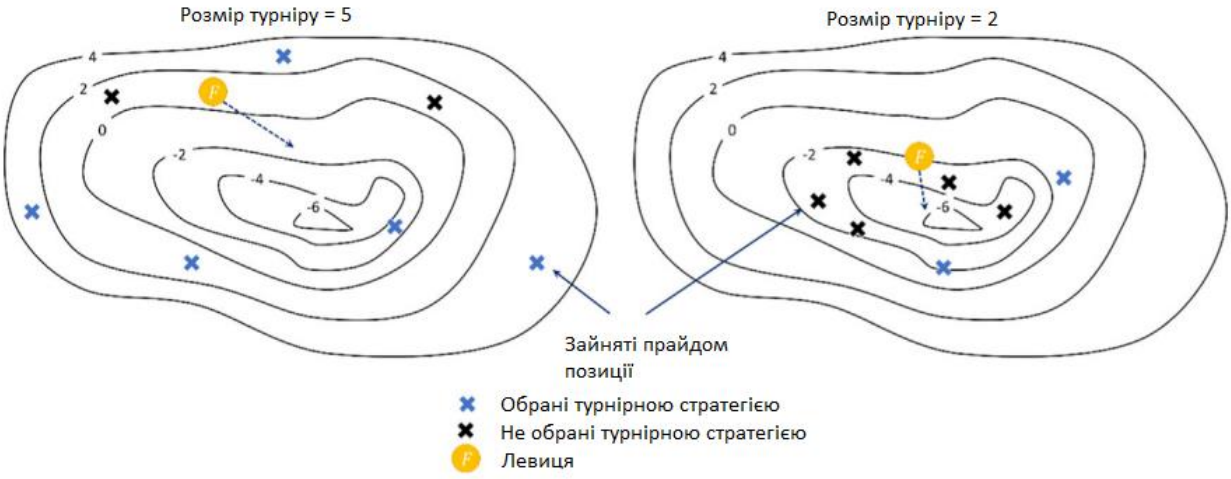

Рисунок 2.4 – Схематичний вигляд різних розмірів турніру.

2.5 Роумінг

Кожен лев-самець у прайді бродить по території цього прайду з певних причин. Щоб наслідувати цю поведінку постійних самців, % R території прайду вибирається випадковим чином і їх відвідує цей лев. У роумінгу, якщо чоловік-резидент відвідує нову позицію, яка є кращою за поточну найкращу позицію, оновіть його найкраще відвідуване рішення. Цей роумінг є потужним локальним пошуком і допомагає алгоритму Lion Optimization Algorithm (LOA) шукати рішення для його покращення. Цей прогрес показаний на рис. 2.5. Як показано на рис. 6, лев рухається в напрямку вибраної території на x одиниць, де x – випадкове число з рівномірним розподілом.

$$x \sim U(0, 2 \times d),$$

де d показує відстань між позицією самця лева і виділеною площею території.

Вектор від положення самця лева до виділеної ділянки території показує вихідний напрямок руху. Щоб забезпечити можливість пошуку ширшої області навколо поточного розчину та додавання властивості інтенсифікації до методу, а також для пошуку більш широкої області навколо поточного розчину, до цього напрямку додається кут θ. Здається, що кут, вибраний рівномірним розподілом між π/6 (рад) і π/6 (рад), є достатнім для цієї мети.

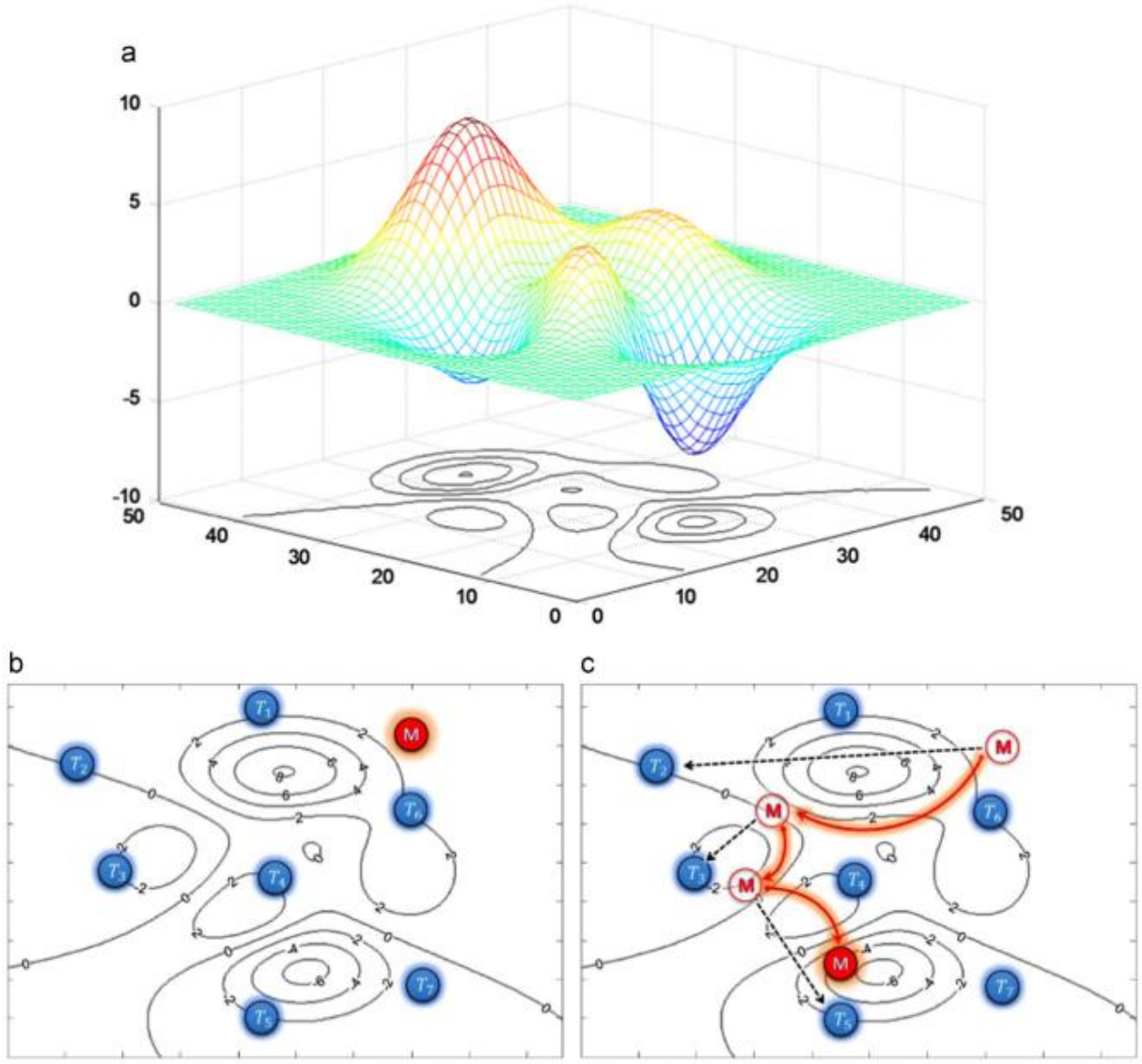

Рисунок 2.5 – Приклад роумінгу чоловіків-резидентів.

Кочівники та їх адаптивний роумінг допомагають запропонованому алгоритму випадковим чином шукати простір рішень і уникати потрапляння

в пастку локального оптимуму. У наведеній вище процедурі нова позиція левів-кочівників генерується наступним чином:

$$Lion_{ij}' = \begin{cases} Lion_{ij}, & if\ rand_j > pr_i \\ RAND_j, & otherwise \end{cases},$$

де $Lion_{ij}$ – поточна позиція *i*-го кочового лева,

j – вимір,

$rand_j$ – однорідне випадкове число в межах *[0, 1],*

RAND – випадково згенерований вектор у просторі пошуку,

а pr_i – ймовірність, яка обчислюється для кожного кочового лева незалежно як наступне:

$$pr_i = 0{,}1 + \min\left(0{,}5, \frac{(Nomad_i - Best_{nomad})}{Best_{nomad}}\right),\ i = 1,2,...,3,$$

де $Nomad_i$ і $Best_{nomad}$ – вартість поточної позиції і-го лева в кочівниках і найкраща вартість кочового лева відповідно.

2.6 Спаровування

Парування є важливим процесом, який забезпечує виживання левів, а також надає можливість для обміну інформацією між членами. У кожному прайді *%Ma* жінок-левів спарюються з одним або кількома мешканцями самців. Ці самці відбираються випадковим чином з того самого прайду, що і самка, щоб отримати потомство. Для левів-кочівників все відрізняється тим, що самка-кочівник спарюється тільки з одним із самців, які вибираються

випадковим чином. Оператор спарювання являє собою лінійну комбінацію батьків для отримання двох нових нащадків. Згідно з наведеними нижче рівняннями, нові дитинчата виробляються після відбору самки лева і самця(ів) для спарювання:

$$Offspring_j 1 = \beta \times FemaleLion_j + \sum \frac{(1-\beta)}{\sum_{i=1}^{NR} S_i} \times MaleLion_j^i \times S_i$$

$$Offspring_j 2 = (1-\beta) \times FemaleLion_j + \sum \frac{\beta}{\sum_{i=1}^{NR} S_i} \times MaleLion_j^i \times S_i$$

де j – розмір,

S_i дорівнює 1, якщо самець і вибрано для спарювання, інакше дорівнює 0,

NR – кількість постійних самців у прайді,

β – випадково згенероване число із нормальним розподілом із середнім значенням 0,5 і стандартним відхиленням 0 : 1.

Один із двох нових нащадків випадковим чином вибирається як самець, а інший як самка. Мутація застосовується до кожного гена одного з отриманих нащадків з ймовірністю (*%Mu*). Випадкове число замінює значення гена. Шляхом спарювання LOA обмінюється інформацією між гендерами, а нові дитинчата успадковують характер від обох статей.

2.7 Оборона

У прайді самці левів, коли стають дорослими, стають агресивними і борються з іншими самцями у своїй прайді. Побиті самці кидають свою гордість і стають кочівниками. З іншого боку, якщо лев-кочівник достатньо сильний, щоб спробувати захопити прайд, воюючи з його самцями, побитий

мешканець-лев виганяється з прайду і стає кочівником. Оператор захисту в LOA розділений на два основних етапи:

1. Захист від нових зрілих самців.
2. Захист від самців-кочівників.

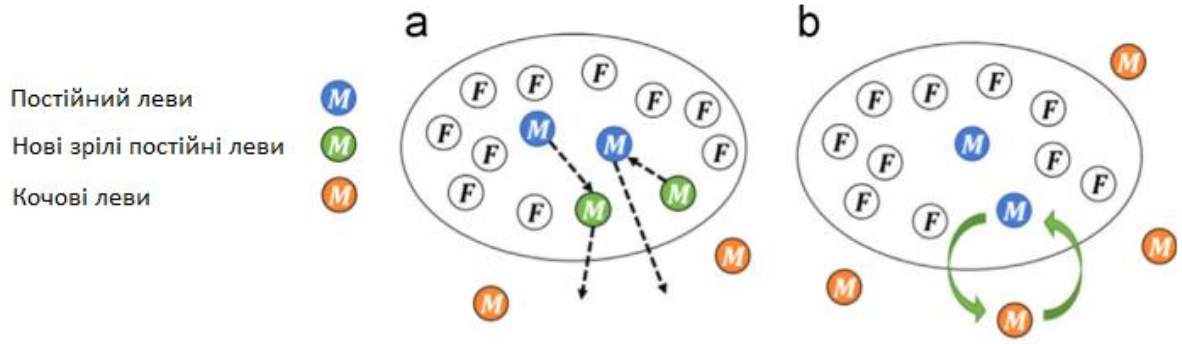

Рисунок 2.6 – Схема захисту в LOA:
а) захист від нових зрілих самців-резидентів;
б) захист від самців-кочівників

Захист від нових зрілих самців-резидентів і Захист від самців-кочівників зображено на рисунку 2.6. Оператор захисту допомагає алгоритму оптимізації левів (LOA) зберегти потужних самців левів як рішення, які відіграють важливу роль у LOA.

2.8 Міграція

Натхнений тим, що лев міняється життям і міграційною поведінкою в природі, коли один лев подорожує з одного прайду на інший або змінює свій спосіб життя, а мешканка самки стає кочівником і навпаки, він підсилює різноманітність цільового прайду завдяки його позиції в попередньому прайді. З іншого боку, міграція лева і зміна способу життя будує міст для обміну інформацією.

У кожному прайді максимальна кількість самок визначається S% населення прайду. Для оператора міграції деякі жінки вибираються

випадковим чином і стають кочівниками. Розмір мігрованих самок у кожному прайді дорівнює надлишку самок у кожному прайді плюс %I від максимальної кількості самок у прайді. Коли обрані самки мігрують з прайдів і стають кочівниками, нові самки-кочівники і старі самки-кочівники сортуються відповідно до їхньої придатності. Потім випадковим чином обирають кращих самок і розподіляють по прайдам, щоб заповнити порожнє місце мігруючих самок. Ця процедура підтримує різноманітність всього населення та обмін інформацією між прайдами.

2.9 Популяційна рівновага левів

Оскільки в популяції левів завжди існує рівновага, наприкінці кожної ітерації кількість живих левів буде контролюватися. Повага до максимально дозволеної кількості кожної статі у кочівників; левів-кочівників з найменшою оцінкою придатності буде видалено. На рисунку 2.7 зображено приклад оператора міграції та популяційної рівноваги Лева.

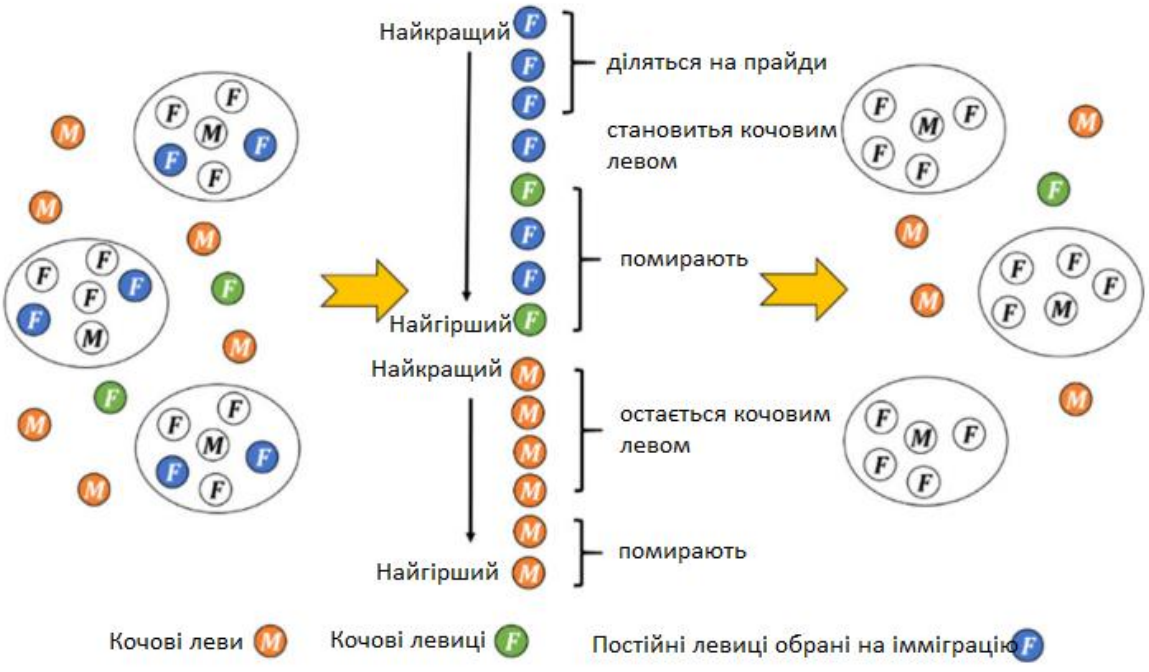

Рисунок 2.7 – Міграція та рівновага популяції левів

2.10 Конвергенція

Для умови зупинки, як зазвичай розглядається в алгоритмах оптимізації, найкращий результат розраховується, коли умова зупинки може бути припущена як час ЦП, максимальна кількість ітерацій, кількість ітерацій без покращення тощо.

Щоб побачити, як алгоритм Lion Optimization Algorithm (LOA) здатний вирішувати проблеми оптимізації, можна звернути увагу на деякі моменти:

– У LOA кожне рішення має певну стать, і кожна стать має свою власну стратегію пошуку. Це допомагає алгоритму Lion Optimization Algorithm (LOA) шукати оптимальну точку за допомогою різних стратегій.

– Загальна мета використання кількох прайдів полягає в тому, що кожен прайд зосереджується на певному регіоні та балансує між розвідкою та експлуатацією. Його характер LOA збільшує здатність його підходити для оптимізації мультимодальних задач.

– Кочівники та їх адаптивний роумінг допомагають алгоритму Lion Optimization Algorithm (LOA) випадковим чином шукати простір рішень і виходити з локального оптимуму.

– Особисті найкращі наразі позиції левів можуть дати цінну та достовірну інформацію, знайдену на даний момент населенням. Запропонована територія прайду допомагає алгоритму Lion Optimization Algorithm (LOA) зберігати найкращі рішення, отримані на даний момент під час ітерації.

– У LOA, спарюючись, леви діляться інформацією між статями, а нові дитинчата успадковують характер від обох статей.

– Самці-резиденти, які перебувають у роумінгу, допомагають алгоритму оптимізації Лева (LOA) використовувати інформацію від сусідів своїх прайдів, які володіють цінними знаннями. Цю процедуру можна розглядати як потужний локальний пошук.

– Запропонований огороджувальний механізм під час полювання має дві переваги; по-перше, забезпечує кругоподібне сусідство навколо рішень і дає можливість мисливцю наблизитися до здобичі з різних сторін, а по-друге, дає можливість рішенням вийти з локального оптимуму.

3 КОМП'ЮТЕРНА МОДЕЛЬ ОПТИМІЗАЦІЇ ЛЕВА НА ОСНОВІ АЛГОРИТМУ КЛАСТЕРИЗАЦІЇ ЗМІШАНИХ ДАНИХ

3.1 Архітектура застосунку

Для реалізації застосунку було обрана мова програмування Python та бібліотека NumPy для роботи з даними.

3.1.1 Python

Python – це багатопарадигмальна мова програмування. Об'єктно-орієнтоване програмування та структурне програмування повністю підтримуються, і багато його функцій підтримують функціональне програмування та аспектно-орієнтоване програмування (включаючи метапрограмування та метаоб'єкти (магічні методи)). Багато інших парадигм підтримуються за допомогою розширень, включаючи проектування за контрактом і логічне програмування.

Python використовує динамічний тип типізації та комбінацію підрахунку посилань і збирача сміття, що визначає цикл, для управління пам'яттю. Він також має динамічне розділення імен (пізнє прив'язування), яке зв'язує імена методів та змінних під час виконання програми.

Дизайн Python пропонує певну підтримку функціонального програмування в традиції Lisp. Він має функції filter,mapanreduce; списки розуміння, словники, набори та вирази генератора. Стандартна бібліотека має два модулі (itertools і functools), які реалізують функціональні інструменти, запозичені з Haskell і Standard ML.

Основна філософія мови викладена в документі The Zen of Python (PEP 20), який включає такі афоризми, як:

- красиве краще, ніж потворне;
- явне краще, ніж неявне;
- просте краще, ніж складне;
- складне краще, ніж заплутано;
- читабельність має значення.

Замість того, щоб усі його функціональні можливості були вбудовані в ядро, Python був розроблений так, щоб бути дуже розширюваним (з модулями). Ця компактна модульність зробила його особливо популярним як засіб додавання програмованих інтерфейсів до існуючих програм. Бачення Ван Россума невеликої основної мови з великою стандартною бібліотекою і легко розширюваним інтерпретатором виникло з його розчарування в ABC, яка підтримувала протилежний підхід. Його часто описують як мову з «батарейками» через його повну стандартну бібліотеку.

Python прагне до простішого, менш захаращенного синтаксису та граматики, надаючи розробникам можливість вибору методології кодування. На відміну від девізу Perl «є більше ніж один спосіб зробити це», Python підтримує філософію дизайну «повинен бути один – і бажано лише один – очевидний спосіб зробити це». Алекс Мартеллі, співробітник Python Software Foundation і автор книги Python, пише, що «описати щось «розумним» не вважається компліментом у культурі Python».

Розробники Python намагаються уникнути передчасної оптимізації та відкидають виправлення до некритичних частин довідкової реалізації CPython, які запропонували б незначне збільшення швидкості за ціною чіткості. Коли швидкість важлива, програміст Python може перемістити важливі за часом функції до модулів розширення, написаних на таких мовах, як C, або використовувати PyPy, компілятор «точно вчасно». Також доступний Cython, який перекладає скрипт Python на C і здійснює прямі виклики API рівня C в інтерпретатор Python.

Розробники Python прагнуть, щоб мова була веселою у використанні. Це відображено в його назві – данину британській комедійній групі Monty Python – і в іноді грайливих підходах до навчальних посібників і довідкових матеріалів, таких як приклади, які посилаються на spam і eggs (посилання на етюд Монті Пайтона). стандартних foo і bar.

Python може служити мовою сценаріїв для веб-додатків, наприклад, через mod_wsgi для веб-сервера Apache. Завдяки інтерфейсу шлюзу веб-сервера стандартний API розвинувся для полегшення цих додатків. Веб-фреймворки, такі як Django, Pylons, Pyramid, TurboGears, web2py, Tornado, Flask, Bottle і Zope, підтримують розробників у розробці та обслуговуванні складних програм. Pyjs і IronPython можна використовувати для розробки програм на базі Ajax на стороні клієнта. SQLAlchemy можна використовувати як картограф даних у реляційну базу даних. Twisted – це фреймворк для програмування зв’язку між комп’ютерами, який використовується (наприклад) Dropbox.

Бібліотеки, такі як NumPy, SciPy і Matplotlib, дозволяють ефективно використовувати Python у наукових обчисленнях зі спеціалізованими бібліотеками, такими як Biopython і Astropy, які забезпечують функціональні можливості для певної області. SageMath – це система комп’ютерної алгебри з інтерфейсом ноутбука, програмованим на Python: його бібліотека охоплює багато аспектів математики, включаючи алгебру, комбінаторику, числову математику, теорію чисел і обчислення. OpenCV має прив’язки Python з багатим набором функцій для комп’ютерного зору та обробки зображень.

Python зазвичай використовується в проектах зі штучним інтелектом і в проектах машинного навчання за допомогою таких бібліотек, як TensorFlow, Keras, Pytorch і Scikit-learn. Як мова сценаріїв із модульною архітектурою, простим синтаксисом та інструментами обробки розширеного тексту, Python часто використовується для обробки природною мовою.

3.1.2 NumPy

NumPy — це фундаментальний пакет для наукових обчислень на Python. Це бібліотека Python, яка надає багатовимірний об’єкт масиву, різні похідні об’єкти (такі як замасковані масиви та матриці), а також набір підпрограм для швидких операцій з масивами, включаючи математичні, логічні, маніпуляції з формою, сортування, вибір, введення/виводу. , дискретні перетворення Фур'є, базова лінійна алгебра, основні статистичні операції, випадкове моделювання та багато іншого.

В основі пакету NumPy лежить об'єкт ndarray. Це інкапсулює n-вимірні масиви однорідних типів даних, при цьому багато операцій виконуються в скомпільованому коді для підвищення продуктивності. Існує кілька важливих відмінностей між масивами NumPy і стандартними послідовностями Python:

– Масиви NumPy мають фіксований розмір під час створення, на відміну від списків Python (які можуть динамічно зростати). Зміна розміру ndarray створить новий масив і видалить оригінал;

– Усі елементи в масиві NumPy повинні мати один тип даних і, таким чином, мати однаковий розмір пам’яті. Виняток: можна мати масиви об’єктів (Python, включаючи NumPy), що дозволяє використовувати масиви елементів різного розміру;

– Масиви NumPy полегшують розширені математичні та інші типи операцій над великою кількістю даних. Як правило, такі операції виконуються ефективніше і з меншою кількістю коду, ніж це можливо з використанням вбудованих послідовностей Python;

– Зростаюча кількість науково-математичних пакетів на основі Python використовує масиви NumPy; хоча вони зазвичай підтримують введення послідовності Python, вони перетворюють такі вхідні дані в масиви NumPy перед обробкою, і вони часто виводять масиви NumPy. Іншими словами, для

того, щоб ефективно використовувати більшу частину (можливо, навіть більшість) сучасного науково-математичного програмного забезпечення на основі Python, просто знати, як використовувати вбудовані типи послідовностей Python, недостатньо – потрібно також знати, як використовувати масиви NumPy.

У наукових обчисленнях особливо важливі моменти про розмір і швидкість послідовності. Як простий приклад розглянемо випадок множення кожного елемента в одновимірній послідовності на відповідний елемент в іншій послідовності такої ж довжини. Якщо дані зберігаються в двох списках Python, a і b, ми можемо повторити кожен елемент як наведено на рисунку 3.1:

```
c = []
for i in range(len(a)):
    c.append(a[i]*b[i])
```

Рисунок 3.1 – Ілюстрація множення елементів в одновимірній послідовності на Python

Це дає правильну відповідь, але якщо a і b кожне містять мільйони чисел, ми заплатимо ціну за неефективність циклу в Python. Ми могли б виконати те саме завдання набагато швидше на C, написавши алгоритм наведений на рисунку 3.2 (для ясності ми нехтуємо оголошеннями змінних та ініціалізаціями, виділенням пам’яті тощо).

```
for (i = 0; i < rows; i++): {
  c[i] = a[i]*b[i];
}
```

Рисунок 3.2 – Ілюстрація множення елементів в одновимірній послідовності на C

Це заощаджує всі накладні витрати, пов’язані з інтерпретацією коду Python та маніпулюванням об’єктами Python, але за рахунок переваг,

отриманих від кодування на Python. Крім того, необхідна робота з кодування збільшується зі збільшенням розмірності наших даних. У випадку 2-D масиву, наприклад, код C (скорочений, як і раніше) розширюється до коду на веденому на рисунку 3.3.

```
for (i = 0; i < rows; i++): {
  for (j = 0; j < columns; j++): {
    c[i][j] = a[i][j]*b[i][j];
  }
}
```

Рисунок 3.3 – Множення елементів в двовимірній послідовності на C

NumPy дає нам найкраще з обох світів: поелементні операції є «режимом за замовчуванням», коли задіяний ndarray, але поелементна операція швидко виконується попередньо скомпільованим кодом C. У NumPy код наведений на рисунку 3.4 виконує те саме, що і попередні приклади, зі швидкістю, близькою до C, але з простотою коду, яку ми очікуємо від чогось на основі Python. Дійсно, ідіома NumPy ще простіше! Цей останній приклад ілюструє дві функції NumPy, які є основою більшої частини його потужності: векторизація та мовлення.

```
c = a * b
```

Рисунок 3.4 – Множення елементів в послідовності за допомогою NumPy

Векторизація описує відсутність будь-якого явного циклу, індексування тощо в коді – ці речі відбуваються, звичайно, просто «за кадром» в оптимізованому, попередньо скомпільованому коді C. Векторизований код має багато переваг, серед яких:

– векторизований код є більш стислим і легшим для читання;
– менше рядків коду зазвичай означає менше помилок;

– код більше нагадує стандартні математичні позначення (що полегшує, як правило, правильне кодування математичних конструкцій);

– векторізація призводить до більш «Pythonic» коду. Без векторизації наш код був би завалений неефективними і важкочитаними циклами for.

Трансляція – це термін, який використовується для опису неявної поелементної поведінки операцій; Взагалі кажучи, у NumPy всі операції, не тільки арифметичні, а й логічні, порозрядні, функціональні тощо, поводяться таким чином неявно поелементно, тобто вони транслюються. Більше того, у наведеному вище прикладі a і b можуть бути багатовимірними масивами однакової форми, або скаляром і масивом, або навіть двома масивами з різними формами, за умови, що менший масив «розширюється» до форми більшого таким чином, щоб отримана трансляція була однозначною.

3.2 Програмна реалізація

Для реалізації алгоритму була обрана середа розробки PyCharm Professional 2021.2.3.

Величезна колекція інструментів PyCharm із коробки включає в себе вбудований налагоджувач і тестовий запуск; Профайлер Python; вбудований термінал; інтеграція з основними VCS та вбудованими інструментами баз даних; можливості віддаленої розробки з віддаленими перекладачами; інтегрований термінал ssh; і інтеграція з Docker і Vagrant.

Він має потужний налагоджувач із графічним інтерфейсом користувача для Python та JavaScript. Дозволяє створюйте та запускайте свої тести за допомогою допомоги в кодуванні та за допомогою програми тестування на основі графічного інтерфейсу.

PyCharm дозволяю економити час за допомогою уніфікованого інтерфейсу користувача для роботи з Git, SVN, Mercurial або іншими системами контролю версій да налаштовувати автоматичне розгортання на

віддаленому хості або віртуальній машині та керування своєю інфраструктурою за допомогою Vagrant і Docker.

PyCharm інтегрується з IPython Notebook, має інтерактивну консоль Python і підтримує Anaconda, а також кілька наукових пакетів, включаючи Matplotlib і NumPy.

Середа розробки має вбудовану підтримку наукових бібліотек. Він підтримує Pandas, Numpy, Matplotlib та інші наукові бібліотеки, пропонуючи найкращий у своєму класі аналіз коду, графіки, засоби перегляду масивів та багато іншого.

PyCharm також кроссплатформенна IDE. PyCharm працює на Windows, macOS або Linux. Ви можете встановити та запустити PyCharm на будь-якій кількості машин, а також використовувати те саме середовище та функціональні можливості на всіх своїх машинах.

Для створення нового проекту необхідно перейти до File -> New Project. У новому вікні необхідно вказати директорію проекту та вказати версію Python як показано на рисунку 3.5. У нашому додатку будемо використовувати Python 3.9.

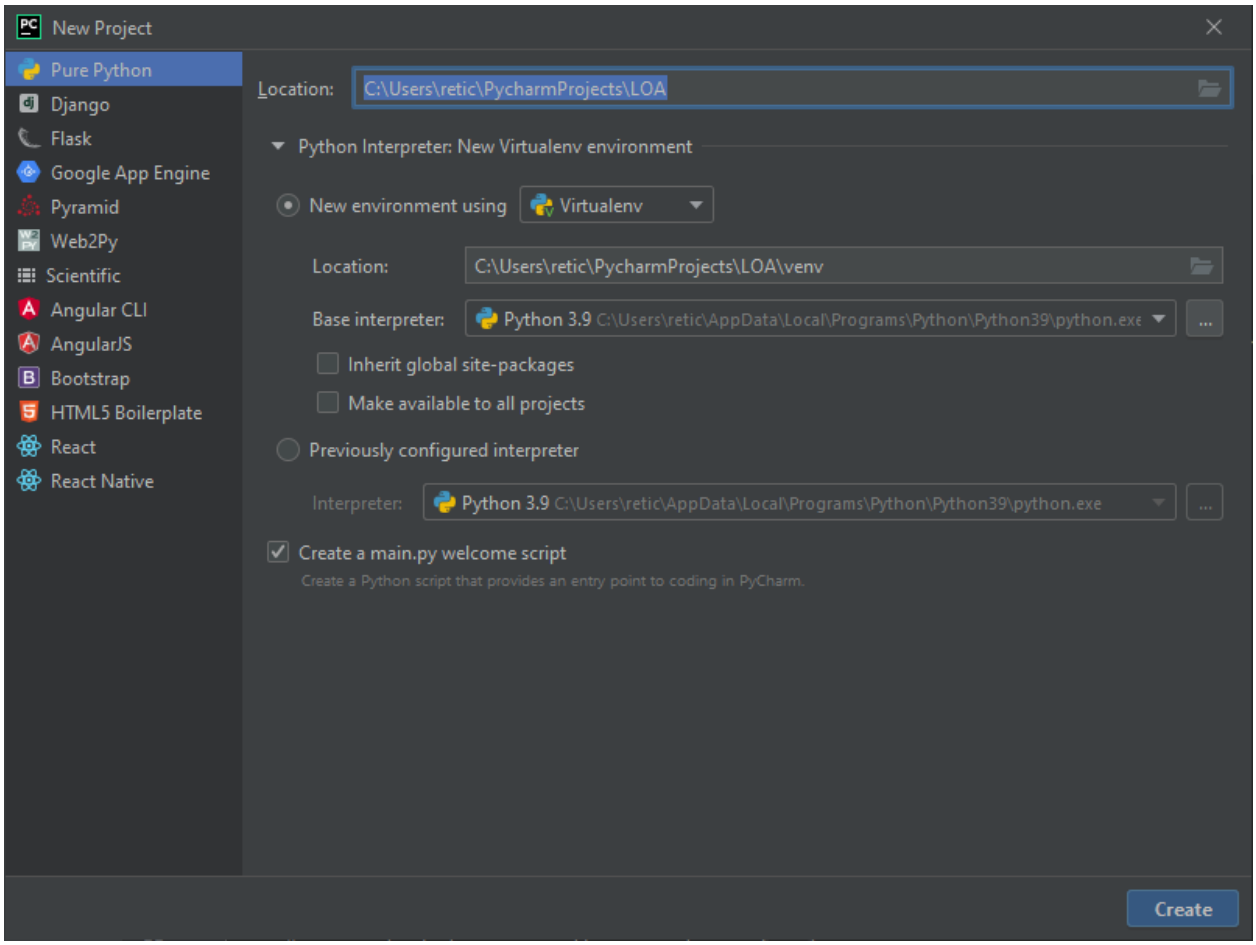

Рисунок 3.5 – Вікно створення нового Python проекту

У вікні Settings на вкладці Python interpreter інсталюємо усі необхідні бібліотеки для реалізації LOA (рис. 3.6).

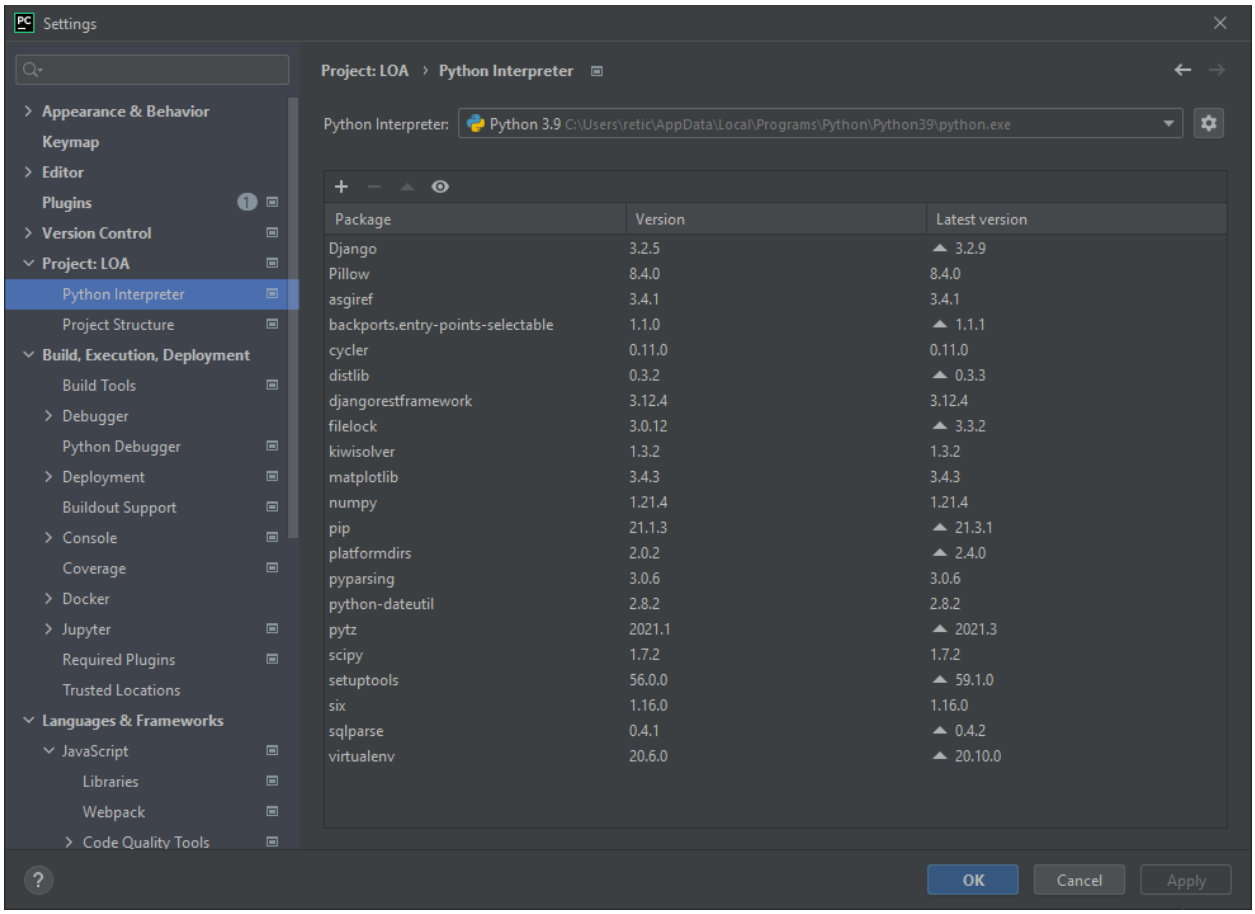

Рисунок 3.6 – Усі встановлені бібліотеки для реалізації алгоритму

Далі згідно з кроками алгоритму описаному у другому розділі реалізуємо алгоритм за допомогою мови Python. У нашому випадку програма являє собою бібліотеку, яку можна підключити використовувати у будь якому Python проекті. У результаті такого підходу користувач бібліотеки може використати реалізований алгоритм у будь якому десктопному чи веб застосунку. Приклад реалізації одного із кроків LOA наведений на рисунку 3.7.

```python
# nomad lions moving randomly in the search space
# under step 4
def nomads_roam(nomad_lions_array, lower_limit, upper_limit, dim):
    for lion in nomad_lions_array:

        # best position evaluation of all nomad lions
        best_position_score = np.min([l.getCurrentPositionScore() for l in nomad_lions_array])

        # the max threshold number in order for the lion to roam
        threshold_roaming_probability = 0.1 + np.min(
            [0.5, (lion.getCurrentPositionScore()[0] - best_position_score) / best_position_score])

        # move lion if not greater than the threshold
        if not (random.random() > threshold_roaming_probability):
            lion.x = np.random.uniform(lower_limit, upper_limit, (1, dim))

        # update the best visited position if better than current
        if lion.getBestVisitedPositionScore() > lion.getCurrentPositionScore():
            lion.bestVisitedPosition = lion.x

    return nomad_lions_array
```

Рисунок 3.7 – Приклад реалізації кроку nomads roam

3.3 Тестування розробленої системи

Для тестування алгоритму був розроблена консольна програма яка 60 разів виконує оптимізацію за допомогою LOA та друкує найкраще значення результату роботи алгоритму оптимізації (рис. 3.8).

```
Finished% 0.0
improved score: 1.84E+09
improved score: 1.80E+09
improved score: 8.72E+08
improved score: 8.64E+08
improved score: 7.18E+08
improved score: 5.81E+08
Finished% 0.03333333333333333
Finished% 0.06666666666666667
Finished% 0.1
Finished% 0.13333333333333333
improved score: 4.93E+08
improved score: 4.54E+08
Finished% 0.16666666666666666
```

Рисунок 3.8 – Результат роботи алгоритму

3.4 Аналіз результатів роботи

Аналіз експерименту виконується з наборами контрольних даних щодо гепатиту, післяопераційного пацієнта, австралійського кредитного схвалення, німецьких кредитних даних і статистичного журналу Heart, доступних у сховищі машинного навчання UCI. Детальна інформація про набори даних наведена в наступній таблиці 3.1.

Ефективність алгоритму кластеризації K-Prototype на основі K-Means, K-Prototype та Lion Optimized Algorithm вимірюється за допомогою чотирьох зовнішніх мір валідності, а саме F-Measure, Rand Index, Jaccard Index та Entropy. Заходи зовнішньої валідності перевіряють якість кластерів шляхом порівняння результатів кластеризації. Усі ці чотири показники мають значення від 0 до 1. У випадку індексів Ренда, індексу Жакара та F-міри значення 1 вказує, що кластери даних абсолютно однакові, тому збільшення значень цих показників свідчить про кращу продуктивність.

Таблиця 3.1 – Деталі наборів даних

№	Набір даних	К-ть екземплярів	К-ть атрибутів	К-ть класів
1	Australian Credit Approval	690	14	2
2	German Credit Approval	1000	20	2
3	Hepatitis	155	19	2
4	Post Operative Patient	90	8	3
5	Stat Log Heart	270	13	2

Відповідно до Rand Index, продуктивність алгоритму кластеризації K-Prototype на основі алгоритму Lion Optimization Algorithm дає послідовні та

покращені результати, ніж K-Means та K-Prototype Algorithm майже у всіх наборах даних.

З таблиці 3.2 та рисунку 3.9 видно, що алгоритм кластеризації K-Prototype на основі алгоритму Lion Optimization Algorithm дає послідовні та кращі результати для статистичних журналів серця, гепатиту, німецьких кредитних даних, ніж післяопераційних пацієнтів та наборів даних австралійського схвалення кредиту.

Таблиця 3.2 – Порівняльний аналіз на основі індексу Ренда

№	Набір даних	K-Means	K-Prototype	Lion Optimization based K-Prototype Clustering Algorithm
1	Australian Credit Approval	0,56	0,58	0,62
2	German Credit Approval	0,53	0,55	0,57
3	Hepatitis	0,72	0,73	0,74
4	Post Operative Patient	0,41	0,45	0,47
5	Stat Log Heart	0,66	0,68	0,72

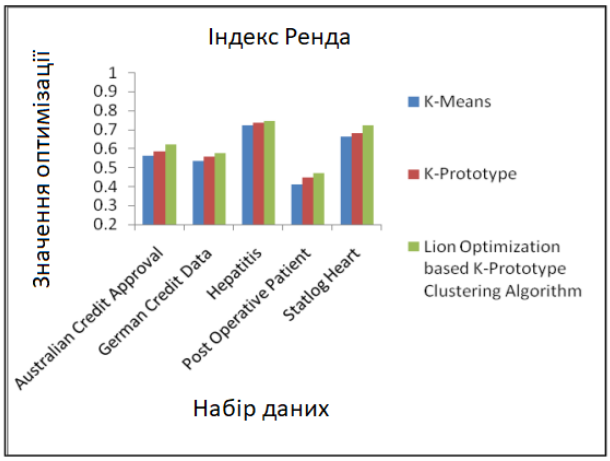

Рисунок 3.9 – Аналіз на основі індексу Ренда

З таблиці 3.3, заснованої на індексі Жакара, продуктивність алгоритму кластеризації K-Prototype на основі алгоритму Lion Optimized Algorithm дає послідовні та кращі результати для наборів даних. Алгоритм K-Means і K-Prototype майже у всіх наборах даних.

Таблиця 3.3 – Порівняльний аналіз на основі індексу Жакара

№	Набір даних	K-Means	K-Prototype	Lion Optimization based K-Prototype Clustering Algorithm
1	Australian Credit Approval	0,43	0,45	0,52
2	German Credit Approval	0,45	0,46	0,48
3	Hepatitis	0,62	0,63	0,65
4	Post Operative Patient	0,33	0,35	0,38
5	Stat Log Heart	0,54	0,56	0,70

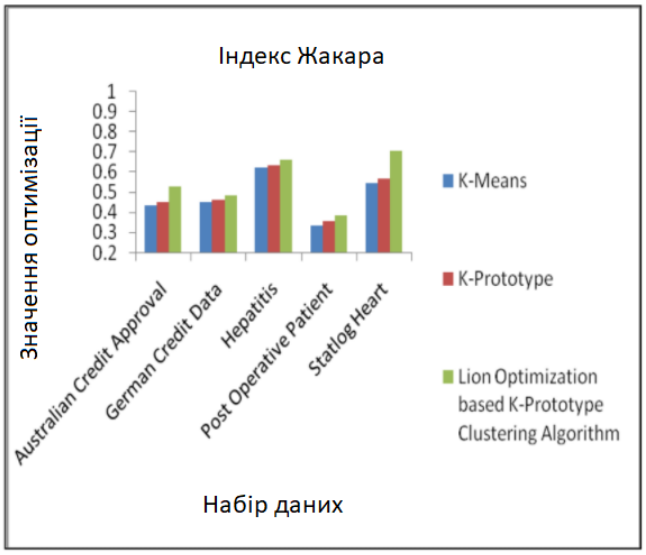

Рисунок 3.10 – Аналіз на основі індексу Жакара

У випадку F-Measure значення 1 вказує на те, що кластери даних абсолютно однакові, тому збільшення значень цих показників свідчить про кращу продуктивність. Виходячи з цього, результати алгоритму кластеризації K-прототипу на основі алгоритму Lion Optimization є помітнішим, ніж алгоритм K-Mean і K-Prototype для всіх наборів даних, представлених у таблиці 3.4.

Таблиця 3.4 – Порівняльний аналіз на основі F-Measure

№	Набір даних	K-Means	K-Prototype	Lion Optimization based K-Prototype Clustering Algorithm
1	Australian Credit Approval	0,76	0,83	0,86
2	German Credit Approval	0,76	0,82	0,84
3	Hepatitis	0,78	0,84	0,87

4	Post Operative Patient	0,79	0,85	0,86
5	Stat Log Heart	0,85	0,87	0,88

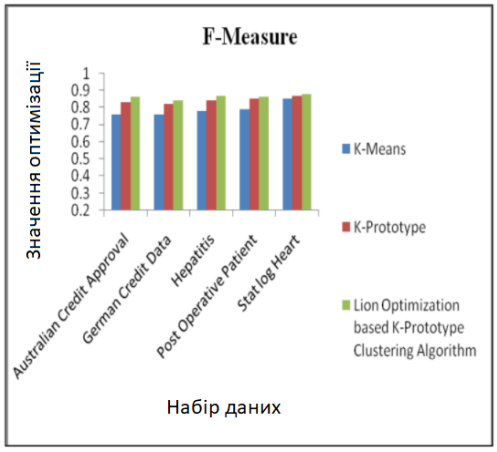

Рисунок 3.11 – Аналіз на основі F-Measure

Зменшення значень міри ентропії свідчить про кращу продуктивність. Виходячи з цього, продуктивність алгоритму кластеризації K-прототипу на основі алгоритму Lion Optimized Algorithm на основі ентропії є дуже значною, ніж K-Mean і K-Prototype для всіх наборів даних, представлених у таблиці 3.5.

Таблиця 3.5 – Порівняльний аналіз на основі ентропії

№	Набір даних	K-Means	K-Prototype	Lion Optimization based K-Prototype Clustering Algorithm
1	Australian Credit Approval	0,51	0,51	0,50
2	German Credit Approval	0,45	0,43	0,45
3	Hepatitis	0,52	0,52	0,52

4	Post Operative Patient	0,42	0,41	0,40
5	Stat Log Heart	0,45	0,44	0,43

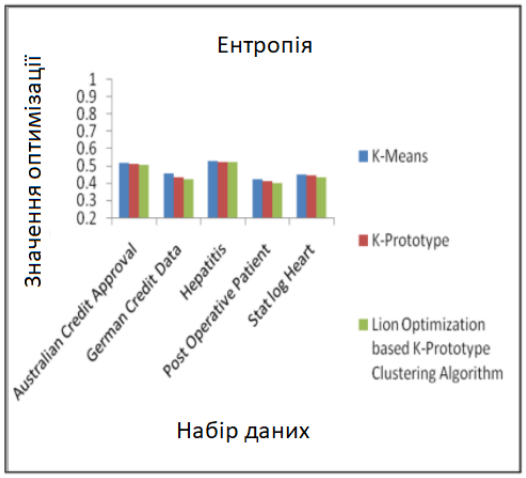

Рисунок 3.12 – Аналіз на основі ентропії

ВИСНОВКИ

За останні десятиліття були розроблені різні алгоритми метаевристичної оптимізації. Багато з цих алгоритмів натхненні природними явищами. У цьому дослідженні представлено новий алгоритм оптимізації, який називається Lion Optimization Algorithm (LOA). LOA сконструйовано на основі моделювання самотньої та спільної поведінки левів, таких як захоплення здобичі, спарювання, територіальне маркування, захист та інші види поведінки. Щоб оцінити продуктивність запровадженого алгоритму, ми протестували його на наборі різноманітних стандартних функцій тесту. Результати, отримані LOA, у більшості випадків забезпечують чудові результати у швидкій конвергенції та досягненні глобального оптимуму, і в усіх випадках можна порівняти з іншими метаевристичними.

Запропонований алгоритм було перевірено на п’яти контрольних наборах даних, які включають як числові, так і категоріальні атрибути. Доведено, що продуктивність запропонованого алгоритму перевершує продуктивність звичайних алгоритмів кластеризації K-Means та K-Prototype.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Sundar S., Singh A. (2012). A swarm intelligence approach to the early/tardy scheduling problem. Swarm Evolut. Comput, 4(0), pp. 25-32.

2. Goldansaz SM., Jolai F., Zahedi Anaraki AH. (2013). A hybrid imperialist competitive algorithm for minimizing makespan in a multi-processor open shop, Appl. Math. Model., 37(23), pp. 9603-9616.

3. Suresh K., Kumarappan N. (2013). Hybrid improved binary particle swarm optimization approach for generation maintenance scheduling problem. Swarm Evolut. Comput, 9(0), pp. 69-89.

4. Meysam Mousavi, S (2013). A new support vector model-based imperialist competitive algorithm for time estimation in new product development projects. Robot. Computer Integr. Manuf, pp. 157-68.

5. F de Lima Neto, et al (2008). A novel search algorithm based on fish school behavior, in: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, SMC.

6. Yang and S. Deb., (2009) Cuckoo Search via Lévy flights, in: Proceedings of the IEEE World Congress on Nature & Biologically Inspired Computing, NaBIC.

7. Rajabioun R. (2011). Cuckoo optimization algorithm. Appl. Soft Comput, 11(8), 5508-5518.

8. Yang X-S. (2010). A new metaheuristic bat-inspired algorithm. Nature Inspired Cooperative Strategies for Optimization, pp. 65–74.

9. Y. Shiqin, J. Jianjun, and Y. Guangxing (2009). A dolphin partner optimization. in: Proceedings of the IEEE WRI Global Congress on Intelligent Systems.

10. Hofmann J., Limmer S., Fey D. (2013). Performance investigations of genetic algorithms on graphics cards. Swarm Evolut. Comput., 12(0), pp. 33-47.

11. R. Tang, et al. (2012). Wolf search algorithm with ephemeral memory, in: Proceedings of the Seventh International Conference on IEEE Digital Information Management.

12. Changdar C., Mahapatra GS., Kumar Pal R. (2014). An efficient genetic algorithm for multi-objective solid travelling salesman problem under fuzziness. Swarm Evolut. Comput., 15(0), pp. 27-37.

13. Wolpert DH., Macready WG. (1997). No free lunch theorems for optimization. Evolut. Comput. IEEE Trans, 1(1), pp. 67-82.

14. Mccomb, K, et al (1993). Female lions can identify potentially infanticidal males from their roars. Proc. R. Soc. Lond. Ser B: Biol. Sci., 252, pp. (1333)59-64.

15. Schaller GB. (1972). The Serengeti lion: a study of predator–prey relations: Wildlife behavior and ecology series. Chicago, Illinois, USA: University of Chicago Press.

16. Scheel D, Packer C. (1991). Group hunting behaviour of lions: a search for cooperation. Anim. Behav., 41, pp. (4)697-709.

17. Rashedi E., Nezamabadi-Pour H., Saryazdi S. (2009). GSA: a gravitational search algorithm. Inf. Sci., 179(13), pp. 2232-48.

18. H.R., Tizhoosh (2005). Opposition-based learning: a new scheme for machine intelligence, in: Proceedings of the CIMCA/IAWTIC.

19. J., Liang, B. Qu, and P. Suganthan (2013). Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization: Computational Intelligence Laboratory.

20. Eusuff MM, Lansey KE (2003). Optimization of water distribution network design using the shuffled frog leaping algorithm: J. Water Res. Plan. Manag., 129(3), pp. 210-25.

21. Passino KM. (2002). Biomimicry of bacterial foraging for distributed optimization and control: Control Syst, IEEE, 22(3), pp. 52-67.

22. H.A., Abbass (2001). MBO: marriage in honey bees optimization – a haplometrosis polygynous swarming approach: Proceedings of the IEEE Congress on Evolutionary Computation.

23. Bhargava V., Fateen SEK, Bonilla-Petriciolet A. (2013). Cuckoo search: a new nature-inspired optimization method for phase equilibrium calculations: Fluid Phase Equilib, 337(0), pp. 191-200.

24. A. Shafronenko, Ye. Bodyanskiy, D. Rudenko (2019). Online neuro fuzzy clustering of data with omissions and outliers based on completion strategy, in: Proceedings of The Second International Workshop on Computer Modeling and Intelligent Systems (CMIS-2019), Zaporizhzhia, pp. 18-27.

25. Shafronenko A., Bodyanskiy Ye., Pliss I., Klymova I. (2021) Online Credibilistic Fuzzy Clustering Method Based on Cauchy Density Distribution Function, 2021 11th International Conference on Advanced Computer Information Technologies (ACIT), pp.704-707. DOI: 10.1109/ACIT52158.2021.9548572

26. Bodyanskiy, Y. V., Shafronenko, A. Y., & Klymova, I. N. (2021). ONLINE FUZZY CLUSTERING OF INCOMPLETE DATA USING CREDIBILISTIC APPROACH AND SIMILARITY MEASURE OF SPECIAL TYPE. Radio Electronics, Computer Science, Control, 1(1), 97-104.

27. Bodyanskiy Y., Shafronenko A., Klymova I., Polyvoda V. (2022) Robust Recurrent Credibilistic Modification of the Gustafson - Kessel Algorithm. In: Babichev S., Lytvynenko V. (eds) Lecture Notes in Computational Intelligence and Decision Making. ISDMCI 2021. Lecture Notes on Data Engineering and Communications Technologies, vol 77. Springer, Cham. DOI: https://doi.org/10.1007/978-3-030-82014-5_42.

28. Бодянський Є.В., Шафроненко А.Ю., Климова І.М. (2021) Метод адаптивної достовірної нечіткої кластеризації даних на основі еволюційного алгоритму. Збірник наукових праць Харківського національного університету Повітряних Сил, 2(68), pp. 80-83. DOI:https://doi.org/10.30748/zhups.2021.68.10.

29. Bodyanskiy Y., Shafronenko A., Klymova I. (2021) Online fuzzy clustering of incomplete data using credibilistic approach and similarity measure of special type, Radio Electronics, Computer Science, Control, pp. 97-104.

30. Y Bodyanskiy, A Shafronenko, I Klymova (2021) Рекурентна достовірна нечітка кластеризація великих даних з використанням функції належності спеціального типу, Біоніка інтелекту, 95(2), pp. 88-81.

31. Bodyanskiy, Y. V., & Shafronenko, A. Y. (2020). ONLINE CREDIBILISTIC FUZZY CLUSTERING OF DATA WITH GAPS. PROBLEMS AND PERSPECTIVES OF MODERN SCIENCE AND PRACTICE, 43.

32. Shafronenko, A., Dolotov, A., Bodyanskiy, Y., & Setlak, G. (2018, August). Fuzzy clustering of distorted observations based on optimal expansion using partial distances. In 2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP), pp. 327-330. IEEE.

33. Shafronenko, A. Y., & Rudenko, D. A. (2020). ONLINE RECURRENT METHOD OF CREDIBILISTIC FUZZY CLUSTERING. ВВК 91, 37.

34. Bodyanskiy, Y. V., Shafronenko, A., & Rudenko, D. (2019). Online Neuro Fuzzy Clustering of Data with Omissions and Outliers based on Completion Strategy. In CMIS, pp. 18-27.

35. Бодянський, Є. В., Шафроненко, А. Ю., & Климова, І. М. (2019). Онлайн достовірна нечітка кластерізація даних з використанням функції належності спеціального типу. Біоніка інтелекту, 2(93), pp. 3-6.

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

37. Shafronenko, A., Bodyanskiy, Y., Pliss, I., & Popov, S. (2020, September). Evolving neo-fuzzy system for distorted data online processing. In 2020 10th International Conference on Advanced Computer Information Technologies (ACIT), pp. 352-355. IEEE.

38. 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, pp. 86-88.

39. 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.

40. Ye, Bodyanskiy, Tyshchenko, O., & Shafronenko, A. (2019). Fuzzy clustering of incomplete data by means of similarity measures-Proc. 2019 IEEE 2nd Ukr. Conf. on Electrical and Computer Engineering (UKRCON), Track 6. Lviv, Ukraine, pp. 149-152.

41. Peters, Tim (19 August 2004). "PEP 20 – The Zen of Python". Python Enhancement Proposals. Python Software Foundation. Archived from the original on 26 December 2018. Retrieved 24 November 2008.