НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичного моделювання та аналізу даних

«На правах рукопису»
УДК 004.94:519.6/519.7

«До захисту допущено»
Завідувач кафедри
_____ Іван ТЕРЕЩЕНКО
«__» _____ 2025 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Математичні методи моделювання, розпізнавання образів та комп'ютерного зору»

зі спеціальності: 113 Прикладна математика
на тему: **«Порівняльний аналіз новітніх алгоритмів ройового інтелекту»**

Виконала:
студентка IV курсу, групи ФІ-12
Краснорyцька Марія Сергіївна _____

Керівник:
к.т.н., ст. досл. доцент кафедри ММАД
Хайдуров Владислав Володимирович _____

Рецензент:
доцент кафедри вищої та прикладної математики
НУБіП України, канд. фіз.-мат. наук, доцент
Цюпій Тамара Іванівна _____

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

Студент _____

Київ — 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичного моделювання та аналізу даних

Рівень вищої освіти — перший (бакалаврський)
Спеціальність — 113 Прикладна математика,
ОПП «Математичні методи моделювання, розпізнавання образів та комп'ютерного зору»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Іван ТЕРЕЩЕНКО
«__» _____ 2025 р.

ЗАВДАННЯ
на дипломну роботу

Студент: Красноруцька Марія Сергіївна

1. Тема роботи: «*Порівняльний аналіз новітніх алгоритмів ройового інтелекту*», науковий керівник дисертації: к.т.н., ст. досл. доцент кафедри ММАД Хайдуров Владислав Володимирович,
затверджені наказом по університету №__ від «__» _____ 2025 р.

2. Термін подання студентом роботи: «02» червня 2025 р.

3. Об'єкт дослідження: складні процеси, а також відповідні їм математичні моделі, за допомогою яких можна визначити оптимальний режим функціонування систем, де відбуваються відповідні процеси.

4. Предмет дослідження: сучасні оптимізатори, що засновані на ройових принципах, які дають можливість визначити найоптимальніші або субоптимальні розв'язки науково-технічних проблем сьогодення у галузі будівництва, енергетики, конструювання тощо.

5. Перелік завдань:

– Огляд сучасних практичних задач, які потребують застосування ройового інтелекту для більш ефективного вирішення.

– Огляд та аналіз публікацій досліджень на цю тему.

– Ретельний аналіз нових переспективних алгоритмів ройового інтелекту.

– Імплементація та тестування цих ройових алгоритмів на математичних моделях.

– Оцінка роботи реалізованих алгоритмів та порівняння їхньої ефективності між собою.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: *презентація доповіді.*

7. Орієнтовний перелік публікацій: *Красноруцька М.С., Хайдуров В.В. Сучасні методи ройового інтелекту для аналізу складних об'єктів і систем. Теоретичні і прикладні проблеми фізики, математики та інформатики: матеріали XXIII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених (14–17 травня 2025 р., м. Київ, Україна). – Київ : КПІ ім. Ігоря Сікорського, Видавництво «Політехніка», 2025. С. 340–343. ISBN 978-966-990-053-1.*

8. Дата видачі завдання: 10 вересня 2024 р.

Календарний план

№ з/п	Назва етапів виконання бакалаврської дисертації	Термін виконання	Примітка
1	Узгодження теми роботи із науковим керівником	02-10 вересня 2024 р.	Виконано
2	Огляд опублікованих джерел за тематикою дослідження	Жовтень-листопад 2024 р.	Виконано
3	Розробка ходу проведення досліджень	Грудень 2024 р.	Виконано
4	Написання програмного забезпечення та проведення досліджень	Січень-квітень 2025 р.	Виконано
5	Аналіз отриманих результатів	Квітень-травень 2025 р.	Виконано
6	Оформлення дипломної роботи	Травень 2025 р.	Виконано

Студент _____ Марія Красноруцька

Керівник _____ Владислав Хайдуров

РЕФЕРАТ

Кваліфікаційна робота містить: 113 стор., 32 рисунки, 31 таблиць, 33 джерел.

Дана дипломна робота присвячена проведенню порівняльного аналізу трьох сучасних ройових алгоритмів – CHC, TBO та EJS – між собою, а також з іншими відомими оптимізаторами. Для цього було проведено декілька експериментів, де алгоритми було випробувано на класичних оптимізаційних функціях, а також на прикладній задачі класифікації набору даних. Експерименти було проведено таким чином, щоб отримані результати можна було порівняти з результатами інших методів, які наведено у відкритих джерелах.

У ході дослідження було встановлено, що CHC виявився найбільш універсальним алгоритмом, а також він показав найкращі результати на функціях із багатьма локальними мінімумами. Алгоритм TBO виявився найповільнішим, але показав високу ефективність на функціях з довгим шляхом до глобального мінімуму. Алгоритм EJS виявився найшвидшим, але не завжди надійним. Проте виявилось, усі досліджувані алгоритми в багатьох випадках перевершують загальновідомі оптимізатори.

CHC, TBO, EJS, ОПТИМІЗАЦІЯ, ЗБІЖНІСТЬ ДО МІНІМУМУ, ГЛОБАЛЬНИЙ МІНІМУМ

ABSTRACT

The thesis contains: 113 pages, 32 figures, 31 tables, 33 sources.

This thesis is devoted to a comparative analysis of three modern swarm algorithms – CHC, TBO and EJS – with each other, as well as with other well-known optimisers. To this end, several experiments were conducted, where the algorithms were tested on classical optimisation functions, as well as on the applied task of classifying a data set. The experiments were carried out in such a way that the results obtained could be compared with the results of other methods available in open sources.

The study found that CHC turned out to be the most versatile algorithm, and it also showed the best results on functions with many local minima. The TBO algorithm turned out to be the slowest, but it showed high efficiency on functions with a long path to the global minimum. The EJS algorithm proved to be the fastest, but not always reliable. Nevertheless, it turned out that all the studied algorithms outperform the well-known optimisers in many cases.

CHC, TBO, EJS, OPTIMISATION, CONVERGENCE TO MINIMUM, GLOBAL MINIMUM

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CHC — Cock-Hen-Chicken
TBO — Tiger Beetle Optimization
EJS — Enhanced Jeallyish Algorithm
CSO — Chicken Search Optimization
GACO — Genetic-Ant Colony Optimization
GA — Genetic Algorithm
GWO — Grey Wolf Optimization
IGWO — Improved Grey Wolf Optimization
PSO — Particle Swarm Optimization
LEO — local escaping operator
MTDE — Multi-trial vector-based differential evolution
SQP — Sequential quadratic programming
YDSE — Young's Double-Slit Experiment
DE — Differential evolution
WSO — White Shark Optimizer
HGS — Hunger Games Search
AVOA — African vultures optimization algorithm
WOA — Whale Optimization Algorithm
SCA — Sine Cosine Algorithm
RSO — Rat Swarm Optimization
SMA — Slime Mould Algorithm
SHO — Sea-Horse Optimizer
DA — Dragonfly Algorithm
TSA — Tangent Search Algorithm
SFO — Sunflower Optimization
ChOA — Chimp optimization algorithm

ВСТУП

Актуальність дослідження. Алгоритми ройового інтелекту є комплексним методом вирішення складних задач оптимізації за допомогою моделювання поведінки популяцій різних видів тварин. У наш час наука про дані сконцентрована на вирішенні проблем, які потребують швидкої обробки та аналізу все більшого обсягу даних. Але більшість традиційних методів вирішення таких задач можна застосовувати тільки до неперервних і диференційованих функцій. Тож вчені довели, що набір методів, робота яких базується на популяційному підході, має великий потенціал у вирішенні задач оптимізації. Тому наразі розробка алгоритмів ройового інтелекту є одним з актуальних напрямів дослідження у сфері штучного інтелекту.

Метою дослідження є проведення якісного порівняльного аналізу новітніх ройових оптимізаторів, що дають змогу отримати шукані рішення проблем і завдань промисловості, які подаються у вигляді математичних моделей в екстремальній формі

Для досягнення основної мети випускної бакалаврської кваліфікаційної роботи необхідно виконати такі науково-практичні і прикладні **завдання**:

– провести огляд сучасних практичних задач, які потребують застосування ройового інтелекту для більш ефективного вирішення;

– виконати огляд та аналіз публікацій досліджень на цю тему;

– провести ретельний аналіз нових переспективних алгоритмів ройового інтелекту;

– виконати імплементацію й тестування цих ройових алгоритмів на математичних моделях;

– провести оцінку роботи реалізованих алгоритмів та порівняння їхньої ефективності між собою.

Об'єктом дослідження є складні процеси, що протікають в

будівництві, енергетиці, конструюванні тощо, а також відповідні їм математичні моделі, за допомогою яких можна визначити оптимальний режим функціонування систем, де відбуваються відповідні процеси.

Предметом дослідження є сучасні оптимізатори, що засновані на ройових принципах, які дають можливість визначити найоптимальніші або субоптимальні розв'язки науково-технічних проблем сьогодення у галузі будівництва, енергетики, конструювання тощо.

Методи дослідження. Аналіз сучасних науково-технічних публікацій в галузі ройового інтелекту, а також його застосування до вирішення прикладних завдань, математичне моделювання, моделювання ройових алгоритмів за допомогою Python коду, порівняльний аналіз ефективності алгоритмів, а також статистичні методи для обробки результатів експериментів.

Практичне значення дослідження полягає у розробці відповідної програми з метою аналізу процесів у складних системах та оптимізаційних математичних моделей, які полягають у пошуку оптимальних параметрів будівельних об'єктів, режимів їх роботи тощо. Розроблене програмне забезпечення також може бути корисним інженерам, що займаються аналізом нелінійних оптимізаційних моделей з обмеженнями у вигляді рівнянь і нерівностей.

Апробація результатів та публікації. Краснорyцька М.С., Хайдуров В.В. Сучасні методи ройового інтелекту для аналізу складних об'єктів і систем. Теоретичні і прикладні проблеми фізики, математики та інформатики: матеріали XXIII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених (14–17 травня 2025 р., м. Київ, Україна). – Київ : КПІ ім. Ігоря Сікорського, Видавництво «Політехніка», 2025. С. 340–343. ISBN 978-966-990-053-1.

1 ОГЛЯД НАУКОВО-ТЕХНІЧНИХ ЗАВДАНЬ І ПРОБЛЕМ, ЯКІ ПОТРЕБУЮТЬ ЗАСТОСУВАННЯ МЕТОДІВ РОЙОВОГО ІНТЕЛЕКТУ

1.1 Теоретичне підгрунтя

Оптимізація є важливою задачею багатьох галузях науки. Метою задач оптимізації є знаходження оптимального розв'язку або близького до нього. Пошук розв'язків такого роду задач можна звести до задачі максимізації або мінімізації функції або функціоналу. Слід зазначити, що пошук розв'язку таких задач здебільшого складається з певних етапів, оскільки за один підхід зазвичай складні оптимізаційні задачі розв'язати неможливо. Часто існує багато можливих варіантів вибору, тож мета в тому, щоб знайти найкращий з цих розв'язків. У задачах неперервної оптимізації немає скінченної кількості альтернативних рішень, тому ставиться задача визначення оптимальних значень змінних. В оптимізаційних задачах важливо проводити оцінку верхньої та нижньої межі складності [23].

Алгоритми глобальної оптимізації мають високі шанси знайти глобальний або наближене рішення, проте важливо розуміти, що жоден алгоритм не може гарантувати збіжність до глобального оптимуму у загальному випадку [28]. Омтиізаційні методи й алгоритми можна поділити на детерміновані, точні, еволюційні алгоритми та методи/алгоритми ройового інтелекту [9]. Детерміновані алгоритми дають один і той самий результат при запусках з одними й тими ж вхідними параметрами. Вони не мають стохастичних критеріїв та демонструють високу ефективність, оскільки загалом мають лінійну швидкість збіжності. Проте цей клас алгоритмів має суттєвий недолік: вони часто можуть застрягати в локальних мінімумах або «ламатися» через обчислювальну складність. Ще одним класом інструментів

оптимізації є точні методи (при дотриманні певних умов вони можуть досягати дуже точних результатів). Завдяки використанню похідних досягається надлінійна швидкість збіжності до стаціонарних точок цільової функції Лагранжа [9].

Еволюційні алгоритми здебільшого є метаевристиками, які базуються на процесі природної еволюції. Точки популяції ініціалізуються випадково, а потім поітераційно еволюціонують за допомогою операторів варіації та селекції. Кожна ітерація – нове покоління [6]. Оператори варіації генерують зміни в просторі пошуку. Після кожної ітерації обчислюється значення цільової функції/функціонала для кожного рішення. Відбір усуває найгірші рішення та залишає лише ті, від яких генеруються кращі рішення. Кожен алгоритм використовує власний набір операторів, іноді натхненних мутацією та схрещування в біологічній еволюції: оператор мутації – схожий на локальний пошук, він змінює один компонент рішення або кілька. Оператор схрещування – комбінує рішення для дослідження проміжкового простору [6]. *Ройовий інтелект* є галуззю штучного інтелекту, яка спрямована на створення систем, основаних на популяційному підході, натхненних колективною поведінкою істот, які співіснують в купі одне з одним та колективно прямують до спільної мети. Координована поведінка популяцій виникає з відносно простих дій або взаємодій між її членами. Відомими прикладами алгоритмів, які використовують ройовий підхід, є алгоритм оптимізації рою частинок (PSO), що моделює природний процес того, як окремі члени зграї птахів користуються досвідом інших членів популяції, що отримується у процесі полювання, алгоритм перестрибування жаб (SFLA), який походить від PSO та моделює процес того, як жаби навчаються полюванню на основі отриманого досвіду іншими жабами [32], (ABC) – для моделювання ієрархії в бджолиному та процесу пошуку їжі на основі цієї системи тощо. Алгоритми ройового інтелекту складаються фази варіацій та селекції. Фаза варіацій відповідає за дослідження простору пошуку, а фаза селекції – за використання отриманої інформації для покращення

результату. Такі алгоритми створюються за принципами децентралізації, самоорганізації, виникнення, стигмергії, адаптивності та надійності [26].

1.2 Розробка загальнонаціональної політики зберігання енергії шляхом вибору оптимального розміру та місця

Пристрої накопичення енергії забезпечують зміщення виробництва і споживання енергії в часі, вони можуть допомогти інтегрувати нестабільні або переривчасті джерела генерації та навантаження. Дослідження в цій галузі можна умовно поділити на два типи. Один підхід припускає, що пристрої зберігання використовуються для максимізації прибутку на ринку електроенергії а інший – що накопичувачі керуються системними операторами для зниження витрат та підвищення надійності мережі [25].

За даними Міжнародного енергетичного агентства, споживання електроенергії росте, що вимагає збільшення виробництва. У США з 2014 по 2015 роки обсяг впровадження накопичувальних систем зріс на 188%. Принцип такий, що системи накопичують енергію при низькому попиті та віддають в мережу під час піків споживання. Це формує задачу оптимізації.

Модель охоплює диспетчеризацію та оптимальний потік потужності з урахуванням переваг. Туди включено розміри пристроїв зберігання енергії, щоб врахувати вплив на фінанси країни.

1.2.1 Моделювання системи зберігання енергії у моделі диспетчеризації одиниць

Система зберігання енергії (СЗЕ) може поводитися як генератор або як навантаження, але не є жодною із них, оскільки її поведінка обмежена максимальною потужністю $\overline{P}_{\mathrm{ES}}$ і максимальною ємністю $\overline{E}_{\mathrm{ES}}$. Замість

двох окремих змінних вводиться єдина матрична змінна $e \in \mathbb{R}_{+}^{T\times T}$. Верхній трикутник матриці містить кількість зарядженої енергії, а нижній трикутник – кількість розрядженої енергії. Математична модель подається у вигляді цільової функції:

$$\min_{P,s,h,u,e} \sum_{t=1}^{T}\sum_{g=1}^{G}\big(B_g\,\beta\,P_{g,t} + C_g\,s_{g,t} + E_g\,h_{g,t} + A_g\,u_{g,t}\big). \tag{1.1}$$

На функцію (1.1) накладаються обмеження у вигляді:

$$u_{g,t}\,\underline{P}_g \le P_{g,t} \le \overline{P}_g\,u_{g,t}, \qquad \forall g,t \tag{1.2}$$

$$-R_g^{\mathrm{D}} \le P_{g,t} - P_{g,t-1} \le R_g^{\mathrm{U}}, \qquad \forall g,t \tag{1.3}$$

$$\sum_{\tau=t-M_g^{\mathrm{U}}+1}^{t} s_{g,\tau} \le u_{g,t}, \qquad \forall g,t \tag{1.4}$$

$$\sum_{\tau=t-M_g^{\mathrm{D}}+1}^{t} h_{g,\tau} \le 1 - u_{g,t}, \qquad \forall g,t \tag{1.5}$$

$$s_{g,t} - h_{g,t} = u_{g,t} - u_{g,t-1}, \qquad \forall g,t \tag{1.6}$$

$$\sum_{g=1}^{G} \overline{P}_g\,u_{g,t} \ge D_t + R_t, \qquad \forall t. \tag{1.7}$$

Окрім обмежень (1.2)–(1.7) вводять також додаткові обмеження на баланс потужності з урахуванням СЗЕ:

$$\sum_{g=1}^{G} P_{g,t} + \sum_{\tau=1}^{t} e_{t,\tau} \ge D_t + \sum_{\tau=t}^{T} e_{t,\tau}, \qquad \forall t \tag{1.8}$$

$$\sum_{g=1}^{G} \overline{P}_g\,u_{g,t} \ge D_t + R_t + \sum_{\tau=t}^{T} e_{t,\tau}, \qquad \forall t \tag{1.9}$$

$$e_{t,\tau} \ge 0, \qquad \forall t,\tau. \tag{1.10}$$

Формули (1.8)–(1.9) оновлюють баланс мережі і явно трактують заряд СЗЕ як додаткове споживання, а розряд – як генерацію.

1.2.2 Оптимальне розміщення та розмір систем зберігання енергії

Після отримання оптимального графіку UC планувальна задача визначає, де і якого розміру слід встановити СЗЕ, щоб мінімізувати сумарні витрати генерації.

$$\min \sum_{t=1}^{T} \sum_{g=1}^{G} f_g\big(P_{g,t}\big), \tag{1.11}$$

Обмеження на (1.11) класичного UC:

$$f_{k,t} = B_{f_k}\big(\theta_{i,t} - \theta_{j,t}\big), \qquad \forall k,t \tag{1.12}$$

$$\sum_{g\in i} P_{g,t} - D_{i,t} = \sum_{k\in i,*} f_{k,t} - \sum_{k\in *,i} f_{k,t} + \sum_{\tau=t}^{T} e_{\tau,t,i} - \sum_{\tau=1}^{t} e_{\tau,t,i}, \qquad \forall i,t \tag{1.13}$$

$$-\overline{f}_{k,t} \le f_{k,t} \le \overline{f}_{k,t}, \qquad \forall k,t \tag{1.14}$$

Окрім обмежень (1.12)–(1.14) є ще операційні обмеження генерації:

$$u_{g,t}\, P_g \le P_{g,t} \le \overline{P}_g\, u_{g,t}, \qquad \forall g,t \tag{1.15}$$

$$-R_g^{\mathrm{D}} \le P_{g,t} - P_{g,t-1} \le R_g^{\mathrm{U}}, \qquad \forall g,t \tag{1.16}$$

Додатково до (1.15)–(1.16) вводять глобальні ліміти інвестицій у СЗЕ:

$$\sum_{i=1}^{N_{\mathrm{bus}}} x_i \le \overline{P}_{\mathrm{ES}}, \qquad \sum_{i=1}^{N_{\mathrm{bus}}} y_i \le \overline{E}_{\mathrm{ES}}. \tag{1.17}$$

До глобальних лімітів (1.17) вводять локальні ліміти кожної СЗЕ:

$$\sum_{\tau=1}^{t} e_{\tau,t,i} \leq x_i, \qquad \forall i,t \tag{1.18}$$

$$\sum_{\tau=t}^{T} e_{\tau,t,i} \leq x_i, \qquad \forall i,t \tag{1.19}$$

$$\sum_{\tau=1}^{t} \sum_{\alpha=t+1}^{T} e_{\tau,\alpha,i} \leq y_i, \qquad \forall i,t \tag{1.20}$$

$$x_i,\, y_i \geq 0, \qquad \forall i. \tag{1.21}$$

Обмеження (1.18)–(1.19) гарантують, що миттєвий заряд/розряд не перевищує встановленою потужності x_i, а (1.20) стежить за станом заряду (SOC) у межах ємності y_i.

1.2.3 Максимальний розмір систем зберігання енергії

Під час розробки треба визначити потреби потужності в межах країни. Це можливо дізнатися з моделі диспетчеризації із системою зберігання енергії. Проте в моделі витрати на системи зберігання енергії та обмеження на їхній розмір не враховані. Матриця змінної e містить рішення відносно заряджання та розряджання в точці оптимального рішення щогодини. Треба розрахувати погодинну диспетчеризацію (1.23), визначити максимальні значення заряджання та розряджання (1.25), а також проаналізувати стан заряду на момент оптимальної експлуатації (1.22). З (1.24) максимальний рівень заряду відповідає максимальній енергетичній ємності.

$$SOC(t) = \sum_{i=1}^{t} \sum_{j=t+1}^{T} e_{i,j}, \tag{1.22}$$

$$P^{ES}(t) = \begin{cases} SOC(t) & \text{if } t = 1, \\ SOC(t) - SOC(t-1) & \text{if } t \neq 1, \end{cases} \tag{1.23}$$

$$E_{max}^{ES} = \max(SOC), \tag{1.24}$$

$$P_{max}^{ES} = \max\left(\max(P^{ES}), |\min(P^{ES})|\right). \tag{1.25}$$

Математична модель (1.22)–(1.25) максимізує розмір системи зберігання енергії.

1.2.4 Розподіл систем зберігання енергії в межах мережі

Також необхідно визначити місця розташування та розміри енергетичних сховищ. Модель оптимального розміщення сховищ враховує мережу вузлів з обмеженнями передачі, енергетичним балансом, обмеженнями генераторів і змінами потужності. Постобробка значень e може виявити максимальні потреби енергосистеми у потужності та енергії. Щоб визначити максимальну потужність, можна проаналізувати погодинний диспетчинг як у рівнянні (1.27). Отже, максимальне значення заряджання або розряджання за годину визначається за рівнянням (1.29).

Аналогічно, щоб знайти максимальну енергетичну ємність, слід спостерігати за станом заряду в точці оптимальної роботи, як у (1.26). Отже, максимальний спостережений рівень заряду є максимальною енергетичною ємністю, згідно з (1.28) [15]:

$$SOC(t,k) = \sum_{i=1}^{t} \sum_{j=t+1}^{T} e_{i,j,k}, \tag{1.26}$$

$$p^{ES}(t,k) = \begin{cases} SOC(t,k), & \text{if } t = 1 \\ SOC(t,k) - SOC(t-1,k), & \text{if } t \neq 1 \end{cases} \tag{1.27}$$

$$E_{\max}^{ES}(k) = \max(SOC(k)), \tag{1.28}$$

$$P_{\max}^{ES}(k) = \max(\max(p^{ES}(k)), |\min(p^{ES}(k))|). \tag{1.29}$$

Математична модель (1.26)–(1.29) дає можливість виконати оптимальний розподіли систем зберігання енергії в мережі.

1.3 Діагностика несправності біореактора на основі метаевристичних ройових методів й алгоритмів

Діагностика несправностей є ключовою проблемою в промисловості, бо надійність, безпека і ефективність, залежать від коректної діагностики систем. Біореактор – це апарат для культивування мікроорганізмів, рослинних і тваринних клітин, в якому відбуваються біохімічні реакції [33]. У хімічній та біохімічній промисловості доволі поширено використання біореакторів, тож коректна робота таких приладів має велике значення для обох галузей.

Основною задачею є оптимізація якості діагностики несправності біореактора, що оцінюється за часом і точністю.

1.3.1 Формулювання умови оптимізаційної задачі

Головною суттю задачі є відслідковування динаміки несправностей. У задачі несправності ділять на такі підкатегорії:
1. Несправності виконавчих механізмів: $f_u = \mathbb{R}^p$.
2. Несправності процесів: $f_p = \mathbb{R}^q$.
3. Несправності датчиків: $f_y = \mathbb{R}^m$.
Тоді динаміку несправностей можна представити у вигляді вектора

$$f = (f_u, f_p, f_y)^t, f \in \mathbb{R}^{p+q+m}.$$

Опишемо модель, яка враховує динаміку несправностей:

$$\begin{aligned} \dot{x}(t) &= g(x(t), u(t), \theta, f), \\ y(t) &= h(x(t), f), \\ x(t_0) &= x_0, \end{aligned}$$

де:

– $x(t) \in \mathbb{R}^n$ – вектор стану системи;

– $u(t) \in \mathbb{R}^p$ – вектор вхідних сигналів;

– $\theta \in \mathbb{R}^l$ – параметри моделі;

– $t \in [t_0, t_1]$;

– $y(t) \in \mathbb{R}^m$ – вихідний вектор, який вимірюється датчиками, що можуть бути під впливом несправностей f_y.

Вважаємо, що динаміка несправності не залежить від часу:

$$\min F(\hat{f}) = \sum_{t=1}^{N_s} \left\| y_t(u, f) - \hat{y}_t(u, \hat{f}) \right\|^2,$$

за умови виконання нерівності:

$$f_{\min} \leqslant \hat{f} \leqslant f_{\max},$$

де:

– N_s – кількість моментів часу для вибірки;

– $\hat{y}_t\left(u, \hat{f}\right)$ – вектор оцінених вихідних значень, отриманих із моделі

– $y_t(u, f)$ – вихідний вектор, виміряний датчиками.

Розв'язок цієї задачі є оцінкою для вектора несправностей $\hat{f}$. Якщо оцінене значення компоненти f_i менше нуля, це свідчить про наявність несправності в системі, а його модуль визначає інтенсивність цієї несправності.

1.3.2 Ключова задача біореактора

Ключова задача біореактора передбачає дослідження його несправностей. Нехай біореактор синтезує мікроорганізми та субстрат, концентрацію позначимо як ξ_1 та ξ_2. Відповідно визначимо вектор стану:

$$x \in \mathbb{R}^2, \quad x = \begin{bmatrix} x_1 \\ \\ x_2 \end{bmatrix}, \quad x_1 = \xi_1, \quad x_2 = \frac{a_1\xi_1\xi_2}{a_2\xi_1 + \xi_2}.$$

Далі опишемо весь процес синтезу за допомогою рівнянь:

$$\dot{x}_1 = x_2 - \eta x_1,$$

$$\dot{x}_2 = -a_2x_2^2 - a_1\eta x_1^2 + \frac{(a_1x_1 - x_2)^2(a_4\eta - a_3x_2)}{a_1a_2x_1^2},$$

$$x(t_0) = x_0, \quad y = x_1.$$

У цій системі $a_1, a_2, a_3, a_4 \in \mathbb{R}$ – деякі параметри, а $\eta \in \mathbb{R}$ – коефіцієнт швидкості розведення. Після синтезу вимірюється концентрація мікроорганізмів, її визначаємо за допомогою рівняння $y = x_1$. Також існує два типи збоїв біореактора, які можна виразити в якості рівнянь:

$$fp_1 = \theta_1\Psi_1, \quad fp_2 = \theta_2\Psi_2.$$

Якщо в моделі опису процесу синтезу врахувати збої, то можна записати систему наступним чином:

$$\dot{x}_1 = x_2 - \eta x_1 + \theta_1\Psi_1 - \theta_2\Psi_2,$$

$$\dot{x}_2 = -a_2x_2^2 - a_1\eta x_1^2 + \frac{(a_1x_1 - x_2)^2(a_4\eta - a_3x_2)}{a_1a_2x_1^2},$$

$$x(t_0) = x_0, \quad y = x_1.$$

Задамо параметри системи та початкову умову:

$$a_1 = 1.0, a_2 = 1.0, a_3 = 1.0, a_4 = 0.1,$$

$$x(0) = \begin{bmatrix} 0.05 \\ \\ 0.025 \end{bmatrix}.$$

Маємо, що вихідний сигнал задається за таких умов: Вхідний сигнал:

$$u(t) = \begin{cases} 0.08, & 0\,\text{год} \leqslant t < 10\,\text{год}, \\ 0.02, & 10\,\text{год} \leqslant t < 20\,\text{год}, \\ 0.08, & t \geqslant 20\,\text{год}. \end{cases}$$

1.3.3 Формулювання задачі виявлення та ізоляція дефектів

Суть задачі зводиться до оцінки значення θ_1 і θ_2 шляхом розв'язку задачі оптимізації:

$$\min_{\hat{\theta}_1, \hat{\theta}_2} F = \sum_{t=1}^{N_s} \left(y_t - \hat{y}_t(\hat{\theta}_1, \hat{\theta}_2, u) \right)^2$$

з урахуванням обмежень:

$$\theta_1^{(\min)} \leqslant \hat{\theta}_1 \leqslant \theta_1^{(\max)}, \quad \theta_2^{(\min)} \leqslant \hat{\theta}_2 \leqslant \theta_2^{(\max)}.$$

Тут y_t – виміряний вихід у момент часу t, а $\hat{y}_t$ – оцінений вихід, отриманий за моделлю.

1.3.4 Метаевристичні алгоритми для розв'язку задач діагностики несправностей

Існує 3 основні метаерестічні алгоритми, які застосовуються для цього: оптимізація методом мурашиних колоній із дисперсією, алгоритм диференціальної еволюції з зіткненням частинок, метод еволюційної стратегії з адаптацією матриці коваріацій.

Оптимізація методом ACO-d – модифікований для рішення проблем неперервної оптимізації варіант звичайної оптимізації ACO. Він моделює поведінку мурах, які шукають найкоротший шлях між колонією та джерелом їжі з урахуванням відкладання й випаровування феромонів, але оновлення матриці феромонів реалізується з урахуванням ефекту дисперсії, а це, в свою чергу, дає змогу враховувати сусідні до найкращого рішення значення. На кожній ітерації Iter алгоритм генерує нову популяцію мурах $P_{\text{Iter}} = \{X_1^{\text{Iter}}, X_2^{\text{Iter}}, \ldots, X_Z^{\text{Iter}}\}$, використовуючи ймовірнісну матрицю P_C, яка залежить від феромонної матриці F. Кожен елемент матриці P_C обчислюється за формулою:

$$p_{c_{ij}}(\text{Iter}) = \frac{\sum_{j=1}^{j} f_{ij}(\text{Iter})}{\sum_{j=1}^{k} f_{ij}(\text{Iter})},$$

де f_{ij} – елементи матриці F, що визначають рівень феромонів для j-го значення i-ї змінної. Матриця F оновлюється на кожній ітерації за формулою:

$$f_{ij}(\text{Iter}) = (1 - C_{\text{evap}}) f_{ij}(\text{Iter} - 1) + \delta_{ij,\text{best}} C_{\text{inc}} f_{ij}(\text{Iter} - 1),$$

де

$$\delta_{ij,\text{best}} = \begin{cases} 1, & \text{якщо } x_j^i = x_i^{\text{best}}, \\ 0, & \text{в іншому випадку}, \end{cases}$$

а x_i^{best} – i-та компонента найкращого розв'язку X_{best}. Оптимізація методом

мурашиних колонії включає коефіцієнт дисперсії C_{dis}, що використовується для розрахунку феромонного відкладення на сусідніх до X_{best} значеннях. Масив таких сусідів $V(x_n^{\text{best}})$ визначається як:

$$V(x_n^{\text{best}}) = \{x_n^m : d(x_n^{\text{best}}, x_n^m) < d_{\max}, 1 \leqslant m \leqslant k\},$$

де $d_{\max}$ обчислюється як середнє значення половини всіх можливих відстаней між x_n^m і x_n^r:

$$d_{\max} = \frac{h + 2h + 3h + \cdots + \frac{k}{2}h}{\frac{k}{2}},$$

з $h = \frac{b-a}{k}$, де $x_n \in [a, b]$. Феромонний слід для $x_n^m \in V(x_n^{\text{best}})$ оновлюється так:

$$f_{nm}(\text{Iter}) = f_{nm}(\text{Iter}) + \frac{C_{\text{dis}}}{|m - m^{\text{best}}|}.$$

Кожен z-й агент нової колонії генерується за схемою:

$$x_n^{(z)} = \begin{cases} x_n^{m^{\text{best}}}, & \text{якщо } q_{\text{rand}} < q_0, \\ x_n^{m'}, & \text{в іншому випадку,} \end{cases}$$

де $m^{\text{best}} : f_{nm^{\text{best}}} \geqslant f_{nm}, \forall m = 1, 2, \ldots, k$, а m' задовольняє умови $p_{cnm'} > q_{\text{rand}}$ та $p_{cnm'} \leqslant p_{cnm}, \forall m \geqslant m'$.

Алгоритм диференціальної еволюції із зіткненням частинок є покращенною версією алгоритму диференціальної еволюції шляхом інтегрування методик з алгоритму зіткнень частинок. Алгоритм складається з таких етапів як мутація, схрещування та селекція. Цей алгоритм має 3 вхідних параметри: розмір популяції Z; коефіцієнт схрещування C_{cross}; та коефіцієнт масштабування C_{scal}.

Різниця між DEwPC та DE в процедурі відбору. В DEwPC додається новий параметр MaxIter_c до оригінальної версії DE.

Розглянемо задачу оптимізації:

$$\min f, \quad \text{де } f : D \subset \mathbb{R}^n \to \mathbb{R}.$$

На кожній ітерації Iter DEwPC генерує нову популяцію з Z допустимих кандидатів рішень $P_{\text{Iter}} = \{X_1^{\text{Iter}}, X_2^{\text{Iter}}, \ldots, X_Z^{\text{Iter}}\}$ для мінімізаційної задачі.

Нехай маємо початкову популяцію $P_0 = \{X_1^0, X_2^0, \ldots, X_Z^0\}$, яка складається з Z можливих кандидатів рішень. Ці кандидати є векторами у $\mathbb{R}^n$.

На кожній ітерації до кожного елемента популяції з попередньої ітерації послідовно застосовуються оператори мутації, схрещування та селекції. У результаті формується нова популяція на поточній ітерації, а найкраще рішення оновлюється. Ця схема виконується до моменту, коли найкраще рішення, надане алгоритмом, відповідає попередньо встановленому критерію зупинки. Механізм генерації DEwPC, як і у DE, узагальнюється наступною нотацією:

$$\text{DE}/X_\delta^{\text{Iter}-1}/\gamma/\lambda,$$

де $X_\delta^{\text{Iter}-1}$ та γ характеризують мутацію, а λ – схрещування. γ вказує кількість пар рішень із P_{Iter}, які використовуються для збурення останнього рішення $X_\delta^{\text{Iter}-1}$, а λ представляє функцію розподілу, яка використовується під час схрещування.

Оператор мутації DEwPC описується як збурення кожного члена популяції з попередньої ітерації з іншими членами популяції. Схему мутації можна узагальнити як $X_\delta^{\text{Iter}-1}/\gamma$, де $X_\delta^{\text{Iter}-1}$ вказує на вектор, що підлягає збуренню, а γ – кількість попарних векторів, вибраних випадковим чином. Цей процес виконується Z разів для генерації кандидатної популяції $\hat{P}_{\text{Iter}} = \{\hat{X}_1^{\text{Iter}}, \hat{X}_2^{\text{Iter}}, \ldots, \hat{X}_Z^{\text{Iter}}\}$. Елементи $\hat{P}_{\text{Iter}}$

називаються векторами-донорами. У DEwPC використовується схема:

$$\hat{X}_z^{\text{Iter}} = X_{\text{best}} + C_{\text{scal}} \cdot (X_\alpha^{\text{Iter}-1} - X_\beta^{\text{Iter}-1}),$$

де $X_{\text{best}}, X_\alpha^{\text{Iter}-1}, X_\beta^{\text{Iter}-1} \in \mathbb{R}^n$ – рішення з $P_{\text{Iter}-1}$, а C_{scal} – параметр мутації.

Процес схрещування в DEwPC є аналогічним до схрещування в DE. Він застосовується до кожного компонента векторів-донорів $\hat{X}_z^{\text{Iter}}$, отриманих після мутації. Формула для схрещування:

$$\hat{x}_{z,(n)}^{\text{Iter}} = \begin{cases} \hat{x}_{z,(n)}^{\text{Iter}}, & \text{якщо } q_{\text{rand}} \leqslant C_{\text{cross}}, \\ x_{\delta,(n)}^{\text{Iter}-1}, & \text{інакше,} \end{cases}$$

де C_{cross} – коефіцієнт схрещування, а q_{rand} – випадкове число.

В свою чергу селекція в DEwPC алгоритмі походить від алгоритму PCA. Вона виконується за наступних правил:

– якщо $f(\hat{X}_z^{\text{Iter}}) \leqslant f(X_\delta^{\text{Iter}-1})$, тоді до $\hat{X}_z^{\text{Iter}}$ застосовується абсорбція;

– в інших випадках виконується розсіювання з деякою ймовірністю.

Метод еволюційної стратегії з адаптацією матриці коваріацій також є одним з метаевристичних еволюційних алгоритмів. Спочатку цю модель пропонували для задач пошуку локальних екстремумів, але виявилося, що із задачами глобального вона також добре справляється.

На кожній ітерації Iter алгоритм генерує нову популяцію λ допустимих розв'язків $\{X_{\text{Iter}}^1, X_{\text{Iter}}^2, \ldots, X_{\text{Iter}}^\lambda\}$ за допомогою процесу мутації на основі інформації про кращі розв'язки, які було отримано на попередній ітерації: $\{X_{\text{Iter}-1}^{\mu(1)}, X_{\text{Iter}-1}^{\mu(2)}, \ldots, X_{\text{Iter}-1}^{\mu(\mu)}\}$.

Процес мутації відбувається за таким законом:

$$X_{\text{Iter}}^z = X_{\text{Iter}-1}^{\text{wmean}} + \sigma_{\text{Iter}-1}\mathcal{N}(0, C_{\text{Iter}-1}), \quad z = 1, 2, \ldots, \lambda,$$

де:

– $X_{\text{Iter}-1}^{\text{wmean}} \in \mathbb{R}^n$ – зважене середнє μ найкращих рішень на попередній ітерації;

– $\sigma_{\text{Iter}-1}$ – параметр масштабування, що оновлюється на кожній ітерації;

– $C_{\text{Iter}-1}$ – матриця коваріації.

Розв'язок X^z_{Iter} має нормальний розподіл з параметрами $\mathcal{N}(X^{\text{wmean}}_{\text{Iter}-1}, \sigma^2_{\text{Iter}-1}C_{\text{Iter}-1})$.

Еволюція масштабу σ_{Iter} визначається шляхом:

$$\sigma_{\text{Iter}} = \sigma_{\text{Iter}-1} \exp\left(\frac{1}{d_\sigma}\left(\frac{\|p_\sigma(\text{Iter}-1)\|}{\hat{\xi}_n} - 1\right)\right),$$

де:

– $d_\sigma \geqslant 1$ – коефіцієнт глушіння;

– $\hat{\xi}_n = \mathbb{E}[\mathcal{N}(0, I)]$ – очікувана довжина вектора зі стандартного нормального розподілу;

– $p_\sigma(\text{Iter}-1)$ – еволюційний шлях.

Еволюційний шлях, в свою чергу, $p_\sigma(\text{Iter})$ обчислюється за формулою:

$$p_\sigma(\text{Iter}) = (1-c_\sigma)p_\sigma(\text{Iter}-1) + \\ + \sqrt{c_\sigma(2-c_\sigma)} \cdot B_{\text{Iter}-1} D^{-1}_{\text{Iter}-1} B^{-1}_{\text{Iter}-1} \frac{X^{\text{wmean}}_{\text{Iter}} - X^{\text{wmean}}_{\text{Iter}-1}}{\sigma_{\text{Iter}-1}}.$$

Матриця C_{Iter} оновлюється з урахуванням еволюційного шляху $p_c(\text{Iter})$:

$$C_{\text{Iter}} = (1-c_{\text{cov}})C_{\text{Iter}-1} + c_{\text{cov}} p_c(\text{Iter}) p_c(\text{Iter})^T,$$

де $c_{\text{cov}} \in [0, 1]$ – швидкість зміни матриці коваріації. На кожній ітерації вибираються μ найкращих векторів за правилом:

$$f(X^{\mu(1)}_{\text{Iter}}) \leqslant f(X^{\mu(2)}_{\text{Iter}}) \leqslant \cdots \leqslant f(X^{\mu(\mu)}_{\text{Iter}}),$$

де кожен розв'язок має вагу w_i. Зважене середнє розраховується як:

$$X^{\text{wmean}}_{\text{Iter}} = \frac{\sum_{i=1}^{\mu} w_i X^{\mu(i)}_{\text{Iter}}}{\sum_{i=1}^{\mu} w_i}.$$

1.3.5 Аналіз результатів діагностики за допомогою алгоритмів

У таблиці 1.2 представлені значення SR та SP, отримані кожним алгоритмом під час діагностики випадків 1–5 (див. таблицю 1.1). Алгоритм зупиняв роботу за однієї з умов: або кількість ітерацій досягла

Таблиця 1.1 – Тестові випадки

Випадок	**Випадок 1**	**Випадок 2**	**Випадок 3**	**Випадок 4**	**Випадок 5**
θ_1	0.01	0.01	0.01	0.01	0.01
θ_2	0.015	0.015	0.015	0.015	0.015
Рівень шуму	2%	5%	8%	10%	15%

200, або відсоток відносної помилки у значеннях оцінених несправностей сягав $Err\theta 1 = 5\%$ та $Err\theta 2 = 5\%$ (цей випадок вважали успішним).

Видно, що SR зменшується від випадку 1 до 5 через зростання шуму. ACO-d та DEwPC досягли SR = 100% у випадках 1 і 2, тобто надійність діагностики $\geqslant 95\%$. Найкраще значення SR CMA-ES становило SR = 96%, що відповідає випадкам 1 та 2.

З Таблиці 2.2 також видно, що для випадків 3 і 4 індикатори SR для ACO-d і DEwPC мають схожі значення, але меншими, ніж у випадках 1 і 2. Алгоритм (μ_w, λ) CMA-ES продемонстрував подібну поведінку з ACO-d і DEwPC у випадку 3, але у випадку 4 його SR є дуже малим.

У випадку 5, як видно з Таблиці 2.2, алгоритми ACO-d і DEwPC не досягли успішних запусків, але (μ_w, λ) CMA-ES досягнув SR = 12%.

На основі значень SR з Таблиці 2.2 маємо висновок, що для рівня шуму до 5% діагностика несправностей із 95% надійністю є на 100% успішною. Проте для шуму до 10% рівень успіху знижується до 84% для ACO-d і DEwPC. Для шуму більше 10% неможливо діагностувати несправності із 95% надійністю за допомогою DEwPC та ACO-d. Алгоритм (μ_w, λ) CMA-ES тоді має успіх 12%.

Індикатор SP може бути використаний для аналізу обчислювальної вартості. Таблиця 7.4 показує, що для випадків 1–3 значення SP є подібними. Тобто обчислювальна вартість діагностики цих несправностей із відносною помилкою менше ніж 5% і рівнем шуму до 8% є подібною для трьох алгоритмів.

Таблиця 1.2 – Значення індикаторів SR та SP для алгоритмів ACO-d, DEwPC та (μ_w, λ) CMA-ES для випадків 1–5

Випадок	ACO-d		DEwPC		(μ_w, λ) CMA-ES	
	SR	SP	SR	SP	SR	SP
1	100%	320	100%	311	96%	305
2	100%	280	100%	290	96%	321
3	84%	453	88%	340	72%	361
4	84%	381	84%	363	16%	2137
5	0%	–	0%	–	12%	2250

Для випадку 4 значення SR, отримане за допомогою алгоритму (μ_w, λ) CMA-ES, майже в шість разів вище за значення, отримані іншими двома алгоритмами. Для випадку 5, який відповідає рівню шуму до 15%, алгоритм (μ_w, λ) CMA-ES показує SP, подібний до його результатів для випадку 4 [21].

1.4 Інші науково-прикладні моделі реальних процесів, що аналізуються з використанням сучасного ройового підходу

1.4.1 Оптимальне планування виробництва та розподілу в ланцюзі постачання

Сучасні ланцюги постачання складаються з кількох етапів, які включають в себе постачання, виробництво, дистриб'юцію та розподіл. Одним із найважливіших викликів є оптимізація виробничих і

розподільчих процесів за умов невизначеності, яка пов'язана з попитом, запасами та умовами транспортування [22]. Математична модель включає наступні компоненти:

– S – множина постачальників ($s = 1, 2, \ldots, S$);
– P – множина заводів ($p = 1, 2, \ldots, P$);
– W – множина розподільчих центрів ($w = 1, 2, \ldots, W$);
– Z – множина зон клієнтів ($z = 1, 2, \ldots, Z$);
– T – множина часових періодів ($t = 1, 2, \ldots, T$);
– R – множина сировин ($r = 1, 2, \ldots, R$);
– G – множина готової продукції ($g = 1, 2, \ldots, G$).

Математична модель у нечіткому вигляді набуває такого виду:

$$\begin{aligned}\min Z_1 = & \sum_{r,s,p,t} (SC_{1rst} + SC_{2rst} + SC_{3rst} + SC_{4rst} \cdot q_{rst}) \\ & + \sum_{g,p,t} (PC_{1gpt} + PC_{2gpt} + PC_{3gpt} + PC_{4gpt} \cdot y_{gpt}) \\ & + \sum_{r,s,p,t} (TR_{rspt} \cdot x_{rspt}) + \sum_{g,p,w,t} (TG_{gpwt} \cdot m_{gpwt}) \\ & + \sum_{g,w,z,t} (TW_{gwzt} \cdot n_{gwzt}) + \sum_{r,p,t} (HR_{rpt} + RI_{rpt}) \\ & + \sum_{g,w,t} (HW_{gwt} + WI_{gwt}), \end{aligned} \tag{1.30}$$

$$\min Z_2 = \sum_{g,w,z,t} (gwzt_1 + gwzt_2 + gwzt_3 + gwzt_4 \cdot n_{gwzt})$$

за умовами:

$$DD_{1gzt} + DD_{2gzt} + DD_{3gzt} + DD_{4gzt} \cdot n_{gwzt} \leqslant w, \quad \forall g, z, t,$$

$$y_{gpt} \leqslant cap_{1gpt} + cap_{2gpt} + cap_{3gpt} + cap_{4gpt}, \quad \forall g, p, t,$$

$$\sum_{p} m_{gpwt} \leqslant V_{1gwt} + V_{2gwt} + V_{3gwt} + V_{4gwt}, \quad \forall g, w, t.$$

Обчислювальні результати застосування MOPSO. Запропоновану модель було протестовано на восьми прикладах, згенерованих випадковим чином. Усі експерименти виконувалися в MATLAB на комп'ютері з процесором Intel® Core™ Duo та 2.2 GB RAM.

Таблиця 1.3 – Результати запропонованої моделі

Problem	P	S	W	Z	NPS	SM	DM	CPU time
1	3	2	2	2	7.00	0.022	0.736	7.83
2	3	2	4	6	5.66	0.023	0.667	9.49
3	5	3	4	6	6.00	0.010	1.058	10.36
4	5	3	6	8	7.00	0.004	0.798	11.86
5	10	4	6	8	8.33	0.005	0.589	19.21
6	10	4	8	10	6.66	0.085	0.593	23.69
7	20	5	8	10	5.00	0.004	0.601	32.20
8	20	5	10	12	7.00	0.004	0.685	37.17

1.4.2 Оптимізація ливарного виробництва

Лиття під тиском є одним з найпоширеніших методів виготовлення різних деталей. Одним з найголовніших завдань лиття під тиском є забезпечення якості виробів та мінімізація проявлення дефектів, що можуть виникнути при неправильному охолодженні, підборі надто високих або низьких температур, тисків тощо [2].

Недостатня оптимізація параметрів може призвести до погіршення якості деталей. На сьогоднішній день не існує універсального методу для визначення оптимальних параметрів лиття під тиском, тому вони визначаються експериментальним шляхом, що збільшує витрати на виробництво.

Використання алгоритму рою частинок для оптимізації

процесу. У даній задачі використовували алгоритм рою частинок є методом штучного інтелекту, що використовує групову поведінку для вирішення задач оптимізації. Цей метод було обрано, бо цей метод є одним з найшвидших та універсальниших, тому легше прослідковувати тенденції в залежності від параметрів

Математична модель. Для оцінки викривлення деталей використовується регресійна модель:

$$\begin{aligned} W = {} & 4.09888 - 0.11621\,T_{\text{cool}} - 0.12179\,t_{\text{inject}} + 0.00587\,\frac{V}{P} - 0.00387\,T_{\text{mold}}, \\ & - 0.02497\,T_{\text{cool}}^2 - 0.00509\,t_{\text{inject}}^2 - 0.01734\,\left(\frac{V}{P}\right)^2 - 0.00772\,T_{\text{mold}}^2, \\ & - 0.04706\,T_{\text{cool}} \cdot t_{\text{inject}} + 0.00419\,T_{\text{cool}} \cdot \frac{V}{P} - 0.00106\,T_{\text{cool}} \cdot T_{\text{mold}}, \\ & + 0.01294\,t_{\text{inject}} \cdot \frac{V}{P} - 0.00106\,t_{\text{inject}} \cdot T_{\text{mold}} - 0.00056\,\frac{V}{P} \cdot T_{\text{mold}}, \end{aligned}$$

де:

– W – значення викривлення;

– T_{cool} – температура охолодження форми;

– t_{inject} – час упорскування;

– V/P – перемикання з упорскування на тиск утримування;

– T_{mold} – температура плавління.

Результати. Для проведення експерименту було використано початкові дані з 25 варіантів комбінацій параметрів. Результати експерименту видно в таблиці 1.4.

Таблиця 1.4 – Результати моделювання

Параметр	**Значення**
Температура охолодження (T_{cool}), °C	40
Час упорскування (t_{inject}), s	2.2
V/P, %	97
Температура плавління (T_{mold}), °C	40
Викривлення (Warpage), mm	3.730

Оптимізація дозволила зменшити викривлення з початкового значення 3.776 мм до 3.730 мм, що вказує на ефективність застосування алгоритму PSO.

1.5 Ройовий інтелект для бінарної класифікації трафіку

Системи виявлення вторгнень і системи запобігання вторгненням є найважливішими інструментами захисту від складних мережевих атак, кількість яких постійно зростає. Через брак надійних наборів даних для тестування та валідації, підходи до виявлення вторгнень, засновані на аномаліях, страждають від послідовної та точної еволюції продуктивності. Набір даних CIC-IDS2017 [20] містить безпечні та найсучасніші поширені атаки, які нагадують реальні дані. Він також включає результати аналізу мережевого трафіку за допомогою CICFlowMeter з маркуванням потоків на основі часової мітки, IP-адрес джерела та призначення, портів джерела та призначення, протоколів та атак. Він опублікований Канадським інститутом кібербезпеки у 2017 році, цей набір став еталоном для досліджень з виявлення аномалій та вторгнень. Набір складається з пакетів трафіку, зібраних протягом 5 днів у 8 окремих сесіях. Він представлений у вигляді .csv-файлів, один на кожну сесію, спеціально для досліджень машинного навчання. Велика кількість IDS систем тестується на наборі CIC-IDS2017 перед впровадження в мережі. Набір даних CIC-IDS2017 включає дані про безпечний трафік та трафік наступних атак: DoS/DDoS, Brute Force, Botnet, Web-атаки, Інфільтрація, PortScan та Heartbleed.

1.5.1 Постановка теоретичних завдань

Основним теоретичним завданням є аналіз результатів бінарної класифікації за допомогою методів логістичної регресії (методом SAGA)

та методу опорних векторів, а також аналіз особливостей датасету.

1.5.2 Постановка практичних завдань

Практична частина роботи полягає у бінарній класифікації датасету на записи про нормальний та аномальний трафік. Для цього дані вимагають попередньої обробки такої як очищення, аналіз паттернів, оптимізація та виділення збалансованого набору.

Для перевірки ефективності класифікації за допомогою кожного з алгоритмів результати алгоритмів мають бути зіставлені один з одним, а також результатами класифікації іншими методами. Також для оцінки якості має бути проведено ROC-аналіз

Висновки до розділу 1

У розділі було розглянуто задачі, які потребують покращення їхньої роботи шляхом оптимізації вхідних параметрів. Для кожної з проблем має підбиратися індивідуальний підхід, зважаючи на початкові вимоги до оптимізації. Кожен алгоритм має свої сильні та слабкі сторони. Наприклад, процес виявлення несправностей біореактора вимагає якомога високої точності, бо від цього залежить, наскільки він буде придатним для досліджень. Тому для цього використовують одразу 3 метаевристики такі як ACO-d, DEwPC та CMA-ES, які в сукупності можуть компенсувати недоліки одне одного, які негативно впливають на точність вимірювань: ACO-d має високу точність, але чутливий до шуму; CMA-ES нейтралізує вплив шуму; DEwPC адаптує процес оптимізації до різних обмежень моделі. Проте треба мати на увазі, що в такому випадку оптимізація є ресурсозатратною займатиме багато часу. Тож ефективна оптимізації вимагає детального вивчення вимог та адаптації до кожного окремого випадку.

2 ОГЛЯД НОВИХ РОЙОВИХ ТЕХНІК, ЩО АКТИВНО ЗАСТОСОВУЮТЬ ДЛЯ ВИРІШЕННЯ ЗАВДАНЬ АНАЛІЗУ РІЗНИХ ІНЖЕНЕРНИХ СИСТЕМ

2.1 Опис роботи алгоритму «Півень-Курка-Курка»

2.1.1 Природне явище, яке лежить в основі алгоритму

Базовий алгоритм наведено в [14]. У стародавні часи в селах Китаю зазвичай було прийнято тримати зграю півнів і курок для отримання яєць і вирощування курчат. Якщо кури запліднені їхні яйці тримали в інкубаторі для штучного виведення курчат. В середньому кури несуться близько 21 дня навесні, а потім з‘являються курчата. Селяни випускали півнів, курок і курчат на подвір’я для пошуку їжі вдень. Під час пошуку їжі півні намагалися знайти місце, де найбільше корму, а потім закликали курок до себе. Кури, які вже мали курчат, вели себе дещо агресивно, щоб захистити потомство, і вели їх до місць із кормом, навчаючи їх знаходити їжу самостійно. Півні періодично сношалися з курками задля продовження роду, а потім вони всі разом виховували курчат. Кури-матері постійно пильнували за своїм потомством, закликаючи курчат слідувати за собою, та ставали агресивними у разі наближення будь-якої потенційної загрози, включаючи людей. Загалом півні, курки та курчата знаходили їжу, керуючись власними інстинктами та вродженому спадковому потенціалу. Проте зі збільшенням чисельності птахів їхні поведінкові моделі під час пошуку їжі дещо модифікувалися.

2.1.2 Дослідження поведінки півнів під час пошуку їжі

Через декілька місяців самці курей дорослішають і починають прагнути до розмноження. Тому вони намагаються знайти якомога

більше їжі, щоб покликати туди курок та скористатися можливістю запліднення. Якщо знайдено так місце з їжею, то вони самі харчуються, викопують їжу їжу дзьобом. Якщо ж не знайдено, вони шукають їжу поруч або переміщаються в інше місце, як показано на рисунку 2.1.

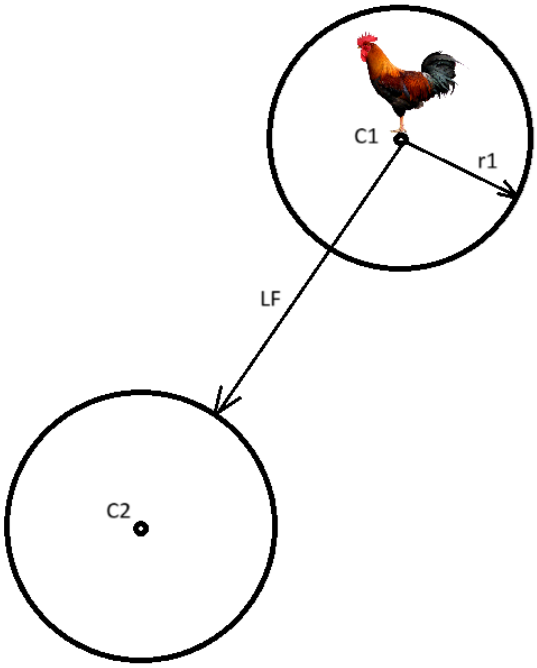


Рисунок 2.1 – Схематичне зображення півня при пошуку їжі

Півень, знайшовши їжу, шукає більше поживи поблизу. Математично це можна описати як випадковий рух від початкової точки:

$$x_i(t+1) = x_i(t) + dance \cdot r_1, \tag{2.1}$$

де $x_i(t+1)$ та $x_i(t)$ – положення i-го півня на ітерації $t+1$ та поточній ітерації відповідно, $dance$ – це постійний параметр, який у цій роботі приймає значення 0.8, а r_1 – випадковий вектор з рівномірним розподілом.

Згідно з рівнянням (2.1), півні детально досліджують місце навколо

поточного положення. Кілька півнів у зграї розходяться на різні території в пошуках їжі. Коли півень не знаходить їжі, він випадково переміщується в нове місце за принципом польоту Леві.

Польоти Леві описуються поєднанням коротких і довгих переміщень, що дає змогу птахам досліджувати територію навколо. Математично це можна виразити так:

$$x_i(t+1) = x_b(t) \cdot LF(D), \tag{2.2}$$

де $x_b(t)$ – найкраще положення на поточній ітерації, а $LF(D)$ – D-вимірний випадковий вектор у (2.2), який підпорядковується:

$$LF(D) = 0.01 \cdot \frac{\mu}{\left|\nu^{\frac{1}{\beta}}\right|},$$

де μ та ν є глобальним середнім значенням і стандартним відхиленням у нормальному розподілі з $\mu = 0$, а

$$\sigma_u = \left[\frac{\Gamma(1+\beta) \cdot \sin\left(\frac{\pi\beta}{2}\right)}{\Gamma\left(\frac{1+\beta}{2}\right) \cdot \beta \cdot 2^{\frac{\beta-1}{2}}}\right]^{\frac{1}{\beta}},$$

де Γ позначає стандартну гамма-функцію в математиці, а β – стала, що дорівнює 1,5.

2.1.3 Дослідження поведінки курей під час пошуку їжі

У разі сусідства півнів і курей, кури можуть слідувати за ними у пошуках їжі. Проте поведінка курей різниться: деякі продовжують триматися за одним певним півнем, інші можуть переміщатися самостійно або з курчатами, обираючи різних півнів.

Півні намагаються зібрати більше курей навколо себе, що підвищує їхні шанси на запліднення. За наявності достатньої кількості їжі кури групуються біля найкращого півня, безперервно досліджуючи

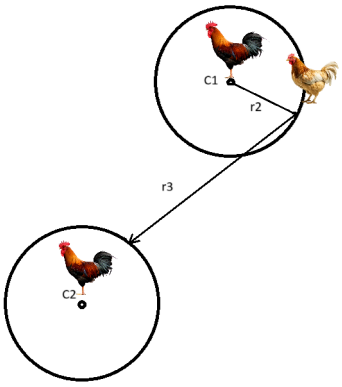


Рисунок 2.2 – Схематичне зображення півня при пошуку їжі

навколишній простір у пошуках оптимальних умов. Цю поведінку можна описати так:

$$x_i(t+1) = x_i(t) + a \cdot r_2 \left[x_b(t) - x_i(t)\right], a = 2,$$

де *maxIter* позначає максимальну кількість ітерацій. r_2 є випадковим числом в інтервалі від 0 до 1.

Кури-матері шукають їжу для курчат, уникаючи зайвих контактів. За нестачі їжі вони можуть слідувати за іншим півнем:

$$x_i(t+1) = x_c(t) + a \cdot r_3 \left[x_c(t) - x_i(t)\right],$$

де $x_c(t)$ представляє випадково вибраного кандидата серед півнів.

2.1.4 Дослідження поведінки курчат під час пошуку їжі

Курчата зазвичай використовують будь-яку можливість знайти їжу. Вони настільки ненажерливі, що можуть клювати здобич, яка значно більша за них. На рисунку 2.3 показано, як курчата слідують за курками, півнями або спираються на власний досвід пошуку їжі.

Можна виділити чотири шляху пошуку. Більшість часу курчата

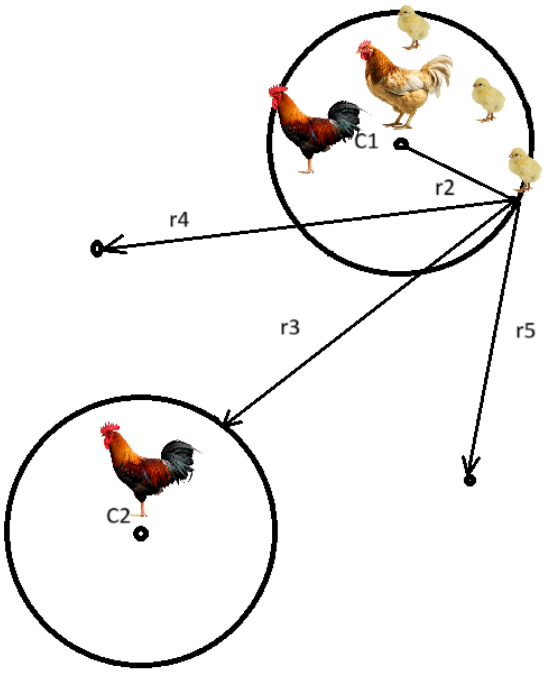


Рисунок 2.3 – Схематичне зображення півня при пошуку їжі

слідували за своїми матерями і випадково вибирали, куди йти:

$$x_i(t+1) = x_h(t) + a \cdot r_4[x_h(t) - x_i(t)].$$

Іноді курчата не впізнавали свою матір. Тому $x_h(t)$ був випадково вибраним кандидатом серед курей. Іноді курчата крокували своїм власним шляхом і починали свій пошук там з невеликою ймовірністю p_1:

$$x_i(t+1) = r_5(ub - lb) + lb,$$

де $[lb, ub]$ – область визначення даної проблеми, r_5 – інше випадкове число, взяте з гаусівського розподілу.

Отриманий курчатами досвід вже дозволяв їм уникати найгірших позицій або загрозливих цілей, як описано нижче:

$$x_i(t+1) = x_i(t) + a \cdot r_6[x_w(t) - x_i(t)],$$

де $x_w(t)$ представляє найгіршу позицію на поточній ітерації.

2.1.5 Результати роботи алгоритму

Алгоритм CHC тестували на 5 функціях з одним мінімумом. У таблиці 2.1 наведено їх. Результат було порівняно з такими методами як ALO, EO, GWO, MOA, PSO, SCA та WOA.

Таблиця 2.1 – Тестові функції

№	Назва функції	Формула	Розмірність	Домен
F1	Ackley 1	$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$	10	[−100,100]
F2	Exponential	$f(x) = \exp\left(\sum_{i=1}^{n} x_i^2\right)$	10	[−3,3]
F3	Powell Sum	$f(x) = \sum_{i=1}^{n} \lvert x_i\rvert^{i+1}$	10	[−1,1]
F4	Sargan	$f(x) = \sum_{i=1}^{n}\left(nx_i^2 + \sum_{j=1}^{n}(x_i x_j)\right)$	10	[−100,100]
F5	Sphere	$f(x) = \sum_{i=1}^{n} x_i^2$	10	[−100,100]

Розмір популяції становив на 50 особин, а кінцеві результати усереднювалися за 100 експериментами методом Монте-Карло.

Fcn	Items	CHC	ALO	EO	GWO	MOA	PSO	SCA	WOA
	best	-4.44089E-16	0	0	0	0	0	0	0
	worst	**3.10862E-15**	4.493038381	5.63993E-14	0.587499278	1.972709423	0.587288391	0.09935742	1.48392E-11
F1	median	**0**	0	0	0	0	0	0	0
	mean	**1.28786E-16**	0.320937809	3.6815E-15	0.002937497	0.149649509	0.057408594	0.001562399	1.09517E-13
	std	**7.32343E-16**	0.847491335	9.76028E-15	0.041542472	0.385225768	0.150277459	0.00882281	1.07945E-12
	best	**0**	0	0	0	0	0	0	0
	worst	**0**	2.42006E-08	1.11022E-16	2.22045E-16	2.70787E-10	1.25833E-08	9.77745E-06	1.11022E-16
F2	median	**0**	0	0	0	0	0	0	0
	mean	**0**	6.49398E-10	5.55112E-19	1.72085E-17	1.51399E-12	1.90038E-10	8.12503E-08	2.77556E-18
	std	**0**	2.57494E-09	7.85046E-18	4.46558E-17	1.92141E-11	1.07264E-09	7.0983E-07	1.73768E-17
	best	**0**	0	0	0	0	0	0	0
	worst	**1.3644E-188**	0.002882117	7.0561E-41	2.18235E-37	6.73138E-18	6.41762E-09	1.51012E-08	5.52118E-33
F3	median	**0**	0	0	0	0	0	0	0
	mean	**8.8203E-191**	0.000153972	4.79006E-43	1.47413E-39	4.07949E-20	7.06394E-11	1.13902E-10	4.25081E-35
	std	**0**	0.000480291	5.02111E-42	1.57042E-38	4.78651E-19	4.95921E-10	1.10848E-09	4.16179E-34
	best	**0**	0	0	0	0	0	0	0
	worst	**1.8306E-139**	0.006403136	3.44724E-23	4.03977E-19	0.007121995	9.49522E-06	0.949028633	8.0982E-08
F4	median	**0**	0	0	0	0	0	0	0
	mean	**2.287E-141**	0.0001162	5.41733E-25	6.94682E-21	3.69026E-05	1.66648E-07	0.006584187	7.57315E-10
	std	**1.7639E-140**	0.00055635	3.32007E-24	3.89627E-20	0.000503562	8.9812E-07	0.067438429	6.65019E-09
	best	**0**	0	0	0	0	0	0	0
	worst	**8.9056E-146**	1.55407E-05	2.40337E-27	7.65955E-23	2.10377E-07	4.33227E-08	0.011284106	2.2145E-31
F5	median	**0**	0	0	0	0	0	0	0
	mean	**4.4781E-148**	4.21791E-07	2.08527E-29	1.92435E-24	2.96301E-09	5.13357E-10	0.000108135	2.39664E-33
	std	**6.297E-147**	1.61737E-06	1.74949E-28	9.53203E-24	2.11306E-08	3.28124E-09	0.000838642	1.80794E-32

Рисунок 2.4 – Результати на F1–F5

На рисунку 2.4 можна побачити результати для кожного з досліджуваних алгоритмів. CHC продемонстрував кращу продуктивність на всіх унімодальних тестових функціях. Порівняно з іншими алгоритмами, CHC забезпечує нижчі середні значення похибки.

2.2 Ройовий алгоритм тигрового жука аналізу глобальних розв'язків задач в екстремальній постановці

2.2.1 Природне явище, яке лежить в основі алгоритму

Базовий алгоритм наведено в [24]. Тигровий жук вважається найдикішим видом у природі через природний хист до полювання. Під час дослідження помешкання жертви він одразу її при виді атакує та може навіть вбити жертву за допомогою кігтів. На рисунку 2.5 зображено

будову жука.

Рисунок 2.5 – Тигровий жук

Тигровий жук має унікальний механізм полювання, це показано на рисунку 2.6. Вони риють нори та намагаються загнати жертву туди.

Рисунок 2.6 – Нори, в яких ховаються тигрові жуки для полювання

Личинки жуків також беруть участь в полюванні. Коли жертва наближається, вони відкривають вхід у норі та затягують здобич.

2.2.2 Моделювання алгоритму TBO

Першим етапом є створення початкової популяції. Вона генерується випадковим чином у просторі (2.3):

$$TB_i^j = L + (U - L) \cdot \text{rand}, \tag{2.3}$$

де TB_i^j – це j-та координата i-го рішення, що апроксимує ТВ, L – нижня межа у вимірі, U – верхня межа, а `rand` – випадкове число від 0 до 1.

Початкова популяція жуків у першій ітерації може бути сформована за допомогою матриці (2.4):

$$P = \begin{bmatrix} TB_1^1 & TB_1^2 & \dots & TB_1^D \\ TB_2^1 & TB_2^2 & \dots & TB_2^D \\ \vdots & \vdots & \ddots & \vdots \\ TB_n^1 & TB_n^2 & \dots & TB_n^D \end{bmatrix}, \tag{2.4}$$

де n – кількість жуків, а D – розмірність кожного рішення.

Кожна особина оцінюється за допомогою цільової функції F. Відповідну оцінку популяції можна подати як:

$$f(P) = \begin{bmatrix} f(TB_1^1 & TB_1^2 & \dots & TB_1^D) \\ f(TB_2^1 & TB_2^2 & \dots & TB_2^D) \\ \vdots & & & \\ f(TB_n^1 & TB_n^2 & \dots & TB_n^D) \end{bmatrix}. \tag{2.5}$$

Далі йде етап оцінювання якості нори. Стратегія полягає в створенні нір біля найбільш оптимальних рішень за допомогою рівняння (2.6). Це створює різні варіанти рішень. Алгоритм перевіряє, які з них дають кращі результати. Хороші рішення показують, в яких напрямках варто шукати далі.

$$H(TB_i) = \left[1 - \exp\left(\frac{f(TB_i)}{f(W(t))}\right)\right] \cdot H_m, \tag{2.6}$$

де:

– $H(TB_i)$ – кількість нір поблизу TB_i;

– H_m – максимальна кількість нір, які копає жук на ділянці;

– $f(TB_i)$ – значення або придатність TB_i;

– $f(W(t))$ – найкраще рішення в поточній ітерації t.

H змінюється від значення, що відповідає найкращому TB, до найгіршого, тож більшість нір створюються біля оптимального розв'язку для кращого вивчення області.

Далі жук розташовується в норі. Кожен жук може викопати $H(TB_i)$ нір навколо себе, згідно з рівнянням (2.6) та залежно від оцінки якості. Кожен жук може створювати нові рішення, копаючи нори, відкладаючи в них личинки. Нори спочатку розташовуються з більшою стандартною девіацією відносно центральної точки, а потім перехід до локальної стратегії пошуку задля зменшення похибок.

Для зменшення розбіжності у стандартному відхиленні розподілу рішень пропонується застосування арктангенсової функції відстані (2.7):

$$SD(t) = SD_0 - \left|\frac{p}{\pi} \cdot \arctan\left(\frac{p}{\pi} \cdot t\right)\right|, \tag{2.7}$$

де:

– p – коефіцієнт контролю зменшення нір та їх розподіл;

– SD_0 – початкове стандартне відхилення;

– t – лічильник ітерацій.

На рисунку 2.7 показано розподіл стандартного відхилення для трьох значень p. Збільшення p розширює діапазон відхилення SD до $[0.8, 1]$. При $p = 1$, SD варіюється в межах $[0.5, 0.9]$, а при $p = 0.5$ – у межах $[0.1, 0.65]$.

Щоб викопати нору навколо заданої позиції, можна використати рівняння (2.8):

$$TB_{\text{new}} = \begin{cases} TB_{\text{old}} + R \cdot SD(t) \cdot \text{rand}, & r \leqslant 0.5 \\ TB_{\text{old}} - R \cdot SD(t) \cdot \text{rand}, & r > 0.5 \end{cases}$$

де:

– R – поле розмноження рішення, випадкове число в інтервалі $[1, 2]$;

– `rand` – випадкове число з інтервалу $[0, 1]$;

– r – додаткове випадкове число з інтервалу $[0, 1]$, яке визначає, чи рішення буде розміщене зліва або справа у відповідному вимірі.

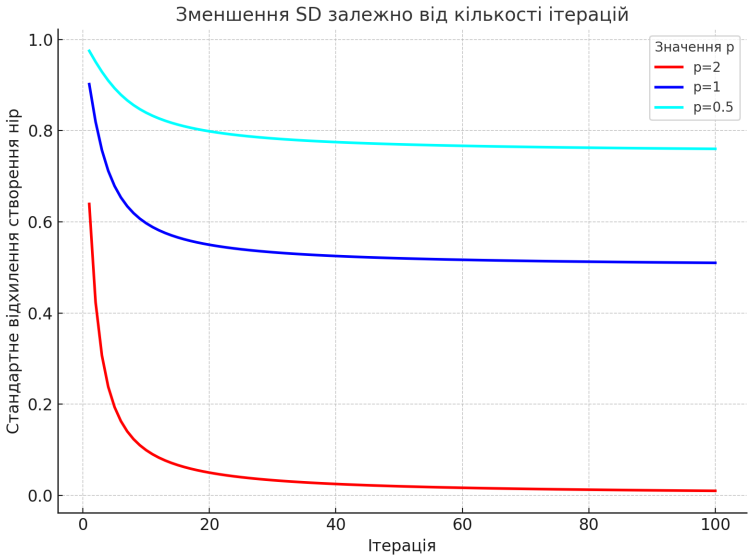


Рисунок 2.7 – Розподіл зменшення ніш для переслідування здобичі відповідно до кількості ітерацій алгоритму

Отвори з личинками можуть вміймати здобич залежно від її якості та розташування. Деякі отвори через це не використовують або знищують, а розв'язки видаляють через невдале полювання. Ймовірність виживання розв'язку задається рівнянням (2.8):

$$p(TB_i) = 1 - \frac{1}{\sum_{i=1}^{n} \frac{f(TB_i)}{f(W(t))}} \times \frac{f(TB_i)}{f(W(t))}. \tag{2.8}$$

У рівнянні (2.8) $p(TB_i)$ – ймовірність виживання розв'язку. Випадкове число r з інтервалу $[0, 1]$ використовується для визначення того, чи буде розв'язок TB_i збережено (2.8). P – популяція розв'язків:

$$P = \begin{cases} P + TB_i, & \text{якщо } r \leqslant p(TB_i), \\ P - TB_i, & \text{якщо } r > p(TB_i). \end{cases} \tag{2.9}$$

Якщо $r \leqslant p(TB_i)$, то отвір і жук, який у ньому знаходиться, залишаються на своєму поточному місці. Інакше, якщо $r > p(TB_i)$, то

відповідний жук не буде врахований у подальших обчисленнях.

Останніми етапами є пошук розв‘язку у місцях, багатих на комах та схрещування. Рівняння (2.10) демонструє пошук у регіоні, який має найвищу ймовірність отримання оптимального рішення:

$$TB_i^j = \alpha \cdot TB_i^j + \beta \cdot (B(t) - \bar{B}) \cdot \text{rand}. \tag{2.10}$$

Тут у (2.10) $B(t)$ – координати найкращого жука, $\bar{B}$ – середнє значення розв'язків, α – коефіцієнт зменшення впливу поточної позиції, а β – коефіцієнт посилення впливу області пошуку. Вагові коефіцієнти α та β можна обчислити за допомогою рівнянь (2.11) та (2.12). Рисунок 2.8 ілюструє зміну цих двох параметрів з плином часу – зменшення α та збільшення β:

$$\alpha = 1 - \left(\frac{t}{\text{MaxT}}\right)^2, \tag{2.11}$$

$$\beta = 1 - \alpha, \tag{2.12}$$

де MaxT – максимальна кількість ітерацій алгоритму тигрового жука, а t – поточна ітерація.

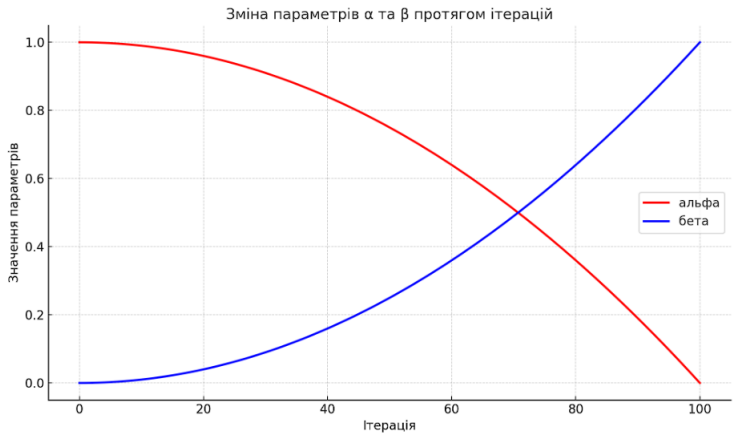


Рисунок 2.8 – Зменшення та збільшення параметрів α та β

У кінці α зменшується, знижуючи вплив поточного розв'язку, а β

збільшується, що розширює область дослідження між розв'язками.

У пошуку партнера TB_i рухається у напрямку випадково обраних із популяції TB_l та TB_k. Схрещування сприяє різноманіттю популяції розв'язків та ефективнішому дослідженню простору. Рівняння (2.13) виражає напрямок руху TB_i у напрямку двох інших жуків

$$TB_i^j = TB_i^j + (TB_k^j - TB_l^j) \cdot \text{rand}. \tag{2.13}$$

Вибір двох жуків TB_k та TB_l, за якими слідує TB_i впливає на поведінку пошуку та запобігає застряганню в локальному оптимумі.

2.2.3 Результати роботи алгоритму

Алгоритм ТВО тестували на мультимодальних та унімодальних функціях. У таблицях функції оцінювання, їхні області визначення оптимальні значення. У дослідженні було виміряно середню обчислювальна похибка, стандартне відхилення та рейтинг алгоритму.

У таблиці 2.2 можна бачити ці функції. Було оцінено ефективність різних алгоритмів, включаючи ТВО, на унімодальних функціях (F1–F7). Системна оцінка підкреслює точність та стійкість алгоритму ТВО, що підтверджується його найвищими рейтингами та найменшими похибками, демонструючи важливість цих метрик для виявлення можливостей алгоритмів. Мультимодальні функції (F8–F13) містять багато локальних мінімумів, що ускладнює процес оптимізації.

Таблиця 2.2 – Тестові функції F1–F13

№	Назва функції	Формула	Розмірність	Домен
F1	Sphere	$f(x) = \sum_{i=1}^{n} x_i^2$	30	$[-100,100]^n$
F2	Schwefel 2.22	$f(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	30	$[-10,10]^n$
F3	Schwefel 1.2	$f(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$	30	$[-100,100]^n$
F4	Schwefel 2.21	$f(x) = \max_{1 \leqslant i \leqslant n} \lvert x_i \rvert$	30	$[-100,100]^n$
F5	Rosenbrock	$f(x) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \right]$	30	$[-30,30]^n$
F6	Step	$f(x) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2$	30	$[-100,100]^n$
F7	Quartic (з шумом)	$f(x) = \sum_{i=1}^{n} i x_i^4 + \text{random}[0,1]$	30	$[-1.28,1.28]^n$
F8	Schwefel	$f(x) = -\sum_{i=1}^{n} x_i \sin(\sqrt{\lvert x_i \rvert})$	30	$[-500,500]^n$
F9	Rastrigin	$f(x) = \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) + 10 \right)$	30	$[-5.12,5.12]^n$
F10	Ackley	$f(x) = -20 \exp\left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2} \right) - \exp\left(\frac{1}{n} \sum_{i=1}^{n} \cos(2\pi x_i) \right) + 20 + e$	30	$[-32,32]^n$
F11	Griewank	$f(x) = \frac{1}{4000} \sum_{i=1}^{n} (x_i - 100)^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i - 100}{\sqrt{i}} \right) + 1$	30	$[-600,600]^n$
F12	Penalized 1	$f(x) = \frac{\pi}{n} \left[10\sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2 \left(1 + 10\sin^2(\pi y_{i+1})\right) + (y_n - 1)^2 \right] + \sum_{i=1}^{n} u(x_i,10,100,4)$	30	$[-50,50]^n$
F13	Penalized 2	$f(x) = 0.1 \left\{ \sin^2(3\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)^2 \left(1 + \sin^2(3\pi x_{i+1})\right) + (x_n - 1)^2 \left(1 + \sin^2(2\pi x_n)\right) \right\} + \sum_{i=1}^{n} u(x_i,5,10,4)$	30	$[-50,50]^n$

Зі збільшенням розмірності простору пошуку кількість локальних екстремумів зростає експоненціально (рисунок 2.9, рисунок 2.10).

Тому такі функції є ефективним інструментом для оцінки здатності алгоритмів до дослідження простору рішень.

F	Metric	TBO	BAT	PSO	DE	FA	BOA	GWO	WHO	BWO	ASO
F1	AVG	**1.46E−14**	1.06E−4	5.23E−4	1.0069	5.82564	1.22E−4	7.33E−8	2.69E−7	1.55E−8	3.45E−12
	STD	**4.56E−7**	2.88E−4	7.55E−4	0.8753	2.99856	8.23E−3	1.72E−4	2.95E−6	1.12E−6	8.28E−6
	RANK	**1.27**	6.85	6.92	7.69	9.52	5.86	4.53	4.87	3.17	2.14
F2	AVG	**2.86E−12**	9.5682	2.83E−4	1.8536	5.7523	0.02569	3.91E−9	5.29E−11	6.33E−10	5.12E−10
	STD	**1.06E−9**	5.1568	3.72E−4	0.7853	0.5692	4.11E−3	6.03E−5	4.39E−8	6.54E−6	5.07E−8
	RANK	**1.12**	9.34	5.86	7.82	8.16	6.72	4.53	2.18	3.78	2.74
F3	AVG	**0.007625**	15.6952	185.1475	28.8536	36.0263	0.00898	0.09025	0.09920	9.68531	83.6954
	STD	0.563227	5.00258	120.4269	5.75863	12.6359	0.086473	0.008632	**0.001563**	0.96635	52.6986
	RANK	**2.19**	6.97	8.23	7.53	8.07	2.84	3.77	4.18	6.42	7.76
F4	AVG	3.83E−09	1.003695	12.14402	1.230064	1.00691	0.008631	2.39E−06	**6.56E−10**	7.76E−09	2.65E−08
	STD	6.73E−05	0.058625	10.75263	0.52392	0.22653	7.49E−03	8.23E−03	6.55E−07	**2.56E−07**	9.16E−06
	RANK	**1.16**	5.89	7.53	6.75	5.93	4.55	3.67	1.28	1.83	2.65
F5	AVG	**21.823**	121.6963	113.8531	189.3269	82.0983	160.61	26.0334	24.14362	28.8536	22.5863
	STD	**0.40369**	132.4621	136.8314	144.112	156.057	120.8543	2.22361	0.4568	0.68532	0.86953
	RANK	**2.26**	6.23	5.79	7.06	5.36	6.84	3.88	2.92	3.76	2.73
F6	AVG	**1.08E−19**	12.02631	0.09522	21.0961	26.48632	1.50265	6.59E−16	1.66E−18	2.56E−18	2.56E−19
	STD	9.23E−11	0.88237	0.002387	1.44603	5.63409	0.02365	1.77E−11	**1.24E−12**	5.88E−12	6.74E−11
	RANK	2.34	6.16	4.88	7.18	7.42	5.56	3.82	2.77	2.54	**2.18**
F7	AVG	**0.01153**	0.039856	0.077362	0.079362	0.01169	0.076329	0.03056	0.02963	0.02632	0.039526
	STD	**0.00914**	0.086362	0.016026	0.013652	0.86392	0.028623	0.086234	0.04436	0.01526	0.014269
	RANK	**2.62**	4.66	5.34	5.68	2.97	5.19	4.16	3.25	3.23	4.29

Рисунок 2.9 – Порівняння результатів на унімодальних функціях

F	Metric	TBO	BAT	PSO	DE	FA	BOA	GWO	WHO	BWO	ASO
F8	AVG	**-9273.23**	−5680.93	−5207.44	−5032.98	−5414.83	−5583.14	−5672.37	−7521.66	−7521.66	**−7432.18**
	STD	286.265	632.11	506.16	703.18	563.53	486.07	533.98	388.16	488.98	**157.1483**
	RANK	**1.63**	3.67	6.12	6.41	5.86	5.69	3.88	2.24	2.39	**1.88**
F9	AVG	5.03E−16	4.03E-4	3.37E−9	1.44E-3	3.93E-6	6.17E-12	2.57E−10	1.86E−10	4.12E−16	**1.47E−8**
	STD	**4.08E−6**	8.16E−02	7.91E−3	5.53E−03	1.3621	6.09E−04	5.92E−3	9.11E−4	3.76E−5	**5.66E−4**
	RANK	**1.22**	6.69	4.37	6.82	5.83	3.72	3.96	3.92	1.26	**4.64**
F10	AVG	**4.63E−13**	0.00439	5.79E−7	0.05369	0.00892	6.33E−8	6.81E−9	3.93E−9	9.11E−6	**2.95E−11**
	STD	6.73E−5	0.08615	2.64E−4	0.80261	0.94627	9.06E−5	**6.86E−8**	7.44E−7	8.16E−5	**4.69E−7**
	RANK	**1.98**	5.83	4.65	6.26	6.18	3.83	2.76	3.18	4.84	**2.17**
F11	AVG	**8.94E−16**	3.64E−6	8.94E−10	0.01793	364E−8	0.01683	8.23E−9	8.94E−15	8.54E−5	**9.16E−9**
	STD	**6.18E−10**	8.76E−3	8.63E−5	0.05636	1.29E−5	0.96315	1.92E−6	**3.14E−12**	5.11E−3	**2.08E−6**
	RANK	**1.67**	3.64	2.09	5.66	2.83	6.83	2.48	1.75	4.09	**2.44**
F12	AVG	**3.78E−24**	0.036982	0.224573	1.29836	9.01E−9	6.08E−11	2.78E−11	2.44E−22	6.08E−20	***5.14E−23***
	STD	2.44E−16	0.864231	0.510369	0.36952	5.69E−5	2.59E−6	6.11E−8	**5.02E−18**	2.16E−16	**6.54E−17**
	RANK	1.86	5.66	6.16	6.73	4.84	4.11	3.86	2.43	3.42	**1.74**
F13	AVG	**8.16E−23**	0.039206	0.18693	0.33582	0.07362	6.86E−15	5.88E−16	8.32E−21	**8.92E−23**	**6.16E−22**
	STD	8.41E−12	0.001755	0.89032	0.77361	0.06695	2.08E−11	6.94E−11	**5.16E−20**	9.74E−11	**6.67E−15**
	RANK	1.48	4.63	5.38	6.93	4.81	3.49	3.23	2.93	**1.36**	**2.21**

Рисунок 2.10 – Порівняння результатів на мультимодальних функціях

2.3 Багатостратегічний ройовий алгоритм пошуку медуз

2.3.1 Короткий опис алгоритму

Базовий алгоритм наведено в [8]. Алгоритм пошуку медуз імітує поведінку медуз у пошуку їжі. У JS медузи можуть рухатися або за течією океану, або усередині групи. Завдяки контролю за часом медузи можуть перемикатися між цими режимами. В покращеній версії алгоритму додано синусо-косинусні фактори навчання, оператор локального виходу з оптимуму та різні стратегії навчання.

2.3.2 Математичне представлення алгоритму

Цю модель можна описати описати як велику кількість поживної їжі в океанічній течії, навколо якої групуються медузи. Медузи слідують за течією, потім прямують всередину групи.

Під час групового переміщення медузи здійснюють пасивний (тип A) та активний (тип B) рух. Для визначення необхідного типу застосовується принцип контролю часу, що регулює перехід між типами.

Першим етапом алгоритму є ініціалізація популяції. Для підвищення різноманітності кандидатів алгоритм використовує логістичне відображення при ініціалізації, це задається наступним рівнянням:

$$P_{i+1} = \eta P_i(1 - P_i),$$

де $\eta = 4$. Значення логістичного хаосу кандидата позначається як P_i, а початкове значення – як P_0, воно має задовольняти:

$$P_0 \in (0, 1), \quad P_0 \notin \{0, 0.25, 0.5, 0.75, 1\}.$$

Це дозволяє уникнути вироджених початкових станів та забезпечує хаотичне, але кероване покриття простору пошуку. Для кожного кандидата напрямок від його поточної позиції до оптимальної можна назвати напрямком течії (*Direction*), що можна описати рівнянням:

$$\overrightarrow{\text{Direction}} = \frac{1}{N}\sum \overrightarrow{\text{Direction}_i} = \frac{1}{N}\sum (P^* - e_c P_i) = \\ = P^* - e_c \frac{\sum P_i}{N} = P^* - e_c \mu.$$

Нехай $df = e_c\mu$, тоді напрямок можна скоротити таким чином:

$$\overrightarrow{\text{Direction}} = P^* - df, \tag{2.14}$$

де N, e_c і μ – кількість кандидатів, коефіцієнт привабливості та середня

позиція всіх медуз відповідно. P^* – найкраща позиція, а df – різниця оптимального та середнього значень.

Популяція має нормальний розподіл, тож відстань поблизу середнього значення $\pm\beta\sigma$ може охоплювати всіх кандидатів, тоді df можна звести до:

$$df = \beta \times r_1 \times \mu. \tag{2.15}$$

Тут $e_c = \beta \times r_1$, $r_1 = \text{rand}(0, 1)$. Таким чином, математичне рівняння (2.14) для океанської течії можна описати рівнянням (2.16):

$$\overrightarrow{\text{Direction}} = P^* - \beta \times r_1 \times \mu. \tag{2.16}$$

Тепер рівняння для кандидатних осіб можна подати як (2.17):

$$P_i(t+1) = P_i(t) + r_2 \times \overrightarrow{\text{Direction}}. \tag{2.17}$$

Об'єднавши рівняння (2.16) та рівняння (2.17) можна перетворити на:

$$P_i(t+1) = P_i(t) + r_2 \times (P^* - \beta \times r_1) \times \mu. \tag{2.18}$$

Тут: $\beta > 0$, $\beta = 3$, $r_2 = \text{rand}(0, 1)$. У рої є руху A і B, між якими кандидати перемикаються. Спочатку медузи формують пасивний рій, потім переходять до активного.

1) **Рух типу A**: у пасивному русі кандидат рухається навколо поточної позиції, і його положення може оновлюватися за рівнянням 2.19:

$$P_i(t+1) = P_i(t) + \gamma \times r_3 \times (Ub - Lb), \tag{2.19}$$

де Ub і Lb – межі пошуку, $\gamma = 0.1$ – коефіцієнт руху, $r_3 = \text{rand}(0,1)$.

2) **Рух типу B**: у активному русі випадково вибирається кандидат j. Якщо у P_j більше їжі, ніж у P_i, тоді P_i рухається до P_j. Інакше – віддаляється.

$$P_i(t+1) = P_i(t) + \overrightarrow{step},$$

де

$$\overrightarrow{step} = \text{rand}(0,1) \times \overrightarrow{DDirection}$$

і

$$\overrightarrow{DDirection} = \begin{cases} P_j(t) - P_i(t), & \text{якщо } f(P_i) \geqslant f(P_j), \\ P_i(t) - P_j(t), & \text{якщо } f(P_i) < f(P_j). \end{cases} \tag{2.20}$$

2.3.3 Покращений алгоритм пошуку медузи

Алгоритм пошуку медузи має низьку точність та ймовірність застрягання в локальних екстремумах. Щоб позбутися їх, є три стратегії вдосконалення:

1) введення синусоїдальних та косинусоїдальних факторів навчання;

2) введення оператора локального виходу;

3) застосування навчання на основі опозиції та квазі-опозиції.

Фактори ω_1 та ω_2 дозволяють медузі навчатися не лише від випадкової особини, але й від найкращої в популяції:

$$\omega_1 = 2 \cdot \sin\left[\left(1 - \frac{t}{T}\right) \cdot \frac{\pi}{2}\right], \omega_2 = 2 \cdot \cos\left[\left(1 - \frac{t}{T}\right) \cdot \frac{\pi}{2}\right].$$

Оновлення позиції для руху типу B відбувається за формулою:

$$P_i(t+1) = \omega_1 \cdot (P_i(t) + \vec{\text{step}}) + \omega_2 \cdot (P^* - P_i(t)),$$

де $\vec{\text{step}}$ – крок згідно з рівняннями 2.20, P^* – найкраща позиція.

Синусоїдальні та косинусоїдальні коефіцієнти можуть підвищити здатність до пошуку локальних екстремумів, але знизити до глобальних.

Оператор локального виходу – це оператор пошуку, який має знаходить нові області пошуку та покращує експлуатацію в різних умовах. Він застосовується, коли медуза прямує за океанською течією. Це

допомагає уникати локальних оптимальних рішень.

LEO формує альтернативне рішення $P_{\text{LEO}}(t)$ до поточного $P_i(t+1)$, що дозволяє досліджувати простір навколо оптимального рішення.

Якщо rand < 0.5:

$$P_{\text{LEO}}(t) = P_i(t+1) + f_1(u_1 P^* - u_2 P_k(t)) + f_2 \rho_1 u_3 (P_{2i}(t) - \\ - P_{1i}(t)) + \frac{u_2(Pr_1(t) - Pr_2(t))}{2},$$

$$P_i(t+1) = P_{\text{LEO}}(t),$$

інакше:

$$P_{\text{LEO}}(t) = P^* + f_1(u_1 P^* - u_2 P_k(t)) + f_2 \rho_1 u_3 (P_{2i}(t) - P_{1i}(t)) + \\ + \frac{u_2(Pr_1(t) - Pr_2(t))}{2},$$

$$P_i(t+1) = P_{\text{LEO}}(t),$$

де:

$$f_1 \sim \text{Uniform}[-1, 1], \quad f_2 \sim \mathcal{N}(0,1),$$
$$u_1 = L_1 \cdot 2 \cdot R_1 + (1 - L_1),$$
$$u_2 = L_1 \cdot R_2 + (1 - L_1),$$
$$u_3 = L_1 \cdot R_3 + (1 - L_1).$$

L_1 – бінарний параметр:

$$L_1 = \begin{cases} 1, & \text{якщо } \mu_1 < 0.5 \\ 0, & \text{інакше} \end{cases}, \quad \text{де } \mu_1 \sim \text{Uniform}[0, 1].$$

Коефіцієнт адаптації ρ_1:

$$\rho_1 = 2 \cdot \text{rand}(0,1) \cdot \alpha - \alpha,$$
$$\alpha = \left| \chi \cdot \sin\left(\frac{3\pi}{2} + \sin(\beta \cdot \frac{3\pi}{2})\right)\right|,$$
$$\chi = \chi_{\min} + (\chi_{\max} - \chi_{\min}) \cdot \left(1 - \left(\frac{t}{T}\right)^3\right)^2,$$

де $\chi_{\min} = 0.2$, $\chi_{\max} = 1.2$. Випадкові кандидати:

$$P_{1i}(t) = L_b + R_4 \cdot (U_b - L_b),$$
$$P_{2i}(t) = L_b + R_5 \cdot (U_b - L_b),$$

де $R_4 = \text{rand}(1, D)$, $R_5 = \text{rand}(1, D)$ Формула для $P_k(t)$:

$$P_k(t) = L_2 \cdot P_p(t) + (1 - L_2) \cdot P_{\text{rand}},$$
$$P_{\text{rand}} = L_b + R_6 \cdot (U_b - L_b),$$

P_p, $p \in \{1, 2, ..., N\}$ – довільне рішення; $R_6 = \text{rand}(0, 1)$ L_2 – бінарний параметр:

$$L_2 = \begin{cases} 1, & \text{якщо } \mu_2 < 0.5, \\ 0, & \text{інакше,} \end{cases} \quad \text{де } \mu_2 \sim \text{rand}(0,1).$$

Методи навчання на основі протилежностей та квазі-протилежного введено для підвищення різноманітності кандидатних особин. Вони застосовуються після завершення дослідження та експлуатації.

Застосовуючи стратегії OBL та QOL для i-ї кандидатної особини P_i у поточній популяції, можна отримати:

– протилежне рішення $P_i^e = (P_{i1}^e, P_{i2}^e, \ldots, P_{iD}^e)$,

– майже-протилежне рішення $\hat{P}_i = (\hat{P}_{i1}, \hat{P}_{i2}, \ldots, \hat{P}_{iD})$.

Компонентні формули для цих рішень подано нижче:

$$P_{id}^e = Lb_d + Ub_d - P_i^d,$$

$$\hat{P}_{id} = \text{rand}\left(\frac{Lb_d + Ub_d}{2}, Lb_d + Ub_d - P_i^d\right),$$

де P_i^d – d-а координата i-ї особини, Lb_d та Ub_d – нижня та верхня межі простору рішень відповідно для d-го виміру.

Таким чином, оновлене положення i-ї медузи P_i^{new} визначається за формулою:

$$P_i^{\text{new}} = \begin{cases} P_i^e, & \text{якщо rand} < p \\ \hat{P}_i, & \text{інакше} \end{cases}, \quad i = 1, 2, \ldots, N, \tag{2.21}$$

де p – ймовірність вибору, зазвичай $p = 0.5$.

На основі рівняння генеруються N нових особин, обчислюються значення функції. Після цього всі кандидати сортуються, і з них вибираються найкращі N.

2.3.4 Результати роботи алгоритму

Алгоритм EJS було порівняно з іншими популярними алгоритмами. Серед них були звичайний JS, HHO, GBO, WOA, AOA, SCA, BMO, SSA, SOA, PSO та MTDE. Це було зроблено з метою перевірки ефективності. У таблиці 2.11 наведено функції, на яких тестувалися алгоритми.

No	Function Name	Optimal Value	Dim	Search Range
F1	Storn's Chebyshev Polynomial Fitting Problem	1	9	[−8192, 8192]
F2	Inverse Hilbert Matrix Problem	1	16	[−16,384, 16,384]
F3	Lennard-Jones Minimum Energy Cluster	1	18	[−4, 4]
F4	Rastrigin's Function	1	10	[−100, 100]
F5	Griewangk's Function	1	10	[−100, 100]
F6	Weierstrass Function	1	10	[−100, 100]
F7	Modified Schwefel's Function	1	10	[−100, 100]
F8	Expanded Schaffer's F6 Function	1	10	[−100, 100]
F9	Happy Cat Function	1	10	[−100, 100]
F10	Ackley Function	1	10	[−100, 100]

Рисунок 2.11 – Тестові функції для EJS

На графіках 2.12 можна бачити графіки збіжності для кожного з методів на F2–F3. Серед усіх алгоритмів EJS має більш виражені

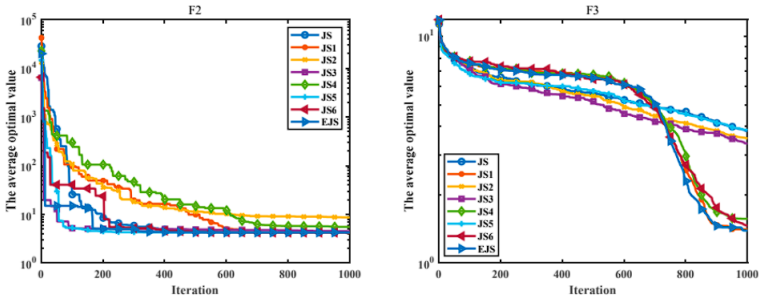


Рисунок 2.12 – Функції збіжності кожного метода

переваги при вирішенні задач із тестового набору функцій. Алгоритм мав найкращі результати на фоні інших методів на всіх функціях за винятком F4, F5 та F7. Отже, EJS здатен суттєво прискорити збіжність та покращити точність. У цей час більшість алгоритмів далекі від теоретичного оптимуму, що підтверджує, що стратегії, впроваджені в алгоритм EJS, значно покращують швидкість збіжності та точність у порівнянні з алгоритмом JS. Алгоритм EJS покращив точність та швидкість збіжності, але це призвело до зростання вимог до обсягу пам'яті та часу обчислення.

2.4 Інші актуальні алгоритми ройової оптимізації

Існує багато інших нових алгоритмів, які активно вивчаються та застосовуються в сучасних задачах. Таким є алгоритм сірих вовків (GWO) [3]. Вовки мають сильну здатність до захоплення здобичі, ведуть соціальний спосіб життя і всередині зграї існує соціальна ієрархія.

Для моделювання ієрархії у зграї вони поділяються на альфа, бета, дельта та омега особини. Найкращий індивід, другий та третій за якістю позначаються як α, β та δ, а решта – як ω. У GWO процес полювання контролюється вовками α, β та δ. Вони ведуть зграю до найкращого місця.

У процесі ітеративного пошуку ймовірне розташування здобичі оцінюється вовками α, β та δ. У процесі оптимізації їхнє положення оновлюються згідно з рівняннями:

$$\vec{D} = \left| \vec{C} \cdot \vec{X}_P(t) - \vec{X}(t) \right|,$$

$$\vec{X}(t+1) = \vec{X}_P(t) - \vec{A} \cdot \vec{D},$$

де t – номер ітерації, $\vec{A}$ та $\vec{C}$ – векторні коефіцієнти, $\vec{X}_P$ – вектор положення здобичі, $\vec{X}$ – положення вовка. Вектори $\vec{A}$ і $\vec{C}$ визначаються так:

$$\vec{A} = 2a \cdot \vec{r}_1 - a,$$

$$\vec{C} = 2 \cdot \vec{r}_2,$$

де a – коефіцієнт, який лінійно зменшується від 2 до 0 зі зростанням кількості ітерацій, $\vec{r}_1$ та $\vec{r}_2$ – випадкові вектори в межах $[0, 1]$.

Відстані до кожного з трьох лідерів розраховуються за формулами:

$$\vec{D}_\alpha = |\vec{C}_1 \cdot (\vec{X}_\alpha - \vec{X})|,$$

$$\vec{D}_\beta = |\vec{C}_2 \cdot (\vec{X}_\beta - \vec{X})|,$$

$$\vec{D}_\delta = |\vec{C}_3 \cdot (\vec{X}_\delta - \vec{X})|.$$

Оновлені положення для кожного з напрямків:

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1 \cdot \vec{D}_\alpha,$$

$$\vec{X}_2 = \vec{X}_\beta - \vec{A}_2 \cdot \vec{D}_\beta,$$

$$\vec{X}_3 = \vec{X}_\delta - \vec{A}_3 \cdot \vec{D}_\delta.$$

Підсумкове оновлене положення обчислюється як середнє:

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3}.$$

У GWO важливо досягнути балансу між розвідкою та експлуатацією:

Цей баланс регулюється векторами $\vec{A}$ та $\vec{C}$:

– якщо $|\vec{A}| > 1$, виконується розвідка;

– якщо $|\vec{A}| < 1$, виконується експлуатація.

Вектор $\vec{A}$ зменшується лінійно з кількістю ітерацій, а $\vec{C}$ генерується випадково на кожному кроці.

GWO має багато алгоритмів, які було створені на його основі з деякими покращеннями. Одним з таких алгоритмів є IGWO [29]. Цей алгоритм є покращеною версією GWO. Він включає механізми еволюції та усунення, щоб досягти належного компромісу між розвідкою та експлуатацією. Основними покращеннями алгоритму є додавання принципу «виживання сильніших», використання диференціальної еволюції як еволюційної моделі для вовків та оновлення зграї вовків згідно з принципом SOF для уникнення локальних оптимумів.

Також існує алгоритм оптимізації горбатого кита [3].Вони зазвичай організовані в групи та мігрують із тропічних вод до полярних. Однією з найцікавіших характеристик горбатих китів є використання методики годування «мережа бульбашок». Вони використовують вокалізації для комунікації та створюють сітку з бульбашок у спіральній траєкторії, щоб оточити здобич.

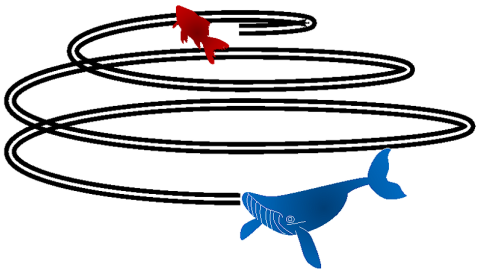


Рисунок 2.13 – Процес харчування в мережі бульбашок

Найкращий кандидат на рішення вважається здобиччю, положення китів оновлюються відносно цього оптимального розв'язку. На кожній ітерації оновлюється також найкращий учасник популяції, якщо він

знаходить краще рішення. WOA моделює техніку годування через оточення здобичі зі зменшенням радіусу та оновлення положення по спіралі.

Поведінка дослідження здійснюється аналогічно процесу оточення, використовуючи випадково згенеровані коефіцієнтні вектори для балансування між фазами дослідження та експлуатації.

Після ініціалізації, на кожній ітерації положення V_{whale} обчислюється як:

$$V_{\text{whale}} = \begin{cases} V_{\text{rand\_whale}} - R_I \cdot W, & (\text{дослідження},) \\ V_{\text{leader\_whale}} - R_I \cdot W, & (\text{оточення здобичі},) \\ W \cdot e^{p_1 \cdot p_2} \cdot \cos(2\pi p_2) + V_{\text{leader\_whale}}, & (\text{годування бульбашками},) \end{cases}$$

де параметри визначаються як:

$$p = \text{rand}(0,1), \quad p_1 = 1, \quad p_2 = (c_2 - 1) \cdot \text{rand}(0,1) + 1,$$

а коефіцієнти:

$$c_1 = 2 - \frac{2i}{\text{MaxIt}}, \quad c_2 = 1 + \frac{i}{\text{MaxIt}}.$$

Значення R_I та p визначають, який із трьох механізмів буде застосовано на ітерації i.

Є ще один новий алгоритм SSA [30] натхненний колективною поведінкою морських сальп. Цей алгоритм достатньо простий в реалізації, а також має хороший баланс між дослідженням і експлуатацією. У SSA є агенти типів лідер та послідовники.

Лідер визначає напрям до їжі, що є глобальним оптимумом на поточній ітерації. Послідовники оновлюють свої координати, орієнтуючись на попередніх сальп у ланцюгу.

Нехай $S_{i,j}^t$ – j-та координата i-го сальпа на ітерації t, а F_j^t – j-та координата найкращого знайденого рішення, тоді позиція лідера

оновлюється за формулою:

$$S_{1,j}^{t+1} = \begin{cases} F_j^t + c_1 \left((ub_j - lb_j)c_2 + lb_j \right), & \text{якщо } c_3 \geqslant 0, \\ F_j^t - c_1 \left((ub_j - lb_j)c_2 + lb_j \right), & \text{якщо } c_3 < 0, \end{cases}$$

де:

– t – номер ітерації;

– ub_j, lb_j – верхня і нижня межі j-го параметра простору;

– $c_2 \sim \mathcal{U}(0,1)$, $c_3 \sim \mathcal{U}(-1,1)$ – випадкові коефіцієнти;

– c_1 – параметр, який керує балансом між дослідженням і експлуатацією, і обчислюється як:

$$c_1 = 2e^{-\left(\frac{4t}{T}\right)^2},$$

де T – максимальна кількість ітерацій.

Позиції послідовників (для $i \geqslant 2$) оновлюються за середнім арифметичним:

$$S_{i,j}^{t+1} = \frac{1}{2} \left(S_{i,j}^t + S_{i-1,j}^t \right).$$

На кожній ітерації всі агенти оцінюються за цільовою функцією, і найкраще знайдене рішення встановлюється як нове джерело їжі.

2.5 Шляхи гібридизації ройових методів й алгоритмів з метою підвищення їх ефективності на прикладних оптимізційних задачах

Оптимізаційні задачі дуже складні та жодна стратегія оптимізації не гарантує отримання оптимального розв'язку. Алгоритми ройового інтелекту показують хороші результати в розв'язку таких задач, але проблеми стають все більш складними, тож потребують точнішого розв'язку. Тоді було придумано об'єднувати підходи для покращення загального результату.

2.5.1 Гібридний алгоритм GACO

Алгоритм наведено в [17]. У запропонованому підході генетичний алгоритм використовується для оптимізації параметрів алгоритму ACO на основі вихідних результатів. Алгоритм ACO виконує роль функції пристосованості для GA. ACO працює з оптимізованими параметрами, генеруючи маршрут і його довжину для задачі комівояжера, яка слугує вхідним параметром для GA. Таким чином, усувається потреба у ручному підборі параметрів.

Алгоритм починається з ініціалізації GA та генерації початкової популяції. Для кожної хромосоми обчислюється функція пристосованості, після чого запускається ACO. У рамках ACO створюються мурахи, які розміщуються у початковий стан з ініціалізацією порожніх маршрутів. Для кожної мурахи відбувається вибір наступного вузла, і якщо маршрути не завершені, процес повертається до попереднього кроку. Після завершення маршрутів виконується оновлення локального та глобального феромону. Якщо критерій завершення не виконано, алгоритм повертається до створення мурах. Після завершення ACO перевіряються критерії оптимізації. Якщо вони виконані, алгоритм переходить до завершення, якщо ні – відбувається вибір, схрещування, мутація та формування популяції, потім обчислюється функція пристосованості. У кінці обирається краща хромосома та алгоритм завершується. Запропонований алгоритм GACO має покращити продуктивність генетичного алгоритму шляхом включення локального пошуку з використанням ACO для задачі TSP.

Згідно з результатами, гібридний генетичний алгоритм швидко знаходить близькі до оптимальних розв'язки. Після декількох поколінь всі ребра глобально оптимального маршруту вже присутні у популяції та кількість унікальних ребер менша, ніж подвійний розмір задачі.

В алгоритмі мурах всі особини сходяться до найкращого маршруту.

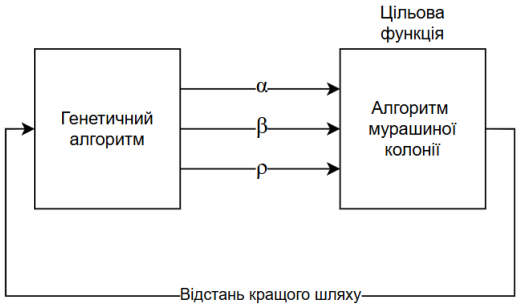


Рисунок 2.14 – Блок-схема запропонованого методу

Розподіл феромонів по ітераціях і вибір наступного міста базується на максимальній ймовірності. Аналіз демонструє, що ребра з багатим феромоном утворюють найкращий маршрут у задачі комівояжера.

У цьому ACO-підході для отримання найкращого рішення потрібно збалансувати компроміс між експлуатацією й дослідженням. Низький коефіцієнт випаровування дозволяє феромонам залишатися довше, однак це компенсується високою упередженістю вибору маршруту.

При порівнянні ACO, GA та GACO було взято координати 40 міст США. Спочатку алгоритм відвідує найближчі ще не відвідані міста, поки не буде пройдено всі міста. Потім обчислюється мінімальна відстань і загальний час туру.

Таблиця 2.3 – Порівняння результатів ACO, GA та GACO для задачі комівояжера

Алгоритм	**Довжина маршруту**	**Час виконання (с)**
GA	25.3990	540.6894
ACO	16.6220	777.8693
GACO	**5.1689**	**280.3890**

2.5.2 Гібридізація алгоритмів PSO та GA

Алгоритм наведено в [16].

Відомою перевагою GA є збереження різноманіття популяції, тому

багато гібридних підходів поєднують оператори GA з PSO.

Адаптивна мутація залежить від продуктивності. Наприклад, гаусівську мутацію використовують з кроком, що змінюється залежно від найкращої пристосованості. Гібриди PSO з оператором кросоверу також демонструють покращену продуктивність. Поширеним є використання квадратичного та випадкового кросоверу з *gbest*. Також хорошою практикою є реалізація елітарності через оновлення частинки за *pbest* та *lbest* та використання ймовірність обміну координат з *pbest*.

Адаптивний кросовер досліджено значно менше, ніж кросовер з мутацією. Жоден відомий підхід не використовує адаптивну ймовірність кросоверу. Деякі роботи поєднують кросовер і мутацію, проте адаптивна параметризація не застосовувалась.

У дослідженні гібридизації PSO алгоритм поділено на три. Перший поєднує адаптивну параметризацію як для кросоверу, так і для мутації, як показано на. Другий та третій – лише одну з цих операцій. Алгоритм виконує кросовер між двома частинками, які вибираються з імовірністю, пропорційною їхній пристосованості. Кожна координата, починаючи з першої до останньої, змінюється згідно з адаптивною ймовірністю кросоверу C_p.

Коли відстань між частинками та їхнім *pbest* наближається до нуля, рух припиняється, і рій збігається.

На відміну від періодичного кросоверу, ця реалізація використовує адаптивну ймовірність. Порогове значення r вибирається випадково з інтервалу $[0, 1]$ та порівнюється з C_p, щоб вирішити, чи змінювати координату d частинки за допомогою pbest-кросоверу. Зміна виконується як середнє значення відповідних *pbest* двох частинок.

Операція мутації використовує адаптивну ймовірність M_p, аналогічну C_p. Вона визначається рівнянням:

$$X_i(d) = X_i(d) + \text{Gaussian}(),$$

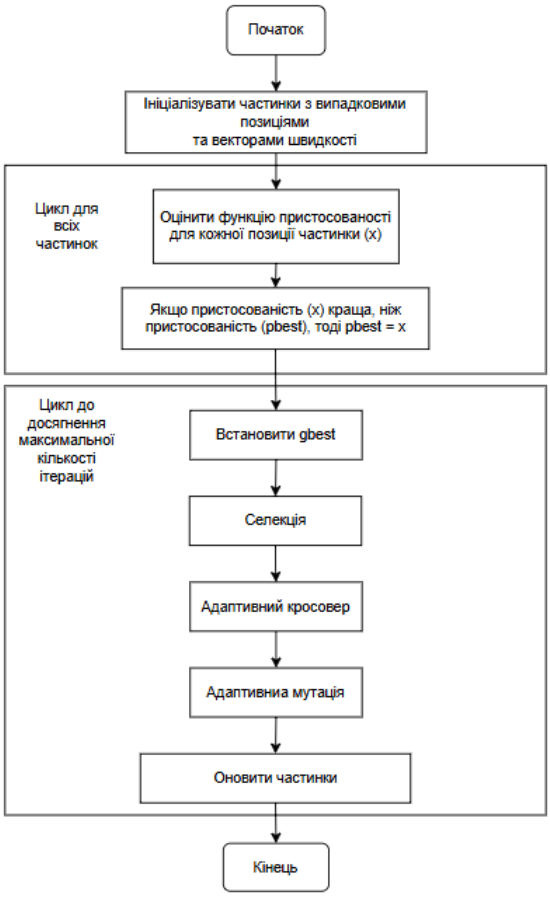


Рисунок 2.15 – Гібрид PSO з адаптивною параметризацією

де *Gaussian()* – гаусівська функція, яка повертає випадкове значення в межах 10% розміру координати.

Algorithm 2.1 Адаптивний кросовер

1: **for** кожної частинки x_i в популяції **do**
2: Обчислити адаптивну ймовірність кросоверу C_p
3: Встановити $r \sim \mathcal{U}(0,1)$
4: Встановити d – випадкову координату $[0, \dim]$
5: Випадково вибрати $pbest_1$ і $pbest_2$ серед усіх частинок
6: **if** $r < C_p$ **then**
7: $x_{id} = x_{id} + \dfrac{pbest_{1d} + pbest_{2d}}{2}$
8: **end if**
9: **end for**

Метою експериментів є порівняння адаптивної ймовірності

кросоверу (ACR) мутації (AMR), поєднаного кросоверу та мутації (ACMR) адаптивної інерційної ваги без гібридизації (AIW).

Спочатку кожен набір комбінується з чотирма варіантами адаптивної інерційної ваги. Найуспішніша комбінація використовується для порівняння між ACR, AMR та ACMR.

Кожен експеримент повторювався 30 разів по 2000 ітерацій. Таким чином, кожен алгоритм мав по 80000 викликів функції пристосованості. Загальні налаштування були наступні: кількість частинок n становила 40, розмірність функції – $dim = 30$, коефіцієнт особистого навчання $c_1 = 0.9$, коефіцієнт соціального навчання $c_2 = 0.9$.

Щоб визначити найбільш відповідний адаптивний підхід для гібридів PSO, було протестовано чотири стратегії адаптації інерційної ваги. Вони базуються на показниках продуктивності PSO, наведених у Таблиці 2.4.

Таблиця 2.4 – Адаптивні підходи

Підхід	**Адаптивний фактор**
Adaptive 1	Особиста пристосованість, найкраща пристосованість
Adaptive 2	Глобальна + особиста пристосованість
Adaptive 3	Особиста пристосованість з ранжуванням
Adaptive 4	Положення частинки

У першому підході використовували значення 0.4 і 0.8. У четвертий підхід – 0.9 і 0.3 відповідно. Третій підхід має індивідуальні мінімальні та максимальні значення. У другому – мінімальне значення параметра є початковим.

Результати в Таблиці 2.5 показують, що гібриди з адаптивною параметризацією здебільшого перевершують класичний PSO з інерційною вагою.

ACR переважає AIW в більшості випадків, але поступається AMR. ACMR дає кращі результати, ніж лише кросовер.

Таблиця 2.5 – Середня найкраща пристосованість (стандартне відхилення) та середня кількість ітерацій до досягнення найкращого розв'язку (30 запусків)

f	ACR	AMR	ACMR	AIW
f_1	2.34E-07 (1.65E-05) 1702	**6.06E-52** (4.76E-48) 1997	1.54E-10 (2.63E-05) 1865	6.88E-08 (7.66E-06) 956
f_2	4.35E-04 (3.20E-02) 833	**3.68E-05** (8.40E-04) 1903	6.47E-05 (1.93E-05) 1074	2.58E-04 (5.94E-01) 1630
f_3	8.08E-04 (2.15E-05) 1502	**4.93E-07** (1.15E-09) 1397	1.14E-02 (1.40E-02) 1660	1.07E-02 (2.56E-01) 1020
f_4	1.46E-03 (5.53E+00) 1644	**1.05E-06** (1.44E+01) 1388	1.66E-05 (2.85E+01) 1396	3.88E-05 (7.83E+00) 794
f_5	1.64E-02 (3.15E-01) 1862	**1.02E-03** (2.21E-01) 1918	2.80E-03 (2.73E-04) 1825	2.96E-02 (1.88E-01) 812
f_6	2.40E-09 (7.66E-11) 1517	**3.91E-11** (5.74E-17) 1528	2.20E-09 (2.88E-07) 1533	6.99E-09 (5.87E-05) 947

На основі сучасного стану розробки PSO можна стверджувати, що адаптивна параметризація та гібридизація можуть суттєво підвищити ефективність алгоритму.

Найкращі результати серед гібридів було отримано завдяки включенню адаптивної мутації.

Висновки до розділу 2

У цьому розділі було розглянуто сучасні алгоритми ройового інтелекту, які показали хороші результати на прикладних задачах. Хороший алгоритм має бути не тільки ефективним, здібним до масштабування та гнучким, а ще й мати хороший баланс між навчанням та дослідженням, велике різноманіття особин та швидка конвергенція. Також треба запобігати застрягання особин в локальних мінімумах. В кожному з методів ці результати можливо досягнути різни засобами. В одних алгоритмах додаються нові стратегії навчання та оператор локального виходу, наприклад, в покращеному алгоритмі пошуку медуз.

А такий алгоритм як IGWO для досягнення того результату використовує диференційну еволюцію для оновлення позицій кожного елемента популяції. Також достатньо поширеною практикою стало створювати гібридні алгоритми. Генетичний алгоритм один з найпопулярніший, з яким схрещують інші методи. Однією з головних причин цьому є те, що багато алгоритмів, які створені на основі природних явищ не зберігають різноманіття популяції протягом всіх ітерацій, а генетичний алгоритм допомагає це запобігти. Як наслідок, гібридизація надає змогу використовувати переваги декількох алгоритмів одночасно. Отже, задача покращення алгоритмів полягає в тому, щоб виявляти слабкі місця в їхній роботі та зайти потрібні методи, щоб їх усунути.

3 РЕАЛІЗАЦІЯ МОДЕЛЕЙ ПРИКЛАДНИХ ЗАДАЧ, ТЕСТУВАННЯ І ПОРІВНЯЛЬНИЙ АНАЛІЗ РОБОТИ РОЙОВИХ АЛГОРИТМІВ НА РЕАЛІЗОВАНИХ МОДЕЛЯХ

3.1 Технічне середовище експерименту

В якості мови програмування для імплементації методів було обрано Python 3.11, тому що він містить велику кількість бібліотек з готовим функціоналом, що спрощує роботу. Код виконувався в Jupyter Notebook з метою простішої декомпозицій окремих частин програми та зручнішого аналізу результатів, використовували. Программа запускалась локально в інтеграному серидовищі Visual Studio Code. Математичні обчислення

Таблиця 3.1 – Характеристики системи

RAM	8 GB
CPU	AMD Ryzen 5 4600H, 6 ядер, частота 3.0 GHz
GPU 0	AMD Radeon (TM) Graphics
GPU 1	NVIDIA GeForce GTX 1650

здійснювалися за допомогою функцій та констант з бібліотеки numpy. Також використовувалася гамма-функція з бібліотеки scipy.special. В даній роботі також було виміряно час роботи кожного алгоритму для різних цільових функцій та фіксованих вхідних параметрів за допомогою модулю time. Для обробки та запису отриманих результатів було використано бібліотеку pandas, бо цей інструмент дозволяє працювати з різними обсягами даних, об'єднати їх та обробляти значення. Частина роботи також була приділена побудові графіків та діаграм. Для цього було використано бібліотеку matplotlib, яка являє собою зручний високорівневий інтерфейс для візуалізації даних. Для моделювання

оптимізаційного процесу проблем з набору CEC2014 було використано бібліотеку opfunu, в якій зібрано понад 500 оптимізаційних задач, в тому числі й набори зі змагань CEC. Для зручного представлення та організації таблиць використовувався Google Sheets.

3.2 Хід проведення обчислювальних експериментів

Кожен алгоритм тестувався на двох наборах функцій. Перший набір являє собою колекцією класичних оптимізаційних проблем, які описано в статті [27].

Таблиця 3.2 – Функції, які використовувалися в дослідженні

№	Назва	Границі	Положення оптимуму
F01	Sphere	$[-100, 100]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F02	Ellipsoid	$[-100, 100]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F03	Sum of Different Powers	$[-10, 10]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F04	Quintic	$[-20, 20]^D$	$\mathbf{x}^* = \{-1, -1, \ldots, -1\}$ or $\{2.2, \ldots, 2.2\}$
F05	Drop-Wave	$[-5.12, 5.12]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F06	Weierstrass	$[-0.5, 0.5]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F07	Alpine 1	$[-10, 10]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F08	Ackley's	$[-32.768, 32.768]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F09	Griewank's	$[-100, 100]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F10	Rastrigin's	$[-5.12, 5.12]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F11	HappyCat	$[-20, 20]^D$	$\mathbf{x}^* = \{-1, -1, \ldots, -1\}$
F12	HGBat	$[-15, 15]^D$	$\mathbf{x}^* = \{-1, -1, \ldots, -1\}$
F13	Rosenbrock's	$[-10, 10]^D$	$\mathbf{x}^* = \{1, 1, \ldots, 1\}$
F14	High Conditioned Elliptic	$[-100, 100]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F15	Discus	$[-100, 100]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F16	Bent Cigar	$[-100, 100]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F17	Perm D, Beta	$[-D, D]^D$ generally[2]	$\mathbf{x}^* = \{1, 2, \ldots, D\}$
F18	Schaffer's F7	$[-100, 100]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F19	Expanded Schaffer's F6	$[-100, 100]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F20	Rotated Hyper-ellipsoid	$[-100, 100]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F21	Schwefel	$[-500, 500]^D$	$\mathbf{x}^* = \{c, c, \ldots, c\}^3$
F22	Sum of Different Powers 2	$[-10, 10]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F23	Xin-She Yang's 1	$[-2\pi, 2\pi]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F24	Schwefel 2.21	$[-100, 100]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F25	Schwefel 2.22	$[-100, 100]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F26	Salomon	$[-20, 20]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F27	Modified Ridge	$[-100, 100]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F28	Zakharov	$[-10, 10]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F29	Modified Xin-She Yang's 3	$[-20, 20]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$
F30	Modified Xin-She Yang's 5	$[-100, 100]^D$	$\mathbf{x}^* = \{0, 0, \ldots, 0\}$

Видно, що майже всі функції, окрім F4 мають один локальний мінімум та усі функції, окрім F4, F11, F12, F17 та F21 мають єдиний

локальний мінімум в точці $\mathbf{x}^* = \{0, 0, \ldots, 0\}$. Це може призводити до упередженості процедури визначення найкращого оптимізатора. Цей набір не є показовим для аналізу здібності до оптимізації, він використовується для спостереження за тенденціями, які відбуваються в роботі алгоритмів протягом циклу роботи.

Другий набір складається з функцій, які було репрезентовано на конкурсі кращих оптимізаційних алгоритмів CEC2014 [12]. Цей набір є ускладненим шляхом додаванням ротацій та зсувів у класичні функції.

Таблиця 3.3 – Категорії функцій та їх оптимальні значення

№	**Функції**	$F_i^* = F_i(x^*)$
	Унімодальні функції	
1	Rotated High Conditioned Elliptic Function	100
2	Rotated Bent Cigar Function	200
3	Rotated Discus Function	300
4	Shifted and Rotated Rosenbrock's Function	400
5	Shifted and Rotated Ackley's Function	500
6	Shifted and Rotated Weierstrass Function	600
7	Shifted and Rotated Griewank's Function	700
8	Shifted Rastrigin's Function	800
9	Shifted and Rotated Rastrigin's Function	900
	Прості мультимодальні функції	
10	Shifted Schwefel's Function	1000
11	Shifted and Rotated Schwefel's Function	1100
12	Shifted and Rotated Katsuura Function	1200
13	Shifted and Rotated HappyCat Function	1300
14	Shifted and Rotated HGBat Function	1400
15	Shifted and Rotated Expanded Griewank's plus Rosenbrock's Function	1500
16	Shifted and Rotated Expanded Scaffer's F6 Function	1600
	Гібридні функції	
17	Hybrid Function 1 (N=3)	1700
18	Hybrid Function 2 (N=3)	1800
19	Hybrid Function 3 (N=4)	1900
20	Hybrid Function 4 (N=4)	2000
21	Hybrid Function 5 (N=5)	2100
22	Hybrid Function 6 (N=5)	2200
	Композиційні функції	
23	Composition Function 1 (N=5)	2300
24	Composition Function 2 (N=3)	2400
25	Composition Function 3 (N=3)	2500
26	Composition Function 4 (N=5)	2600
27	Composition Function 5 (N=5)	2700
28	Composition Function 6 (N=5)	2800
29	Composition Function 7 (N=3)	2900
30	Composition Function 8 (N=3)	3000
	Search Range: $[-100, 100]^D$	

Тут D – розмірність. Для зручності було використано один діапазон пошуку. Кожну функцію було зсунуто масштабовано та обернено.

В даній роботі можна виділити наступні дослідниці питання:

1) Чи перевершує оптимізатор Cock-Chicken-Hen оригінальний Chicken Serach Optimizer за якістю результатів на стандартних бенчмарк-функціях?

2) Які результати оптимізації демонструють досліджувані алгоритми на стандартних функціях?

3) Як ці три алгоритми показують себе у порівнянні з іншими відомими методами?

4) Який із трьох обраних алгоритмів є найефективнішим для вирішення практичної задачі?

3.3 Порівняння Cock-Chicken-Hen з оригінальним Chicken Serach Optimizer

В оригінальній статті [14] питання, чи кращий алгоритм CHC за алгоритм CSO, на основі якого і було розроблено CHC. Цей аналіз було включено в дану роботу, щоб дізнатися, чи є сенс у вивченні цього алгоритму надалі. Для цього обидва алгоритми було запущено на першому наборі функцій. Розмірність для кожної функції встановлювалась як $D = 10$. У таблиці 3.4 вказано початкові параметри

Таблиця 3.4 – Параметри запуску алгоритмів CHC та CSO

Розмір популяції	100
Кількість ітерацій	150
Кількість запусків	50
Кількість викликів функції	15000

запуску кожного з алгоритмів. В даному експерименті для кожної з тридцяти функцій було зафіксовано середнє значення знайденого мінімуму та стандартне відхилення для п'ятдесяти запусків, а також побудовано графіки медіанної збіжності для відслідковування швидкості

збіжності до мінімуму.

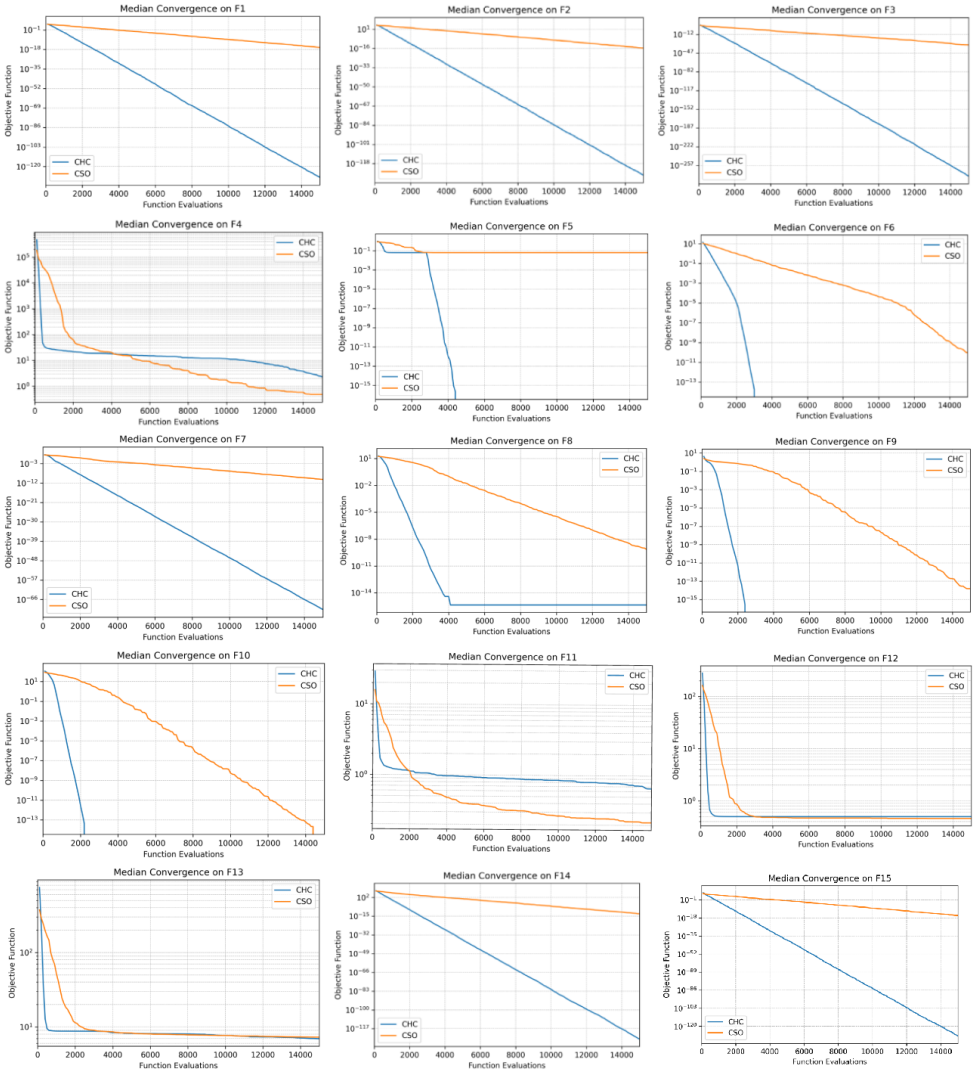


Рисунок 3.1 – Графіки медіанної збіжності для F1-F15

На графіках 3.1 видно, що майже в усіх випадках, окрім F4 та F11 переважає CHC. А результати CSO в багатьох випадках навіть до оптимуму не доходять. А в таблиці 3.5 бачимо, що CHC на відміну від CSO.

Об'єктивно хороший результат алгоритм CSO показав на функціях F4 та F11. В цих випадках середнє значення та стандартне відхилення менше, ніж в CHC. Проте варто зазначити, що ця різниця незначна в порівнянні з тим, що на інших функціях CSO демонструє на порядок нижчу продуктивність. З графіків 3.2 та таблиці 3.6 видно, що тенденція з ефективністю алгоритмів така сама. На тільки на функціях F17 та F21

Таблиця 3.5 – Порівняння середнього мінімума та стандартного відхилення для F1–F15

Fn	Stat	CHC	CSO
F1	Avg	**5,43E-126**	1,39E-16
	Std	8,69E-01	**7,22E-01**
F2	Avg	**9,19E-126**	4,51E-16
	Std	**4,32E-125**	6,23E-16
F3	Avg	**1,95E-257**	7,35E-29
	Std	**0,00E+00**	4,49E-28
F4	Avg	2,32E+00	**7,64E-01**
	Std	9,79E-01	**5,91E-01**
F5	Avg	**1,15E-02**	6,38E-02
	Std	2,45E-02	**1,63E-08**
F6	Avg	**0,00E+00**	9,43E-10
	Std	**0,00E+00**	4,41E-09
F7	Avg	**1,22E-68**	7,14E-10
	Std	**6,17E-68**	2,49E-09
F8	Avg	**4,44E-16**	9,47E-10
	Std	**0,00E+00**	9,18E-10
F9	Avg	**0,00E+00**	1,67E-04
	Std	**0,00E+00**	1,17E-03
F10	Avg	**0,00E+00**	1,42E-12
	Std	**0,00E+00**	5,54E-12
F11	Avg	6,49E-01	**2,34E-01**
	Std	1,26E-01	**5,21E-02**
F12	Avg	**4,92E-01**	4,60E-01
	Std	**1,64E-02**	4,23E-02
F13	Avg	**7,11E+00**	7,38E+00
	Std	**5,01E-01**	2,60E-01
F14	Avg	**2,75E-122**	9,16E-13
	Std	**1,76E-121**	2,02E-12
F15	Avg	**1,85E-124**	8,59E-16
	Std	**1,07E-123**	1,33E-15

Таблиця 3.6 – Порівняння середнього мінімума та стандартного відхилення для F16–F30 (виділення згідно з таблицею)

Fn	Stat	CHC	CSO
F16	Avg	**1,52E-118**	3,50E-10
	Std	**6,91E-118**	8,21E-10
F17	Avg	9,06E+15	**7,01E+15**
	Std	**3,71E+15**	3,75E+11
F18	Avg	**2,62E-133**	5,23E-17
	Std	**1,02E-132**	1,57E-16
F19	Avg	**4,58E-02**	1,19E+00
	Std	**1,83E-01**	6,34E-01
F20	Avg	**9,22E-123**	2,22E-15
	Std	**4,31E-122**	5,66E-15
F21	Avg	1,69E+03	**1,03E+03**
	Std	**1,99E+02**	2,97E+02
F22	Avg	**1,82E-104**	6,80E-15
	Std	**9,01E-104**	1,29E-14
F23	Avg	**2,38E-03**	2,66E-03
	Std	**1,61E-04**	1,38E-04
F24	Avg	**4,36E-59**	2,58E-06
	Std	**1,96E-58**	2,88E-06
F25	Avg	**3,06E-68**	4,06E-10
	Std	**1,10E-67**	4,52E-10
F26	Avg	**2,40E-02**	9,99E-02
	Std	4,27E-02	**7,09E-08**
F27	Avg	**1,10E-13**	3,17E-02
	Std	**1,05E-13**	7,58E-03
F28	Avg	**1,45E-95**	2,19E-05
	Std	**9,66E-95**	4,26E-05
F29	Avg	**0,00E+00**	0,00E+00
	Std	**0,00E+00**	0,00E+00
F30	Avg	**1,00E+04**	1,00E+04
	Std	**9,05E-01**	1,10E+00

CSO показав кращий середній мінумум. Проте за значеннями в таблиці 3.6 видно, що результати алгоритмів в обох випадках значно гірші за оптимальні. У всіх інших випадках алгоритм CHC показав або краще

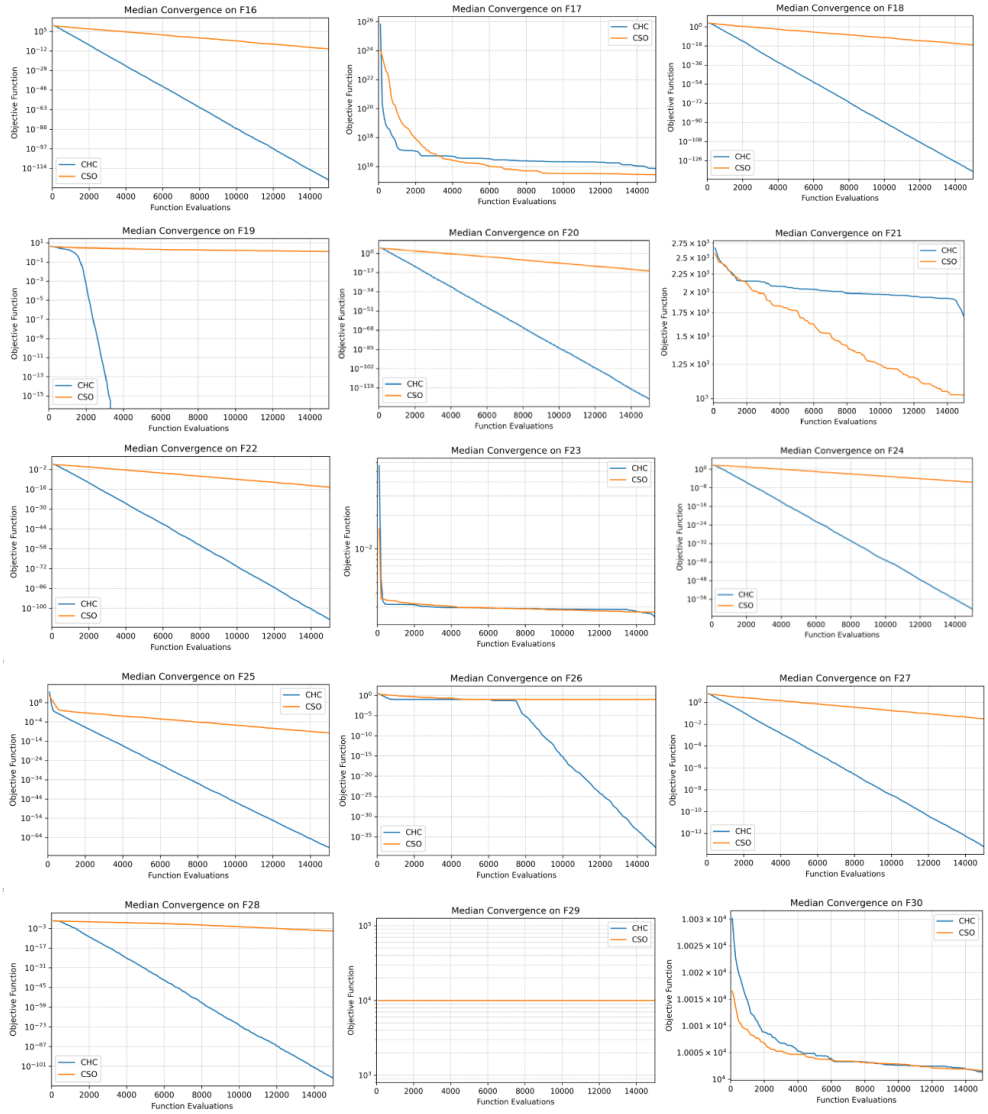


Рисунок 3.2 – Графіки медіанної збіжності для F16-F30

значення, або таке саме.

3.4 Тестування CHC, TBO та EJS на функціях з наборів

Спочатку усі три алгоритми було запущено на функціях першого набору з метою перевірити здатність алгоритмів до оптимізації в загальних рисах та провести аналіз переваг та недоліків на різних типах функцій.

У таблиці 3.7 наведено значення початкових параметрів. Кількість викликів функції є добутком розміру популяції та кількості ітерацій.

Таблиця 3.7 – Параметри запуску алгоритмів на першому наборі

Розмір популяції	50
Кількість ітерацій	50
Кількість запусків	50
Кількість викликів функції	2500

З графіків F1–F15 одразу можна сказати, що CHC збігається до мінімуму швидше. А там, де не найшвидше, як у F4, F11 та F13, усі досліджувані алгоритми показали далекий від оптимального результат. За винятком F12 – на цій функції TBO відпрацював найкраще із запропонованих алгоритмів.

На функціях F16–F30 CHC теж показав себе як алгоритм, який швидше за інші збігається. Якщо звернути увагу на графіки оптимізації EJS можна помітити тенденцію різких провалів в глобальний мінімум. На функціях F17, F21, F29 та F30 усі розглянуті алгоритми показали поганий процес збіжності до мінімуму. F29 та F30 на рисунках 3.5 та 3.6 мають багато локальних мінімумів, а амплітуди між локальними мінімумами мають порядок $10^3 - 10^4$. Тож алгоритми можуть застрягати. EJS демонструє специфічну поведінку на цих функціях, бо протягом ітерацій працює так само як і інші алгоритми, навіть на F30 має гіршу швидкість збіжності, і тільки в кінці виходить з локального мінімуму. Для дослідження отриманих значень та порівняння з результатами було досліджено було використано наступні метрики:

– **середнє значення (Avg)**:

$$\text{Avg} = \frac{1}{n}\sum_{i=1}^{n} f_i;$$

– **стандартне відхилення (Std)**:

$$\text{Std} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (f_i - \text{Avg})^2};$$

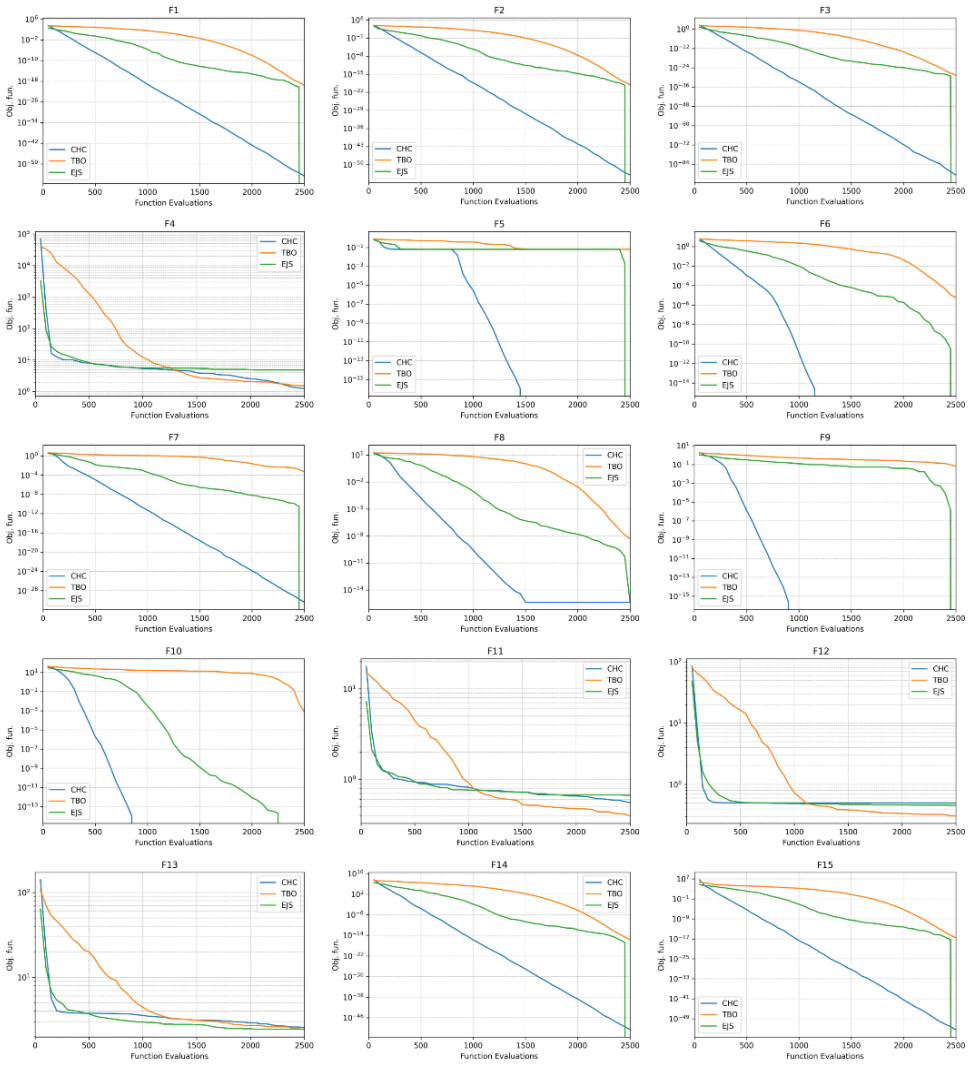


Рисунок 3.3 – Графіки медіанної збіжності для F1–F15

– **медіана (Median)**:

$$\text{Median} = \begin{cases} f_{(k)}, & \text{якщо } n = 2k-1, \\ \frac{f_{(k)}+f_{(k+1)}}{2}, & \text{якщо } n = 2k; \end{cases}$$

– **нормалізована відстань у просторі змінних** (Δx):

$$\Delta x = \sqrt{\frac{1}{D}\sum_{i=1}^{D}\left(\frac{x_i - x_i^*}{R_i}\right)^2},$$

Дозволяє визначити, наскільки далеко знаходиться знайдена оптимальна позиція від справжньої.

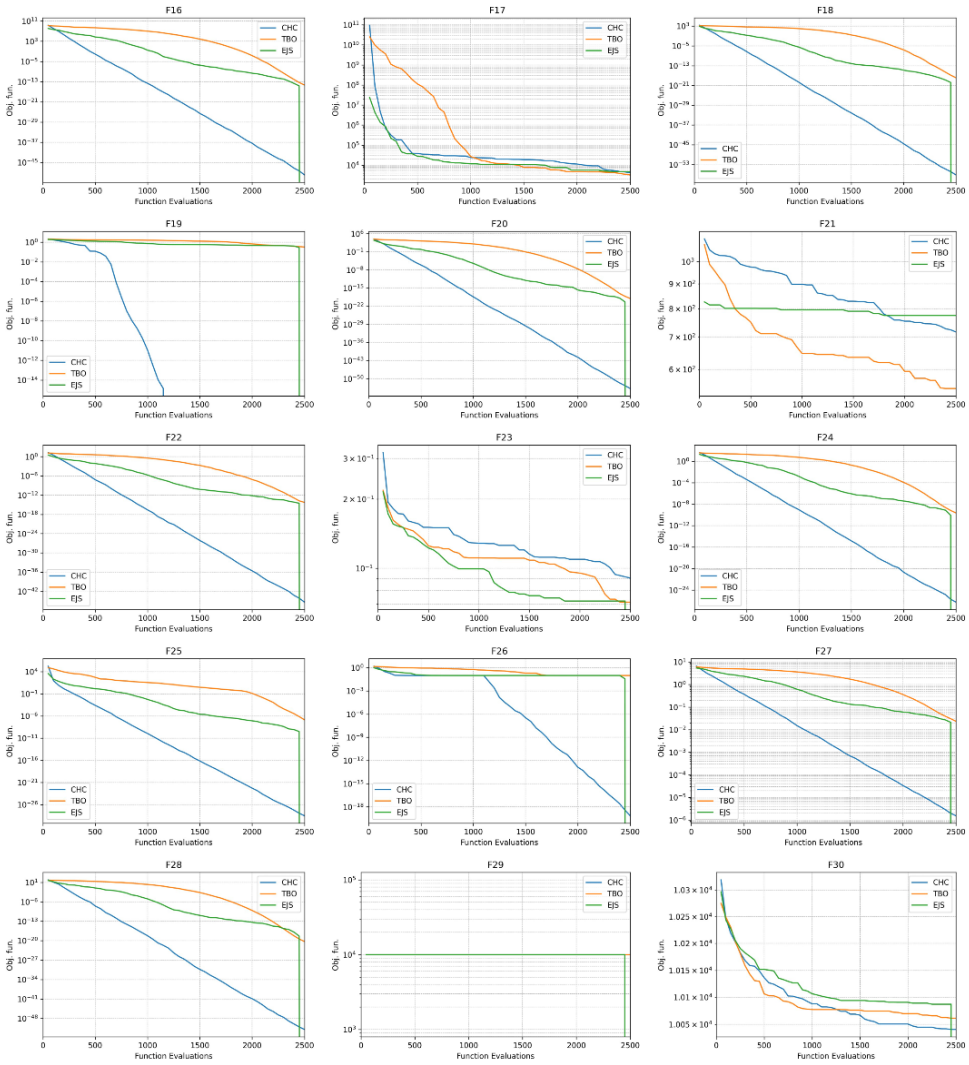


Рисунок 3.4 – Графіки медіанної збіжності для F16–F30

– **нормалізована помилка функції** (Δf):

$$\Delta f = \frac{f_{\min}}{f^*_{\max}};$$

визначає похибку знайденого значення функції.

– **інтегральна метрика якості** (Δt):

$$\Delta t = \sqrt{\frac{\Delta x^2 + \Delta f^2}{2}}.$$

Об'єднуємо результати Δx та Δf, щоб оцінити загальну точність.

Тут n – кількість запусків, x_i - знайдена точка мінімума. x_i^* – справжня точка мінімума. f_{min} – знайдений мінімум. $f^*_{\max}$ – справжній мінімум.

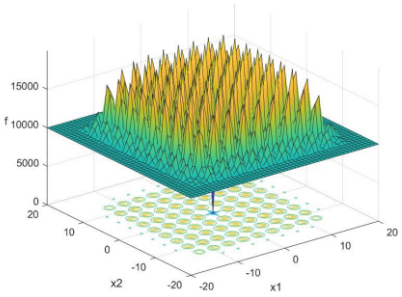


Рисунок 3.5 – 3D-візуалізація функції F29

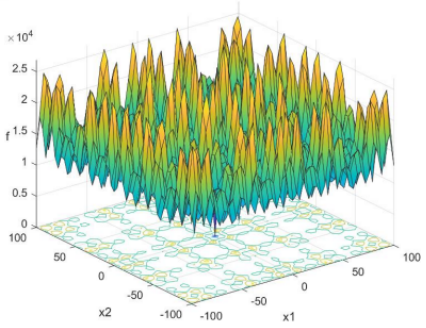


Рисунок 3.6 – 3D-візуалізація функції F30

Таблиця 3.8 – Середні значення для 50 запусків

ID	GA	PSO	SQP	CHC	TBO	EJS
F01	7.59E-04	7.37E-03	1.38E-14	1.40E-48	1.09E-19	**0.00E+00**
F02	1.02E-03	2.86E-02	9.16E-14	4.78E-51	1.81E-19	**0.00E+00**
F03	4.25E-05	2.70E-07	5.68E-11	9.30E-84	8.29E-28	**0.00E+00**
F04	2.12E-01	2.42E-01	3.78E-06	1.30E+00	1.47E+00	**0.00E+00**
F05	1.86E-01	6.38E-02	8.61E-01	3.83E-03	5.87E-02	**0.00E+00**
F06	3.21E-01	4.78E-02	5.89E+00	**0.00E+00**	6.74E-06	**0.00E+00**
F07	8.21E-03	2.32E-02	2.52E-06	7.61E-28	1.02E-02	**0.00E+00**
F08	2.60E-01	3.21E-02	1.89E+01	**4.44E-16**	7.31E-09	**4.44E-16**
F09	1.48E-02	1.58E-01	3.35E+00	5.30E-03	8.47E-02	**0.00E+00**
F10	5.93E-01	3.49E+00	2.55E+01	**0.00E+00**	8.87E-01	**0.00E+00**
F11	4.66E-01	1.89E-01	**4.18E-02**	5.30E-01	4.11E-01	6.78E-01
F12	5.27E-01	**2.65E-01**	4.90E-01	4.81E-01	2.73E-01	4.47E-01
F13	2.57E+00	2.54E+00	**5.50E-01**	2.67E+00	2.28E+00	2.41E+00
F14	7.39E+01	6.00E+01	3.50E-11	7.29E-46	3.91E-16	**0.00E+00**
F15	5.80E+02	1.86E+02	1.50E-11	5.12E-51	6.92E-17	**0.00E+00**
F16	7.71E+02	2.42E+03	1.16E-10	3.35E-46	9.42E-15	**0.00E+00**
F17	3.86E+03	7.69E+02	**7.81E+01**	5.23E+03	2.45E+03	7.95E+03
F18	3.08E-01	1.01E-01	7.12E+01	1.69E-52	4.71E-18	**0.00E+00**
F19	9.11E-01	7.40E-01	2.32E+00	1.78E-02	3.46E-01	**0.00E+00**
F20	2.03E-03	9.91E-03	6.24E-14	5.19E-52	1.69E-19	**0.00E+00**
F21	**2.58E+02**	2.84E+02	9.63E+02	6.86E+02	5.64E+02	6.82E+02
F22	5.61E-05	8.71E-07	9.31E-13	9.18E-44	1.24E-14	**0.00E+00**
F23	4.84E-02	9.07E-02	1.68E-01	9.08E-02	7.12E-02	**0.00E+00**
F24	4.13E-01	6.92E-02	2.81E-07	1.00E-24	2.95E-10	**0.00E+00**
F25	2.24E-02	2.51E-01	5.09E+00	1.16E-28	5.97E-05	**0.00E+00**
F26	2.50E-01	1.06E-01	2.16E+00	1.40E-02	9.08E-02	**0.00E+00**
F27	8.25E-01	1.09E+00	6.08E-02	2.16E-06	2.40E-02	**0.00E+00**
F28	2.50E-01	2.54E-03	6.01E-14	7.88E-47	1.01E-20	**0.00E+00**
F29	1.00E+04	1.00E+04	1.00E+04	1.00E+04	1.00E+04	**0.00E+00**
F30	9.61E+03	1.00E+04	1.15E+04	8.44E+03	1.01E+04	**0.00E+00**

В таблиці 3.8 видно середні результати мінімумів функцій для п'ятдесяти запусків. Також було для порівняння додано результати алгоритмів GA, PSO та SQP. Одразу можна побачити, що більшість кращих результатів припадає на EJS оптимізатор. Відносно GA, PSO та

SQP досліджувані алгоритми показали на порядок кращий результати. Проте можна помітити, що алгоритми CHC та TBO на функції F4 з двома глобальними мінімумами відпрацювали гірше, ніж усі інші алгоритми.

Таблиця 3.9 – Стандартні відхилення для 50 запусків

ID	GA	PSO	SQP	CHC	TBO	EJS
F01	1.10E-03	1.94E-02	4.00E-14	8.43E-48	2.84E-19	**0.00E+00**
F02	1.43E-03	9.20E-02	1.65E-13	2.65E-50	2.55E-19	**0.00E+00**
F03	6.83E-05	1.01E-06	2.80E-10	5.92E-83	2.68E-27	**0.00E+00**
F04	1.46E-01	2.93E-01	**1.48E-05**	6.26E-01	5.51E-01	1.14E+00
F05	1.17E-01	7.59E-07	1.57E-01	1.53E-02	1.72E-02	**0.00E+00**
F06	2.32E-01	4.01E-02	1.22E+00	**0.00E+00**	8.73E-06	**0.00E+00**
F07	1.33E-02	8.67E-02	7.94E-06	5.25E-27	6.72E-02	**0.00E+00**
F08	7.18E-01	4.49E-02	1.64E+00	**0.00E+00**	1.35E-08	**0.00E+00**
F09	3.78E-02	8.99E-02	1.93E+00	3.12E-02	8.18E-02	**0.00E+00**
F10	9.86E-01	2.70E+00	1.79E+01	**0.00E+00**	2.13E+00	**0.00E+00**
F11	3.15E-01	7.90E-02	**1.97E-02**	9.55E-02	8.87E-02	1.01E-01
F12	3.42E-01	1.06E-01	8.07E-02	**3.62E-02**	8.60E-02	6.38E-02
F13	4.00E+00	2.23E+00	1.36E+00	3.08E-01	7.25E-01	**3.00E-01**
F14	1.49E+02	2.08E+02	2.89E-11	5.14E-45	5.14E-16	**0.00E+00**
F15	2.31E+03	1.30E+03	2.39E-11	2.16E-50	1.22E-16	**0.00E+00**
F16	1.65E+03	3.71E+03	1.35E-10	2.02E-45	1.32E-14	**0.00E+00**
F17	9.50E+03	1.47E+03	**2.54E+02**	4.32E+03	1.96E+03	5.60E+03
F18	4.03E-01	2.21E-01	2.19E+01	1.15E-51	9.68E-18	**0.00E+00**
F19	5.12E-01	3.73E-01	2.65E-01	6.78E-02	1.43E-01	**0.00E+00**
F20	4.67E-03	1.82E-02	1.05E-13	1.96E-51	3.14E-19	**0.00E+00**
F21	1.49E+02	1.62E+02	2.92E+02	1.64E+02	**9.36E+01**	1.52E+02
F22	1.30E-04	4.42E-06	3.74E-12	2.63E-43	1.30E-14	**0.00E+00**
F23	1.24E-02	2.85E-02	5.73E-02	3.04E-02	1.43E-02	**0.00E+00**
F24	4.96E-01	6.56E-02	1.42E-07	3.62E-24	2.86E-10	**0.00E+00**
F25	1.50E-02	1.23E+00	1.37E+01	4.63E-28	2.08E-04	**0.00E+00**
F26	1.60E-01	2.37E-02	8.83E-01	3.50E-02	2.72E-02	**0.00E+00**
F27	1.54E-01	2.57E-01	4.52E-02	1.28E-06	3.94E-03	**0.00E+00**
F28	1.52E+00	5.37E-03	6.91E-14	4.23E-46	1.58E-20	**0.00E+00**
F29	**0.00E+00**	**0.00E+00**	9.51E-08	**0.00E+00**	**0.00E+00**	**0.00E+00**
F30	1.91E+03	2.58E+01	1.36E+03	3.72E+03	2.52E+01	**0.00E+00**

На основі результатів у таблиці 3.9 можна одразу сказати, що найстабільнішим виявився алгоритм EJS. Алгоритм CHC показав себе як другий по стабільності із досліджуваний методів, він продемонстрував кращу стабільність в 5 випадках. У багатьох випадках він був найстабільнішим, якщо не враховувати результати EJS. Результати TBO хоч і не були рекордними, але вони і не були найгіршими для жодної з функцій та для функції F21 з великою кількістю локальних мінімумів

показав приблизно в 10 разів краще значення, ніж інші методи.

Таблиця 3.10 – Медіанні значення для 50 запусків

ID	GA	PSO	SQP	CHC	TBO	EJS
F01	3.48E-04	6.70E-04	3.77E-16	2.84E-56	3.00E-20	**0.00E+00**
F02	4.27E-04	3.72E-03	1.59E-14	6.20E-55	8.00E-20	**0.00E+00**
F03	1.23E-05	7.91E-09	1.71E-15	1.71E-91	6.46E-29	**0.00E+00**
F04	1.56E-01	1.27E-01	**1.02E-06**	1.19E+00	1.37E+00	4.38E+00
F05	2.14E-01	6.38E-02	9.08E-01	**0.00E+00**	6.38E-02	**0.00E+00**
F06	2.59E-01	3.78E-02	5.68E+00	**0.00E+00**	3.08E-06	**0.00E+00**
F07	2.89E-03	8.94E-04	7.31E-07	3.98E-31	2.44E-04	**0.00E+00**
F08	2.80E-02	1.35E-02	1.92E+01	**4.44E-16**	2.63E-09	**4.44E-16**
F09	9.05E-05	1.41E-01	3.29E+00	**0.00E+00**	6.22E-02	**0.00E+00**
F10	6.30E-02	2.93E+00	2.24E+01	**0.00E+00**	5.99E-04	**0.00E+00**
F11	3.67E-01	1.73E-01	**3.72E-02**	5.36E-01	4.28E-01	6.83E-01
F12	4.29E-01	**2.43E-01**	4.99E-01	5.00E-01	2.82E-01	4.80E-01
F13	2.07E+00	1.88E+00	**5.29E-11**	2.72E+00	2.52E+00	2.40E+00
F14	1.58E+01	1.46E+00	4.69E-11	1.81E-52	1.77E-16	**0.00E+00**
F15	6.04E+00	4.27E-02	1.18E-12	3.90E-54	2.56E-17	**0.00E+00**
F16	2.31E+02	2.78E+02	5.35E-11	2.63E-50	4.19E-15	**0.00E+00**
F17	7.01E+02	1.60E+02	**1.13E-01**	3.66E+03	2.03E+03	7.00E+03
F18	1.12E-01	2.62E-02	7.55E+01	3.58E-58	1.02E-18	**0.00E+00**
F19	9.48E-01	7.07E-01	2.43E+00	**0.00E+00**	3.23E-01	**0.00E+00**
F20	7.04E-04	2.34E-03	1.18E-14	8.10E-56	5.98E-20	**0.00E+00**
F21	2.47E+02	**2.38E+02**	9.98E+02	7.19E+02	5.81E+02	7.05E+02
F22	9.14E-06	8.51E-09	9.93E-16	1.81E-46	7.81E-15	**0.00E+00**
F23	4.18E-02	9.14E-02	2.09E-01	9.16E-02	7.01E-02	**0.00E+00**
F24	1.56E-01	4.51E-02	2.77E-07	1.20E-26	1.86E-10	**0.00E+00**
F25	1.93E-02	4.86E-02	1.09E-01	1.15E-29	3.92E-07	**0.00E+00**
F26	2.00E-01	9.99E-02	2.50E+00	5.11E-24	9.99E-02	**0.00E+00**
F27	8.25E-01	1.08E+00	4.80E-02	1.78E-06	2.44E-02	**0.00E+00**
F28	5.72E-03	3.89E-04	4.06E-14	3.12E-52	4.46E-21	**0.00E+00**
F29	1.00E+04	1.00E+04	1.00E+04	1.00E+04	1.00E+04	**0.00E+00**
F30	1.00E+04	1.00E+04	1.13E+04	1.00E+04	1.01E+04	**0.00E+00**

У таблиці 3.10 можна побачити знайдені медіанні значення для кожної функції. Якщо порівняти з таблицею 3.8, то можна побачити, що знайдені медіанні значення не сильно відрізняються від середніх, що вкотре підтверджує здібність алгоритмів до стабільної роботи. У таблиці Б.3 наведено похибки знайдених точок мінімуму, знайденого значення мінімуму та інтегральна метрика якості відповідно. Можна помітити, що похибка по значенню мінімуму менша, ніж по відстані у просторі змінних у більшості випадках, а інтегральна метрика якості близька за значеннями до Δx. Можна сказати, що серед досліджуваних алгоритмів EJS та CHC виявилися найкращими на даних функціях. TBO показав

результати в деяких випадках на рівні SQP, в деяких – трохи краще.

У таблиці Б.4 позначено початкові параметри експериментів.

У цьому експерименті було виміряно такі метрики як середнє значення знайденого мінімуму та стандартне відхилення. А також було заміряно час роботи алгоритмів на кожній функції для того, щоб дізнатися, як впливає кладність функції на зміну тривалості роботи та порівняти один з одним. У таблиці 3.11 зображено рейтинги алгоритмів

Таблиця 3.11 – Середні ранги алгоритмів

MTDE	**YDSE**	**CHIO**	**WSO**	**HGS**	**CHC**	**TBO**	**PSO**
1.867	1.933	3.267	3.800	4.067	4.600	4.667	4.800
DE	**AVOA**	**SMA**	**WOA**	**SCA**	**EJS**	**RSO**	
5.467	5.467	5.533	7.533	9.333	10.133	11.067	

по результатам середніх значень з таблиць Б.6 та 3.12.

Таблиця 3.12 – Середні значення та стандартні відхилення (SMA–EJS)

Fn	**Stat**	**SMA**	**WOA**	**WSO**	**YDSE**	**CHC**	**TBO**	**EJS**
F1	Avg	1.58E+05	4.75E+06	1.64E+04	1.00E+02	5.15E+04	8.31E+04	6.59E+06
F1	Std	1.61E+05	4.54E+06	3.88E+04	0.00E+00	5.94E+04	3.45E+04	2.97E+06
F2	Avg	1.01E+08	2.01E+04	2.00E+02	2.00E+02	2.80E+03	7.26E+02	8.26E+08
F2	Std	1.06E+04	2.28E+04	1.36E+02	0.00E+00	4.64E+03	3.10E+02	3.74E+08
F3	Avg	3.02E+02	6.99E+03	3.57E+02	3.00E+02	3.56E+02	3.75E+02	3.52E+03
F3	Std	4.68E+02	3.70E+03	1.04E+02	0.00E+00	7.71E+01	3.08E+01	1.44E+03
F4	Avg	4.03E+02	4.12E+02	4.00E+02	4.00E+02	4.17E+02	4.01E+02	4.48E+02
F4	Std	8.45E+00	4.04E+01	2.04E+01	0.00E+00	1.57E+01	6.26E+00	1.28E+01
F5	Avg	5.20E+02	5.20E+02	5.19E+02	5.20E+02	5.20E+02	5.20E+02	5.20E+02
F5	Std	2.61E-02	8.17E+00	4.88E+00	9.59E-03	3.17E-02	7.50E-02	6.49E-02
F6	Avg	6.04E+02	6.06E+02	6.03E+02	6.01E+02	6.04E+02	6.02E+02	6.06E+02
F6	Std	1.23E+00	1.44E+00	1.49E+00	8.42E-01	1.55E+00	1.16E+00	8.51E-01
F7	Avg	7.00E+02	7.01E+02	7.01E+02	7.00E+02	7.00E+02	7.00E+02	7.12E+02
F7	Std	7.35E-02	3.88E-01	4.10E-01	1.22E-02	8.52E-02	6.57E-02	4.38E+00
F8	Avg	8.29E+02	8.00E+02	8.30E+02	8.02E+02	8.04E+02	8.14E+02	8.46E+02
F8	Std	4.28E+00	9.96E+00	3.22E+00	7.17E-01	1.50E+00	2.85E+00	7.11E+00
F9	Avg	9.11E+02	9.36E+02	9.08E+02	9.05E+02	9.27E+02	9.24E+02	9.45E+02
F9	Std	4.84E+00	1.32E+01	3.22E+00	1.93E+00	8.65E+00	3.01E+00	5.20E+00
F10	Avg	1.16E+03	1.68E+03	1.22E+03	1.04E+03	1.03E+03	1.37E+03	1.76E+03
F10	Std	8.28E+01	3.17E+02	1.16E+02	3.81E+01	1.61E+01	1.15E+02	1.99E+02
F11	Avg	1.78E+03	2.13E+03	1.37E+03	1.51E+03	1.70E+03	2.07E+03	2.21E+03
F11	Std	2.01E+02	2.93E+02	1.60E+02	1.63E+02	2.92E+02	1.51E+02	1.62E+02
F12	Avg	1.20E+03	1.20E+03	1.20E+03	1.20E+03	1.20E+03	1.20E+03	1.20E+03
F12	Std	9.15E-02	2.75E-01	1.94E-01	5.87E-02	1.48E-01	1.30E-01	2.01E-01
F13	Avg	1.30E+03	1.30E+03	1.30E+03	1.30E+03	1.30E+03	1.30E+03	1.30E+03
F13	Std	6.53E-02	1.68E-01	1.37E-01	2.39E-02	9.04E-02	3.82E-02	1.40E-01
F14	Avg	1.40E+03	1.40E+03	1.40E+03	1.40E+03	1.40E+03	1.40E+03	1.40E+03
F14	Std	4.59E-02	2.37E-01	1.82E-01	3.11E-02	7.00E-02	2.72E-02	1.94E-01
F15	Avg	1.51E+03	1.50E+03	1.50E+03	1.50E+03	1.50E+03	1.50E+03	1.51E+03
F15	Std	3.22E+00	6.79E+00	1.06E+00	1.69E-01	6.18E-01	2.82E-01	1.34E+00

Видно, що алгоритмом-лідером в контексті середніх мінімумів є

MTDE. Досліджувані алгоритми CHC, TBO та EJS посідають 6, 7 та 14 місце з розглянутих 15 алгоритмів, причому CHC та TBO показали результати приблизно однакової точності. У таблицях Б.6 та 3.12 видно, що усі алгоритми впоралися з оптимізацією функцій F5, F6, F12, F14 та F15, серед досліджуваних алгоритмів лідерували по черзі CHC та TBO. На функціях F4, F7-F9 загальні результати вже трохи гірші та алгоритм EJS вже видавав слабші результати. На F1-F3 більшість алгоритмів не знайшли оптимальне значення, але CHC та TBO відносно інших надали хороший результат, а у випадках F2-F3 TBO перевершив CHC. На функціях F10-F11 маємо, що з усіх трьох алгоритмів порівняно хороші результати видав лише CHC.

Таблиця 3.13 – Середні ранги по стандартному відхиленню

YDSE	**MTDE**	**TBO**	**CHIO**	**SMA**	**CHC**	**HGS**	**WSO**
1.933	4.000	4.333	4.467	6.867	7.267	7.867	8.200
SCA	**PSO**	**DE**	**EJS**	**AVOA**	**WOA**	**RSO**	
9.000	9.400	9.467	10.000	10.467	13.267	13.400	

У 3.13 бачимо, що TBO входить у трійку найстабільніших, CHC має стабільність краще, ніж в половини алгоритмів, а EJS є не найгіршим з усіх.

Таблиця 3.14 – Середній час виконання в секундах та стандартне відхилення для CHC, TBO, EJS на F1–F15

ID	**CHC**	**TBO**	**EJS**
F1	18.11 ± 0.64	22.92 ± 3.09	20.18 ± 0.55
F2	15.78 ± 0.44	20.93 ± 0.71	18.69 ± 0.72
F3	15.95 ± 0.59	20.77 ± 0.61	19.26 ± 1.79
F4	21.00 ± 0.96	29.65 ± 1.84	22.12 ± 0.75
F5	20.50 ± 0.78	27.84 ± 0.85	22.07 ± 1.10
F6	158.83 ± 2.91	208.55 ± 4.45	104.46 ± 2.37
F7	20.03 ± 0.72	24.79 ± 2.19	20.92 ± 0.73
F8	17.40 ± 0.49	21.29 ± 0.54	19.72 ± 0.99
F9	18.57 ± 0.62	24.27 ± 0.58	19.75 ± 0.63
F10	36.79 ± 0.70	46.95 ± 2.55	32.67 ± 0.72
F11	27.27 ± 0.52	34.53 ± 0.71	26.22 ± 0.55
F12	553.48 ± 2.12	682.79 ± 3.07	356.54 ± 8.41
F13	20.94 ± 0.63	28.32 ± 0.59	22.79 ± 0.53
F14	21.26 ± 0.68	29.33 ± 0.65	23.70 ± 0.97
F15	25.60 ± 0.51	33.59 ± 0.75	25.58 ± 0.58

У таблиці 3.14 можна бачити, що TBO працював значно довше за інші алгоритми, EJS виявився найшвидшим та найстабільнішим, а CHC переважав його в окремих випадках.

Для функцій F16-F30 було порівняно роботу алгоритмів CHC, TBO та EJS з GA, SCA, DA, SFO, TSA, ChOA та SHO [31].

Таблиця 3.15 – Параметри запуску алгоритмів на другому тестовому наборі на F16–F30

Розмір популяції	30
Кількість ітерацій	1000
Кількість запусків	30
Розмірність функцій	30

У таблиці 3.15 позначено початкові параметри експерименту.

У таблиці Б.5 продемонстровано результати алгоритмів на функціях F16–F30.

Таблиця 3.16 – Ранги по середнім мінімумам

SHO	2.667	**CHC**	3.333	**SCA**	4.067	**TBO**	4.200	**EJS**	4.400
DA	4.933	**TSA**	5.667	**ChOA**	6.000	**GA**	7.333	**SFO**	7.533

Таблиця 3.17 – Ранги по стандартному відхиленню

EJS	3.867	**SCA**	4.067	**CHC**	4.200	**TBO**	4.400	**SHO**	4.600
ChOA	5.400	**DA**	6.467	**SFO**	6.800	**TSA**	7.467	**GA**	7.667

У таблицях 3.16 та 3.17 показано ранги алгоритмів за середніми значеннями мінімумами та стандартним відхиленням. З таблиць можна зробити висновок, що на функціях F16-F30 CHC демонструє дуже високу ефективність та посідає 3 місце по стабільності з усіх досліджуваних алгоритмів. На функціях F17-F22 він був найкращим серед досліджуваних. EJS показав себе як найстабільніший з усіх алгоритм. Також мав найкраще серед досліджуваних алгоритмів на функціях F23,

F27–F30. Проте усі алгоритми входять у п'ятірку кращих алгоритмів серед усіх порівнюваних.

Таблиця 3.18 – Середній час виконання в секундах та стандартне відхилення для CHC, TBO, EJS на F16–F30

ID	CHC	TBO	EJS
F16	20.86 ± 0.19	22.28 ± 0.22	12.08 ± 0.18
F17	5.67 ± 0.07	5.72 ± 0.10	3.68 ± 0.06
F18	3.43 ± 0.06	3.17 ± 0.05	2.43 ± 0.05
F19	14.15 ± 0.19	14.77 ± 0.17	8.28 ± 0.20
F20	4.59 ± 0.16	4.63 ± 0.48	3.11 ± 0.12
F21	9.02 ± 0.11	9.32 ± 0.23	5.27 ± 0.10
F22	41.32 ± 0.22	45.67 ± 0.37	29.54 ± 0.15
F23	9.87 ± 0.17	10.02 ± 0.18	6.11 ± 0.12
F24	18.01 ± 0.13	19.15 ± 0.21	11.28 ± 0.11
F25	27.45 ± 0.14	30.02 ± 0.29	16.67 ± 0.13
F26	56.44 ± 0.32	68.45 ± 1.60	35.03 ± 0.13
F27	46.74 ± 0.10	50.74 ± 0.36	26.22 ± 0.08
F28	29.92 ± 0.09	32.22 ± 0.09	17.10 ± 0.06
F29	21.85 ± 0.08	23.46 ± 0.09	12.66 ± 0.06
F30	43.41 ± 0.14	47.26 ± 0.72	24.47 ± 0.11

У таблиці 3.18 можна бачити, що EJS є абсолютним лідером по часу виконання, майже вдвоє швидший за TBO, а CHC виконується майже за трохи менший час ніж, TBO.

3.5 Бінарна класифікація набору CIC-IDS2017

CIC-IDS2017 містить трафік мережі для розробки та оцінки систем виявлення вторгнень. Цей набір призначений для відображення трафіку, він включає понад 2,8 мільйона мережевих пакетів, зібраних протягом 7 днів у реальному мережевому середовищі. У наборі є як нормальний трафік, так і сім різних сценаріїв атак: Brute Force, Heartbleed, Botnet, DoS, DDoS, Port Scan, Web Attack і Infiltration. Набір даних є сильно незбалансованим. Більшість записів належать до класу «Benign», тобто нормального трафіку, і лише відносно невелика кількість записів – до інших класів. Набір складається з 2 830 743 рядків і 79 стовпців. Із них 78 стовпців – числові ознаки, а «label» – категоріальний стовпець.

Хід експерименту включав наступні кроки:

1) Завантаження у середовище `Kaggle Notebook`.

2) Очищення даних.

3) Аналіз наявних шаблонів у даних.

4) Попередня обробка даних.

5) Формування збалансованого набору даних для задачі бінарної класифікації.

6) Застосування метаевристичних алгоритмів `CHC`, `TBO` та `EJS` для задачі бінарної класифікації.

7) Порівняння отриманих результатів з результатами традиційних методів машинного навчання, зокрема логістичної регресії (з `SAGA solver`) та методу опорних векторів (`SVM`).

3.5.1 Завантаження у середовище Kaggle Notebook

Kaggle Notebook – це безкоштовне середовище для роботи з Jupyter Notebook, яке може використовувати GPU, яке дозволяє виконувати операції машинного навчання на хмарних серверах [7]. Kaggle дозволяє безкоштовно завантажити будь-який набір даних, у тому числі і CIC-IDS2017.

3.5.2 Очищення даних

На початковому етапі обробки було виявлено та видалено з датасету 308381 дублікатів. Крім того, було знайдено 353 пропущених значення. Після додаткової перевірки на наявність нескінченних значень загальна кількість пропущених значень становила 3128.

Також було виконано базовий статистичний аналіз для ключових ознак. Медіанне значення для ознаки `Flow Bytes/s` становить 3715,04, а для `Flow Packets/s` – 69,74, що дозволяє скласти уявлення про типові потоки у трафіку мережі. Пропущені значення було імп'ютовано медіанними значеннями.

3.5.3 Аналіз наявних шаблонів

Мітки було згруповано за типами трафіку, для спрощення аналізу закономірностей, результати наведено у таблиці 3.19.

Таблиця 3.19 – Відповідність міток класів типам трафіку

Label	**Class Name**
0	BENIGN
1	Bot
2	Brute Force
3	DDoS
4	DoS
5	Heartbleed
6	Infiltration
7	Port Scan
8	Web Attack

Для оцінки зв'язків між ознаками побудовано кореляційну матрицю та виявлено 32 позитивно корельованих ознаки, які можуть бути корисними для прогнозування. Для зменшення обчислювальних витрат перевірено лише 20% вибірки на репрезентативність. На репрезентативність досліджували 79 ознак, 14 з яких показали відхилення більше 5%, як показано на рисунку 3.7. Такий відсоток відхилення може вказувати на порушення репрезентативності вибірки. Також було визначено, що вибірка містила великий відсоток викидів, які і були причиною збільшення варіативності.

На рисунку 3.8 можна бачити, що викиди перевищують 20% для багатьох ознак.

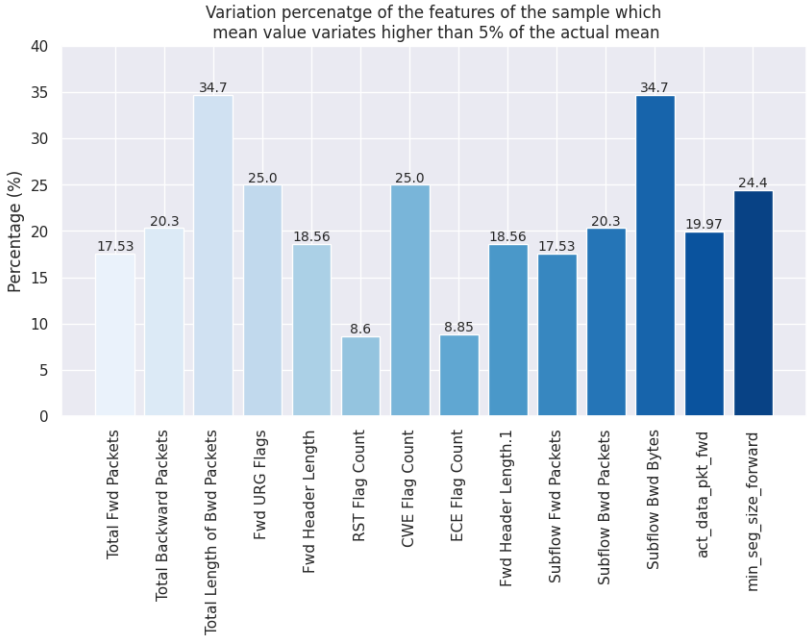


Рисунок 3.7 – Відсоток варіації ознак вибірки, середнє значення яких відхиляється більше ніж на 5% від фактичного середнього значення

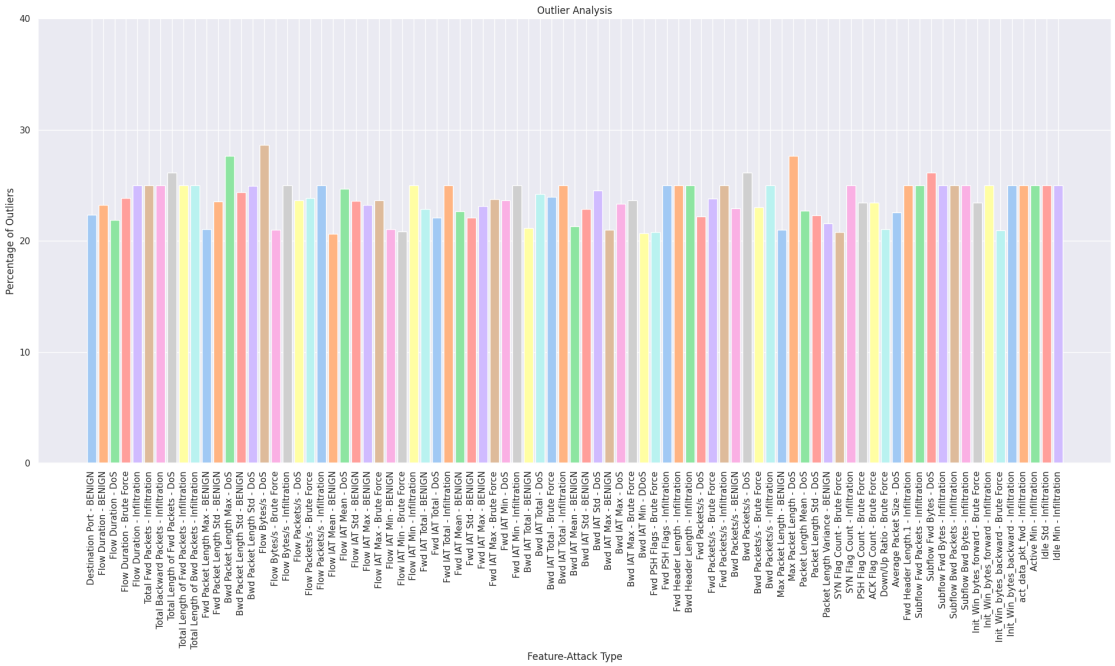


Рисунок 3.8 – Графік відсотку викидів для ознак

3.5.4 Попередня обробка даних

Для покращення продуктивності для типі даних з плаваючою комою float64 ми зменшили розрядність до float32, бо нема необхідності зберігати подвійну точність, що дозволило заощадити використання пам'яті на 47,5%.

Стовпці з нульовим стандартним відхиленням мають однакове значення у всіх рядках. Через відсутність варіації вони не допомагають відрізняти класи або групи даних та не майже не мають зв'язку з іншими ознаками, тож їх було видалено.

Для відбору ознак було використано метод головних компонент PCA. Він дозволяє виділити найінформативніші ознаки з великих наборів даних, зберігаючи при цьому найбільш релевантну інформацію з початкового датасету. Це зменшує складність моделі, оскільки додавання кожної нової ознаки негативно впливає на її продуктивність – це явище також відоме як «прокляття розмірності» [11].

У цьому випадку ми застосували Incremental PCA. Цей варіант кращий тим, що дозволяє обробляти великий обсяг даних по частинах без перевантаження RAM-пам'яті.

3.5.5 Створення збалансованого набору даних для бінарної класифікації

Набір даних є сильно незбалансованим, це проблема, бо більшість методів машинного навчання демонструють на меншому класі низьку продуктивність. Один із підходів до вирішення проблеми незбалансованих наборів даних – надмірна вибірка для класу меншин. Найпростіший підхід передбачає дублювання прикладів у класі меншин, хоча ці приклади не додають жодної нової інформації до моделі. Натомість, нові приклади можна синтезувати з існуючих. Це тип доповнення даних для

класу меншин, який називається технікою синтетичної надмірної вибірки для меншин [4], його і було використано.

Таблиця 3.20 – Розподіл даних за типами атак

Attack Type	**Count**
1	7510
0	7490

У таблиці 3.20 можна бачити, як було поділено дані для бінарної класифікації. Тут 0 – нормальний трафік, а 1 – будь-яка атака. Далі було цей набір розділено на тренувальний та тестовий набір в пропорціях 75 до 25 відповідно.

3.5.6 Застосування CHC, TBO та EJS для бінарної класифікації та порівняння результатів з іншими методами

Для реалізації запуску було обрано підхід до навчання логістичної регресії. Маємо функцію, оцінює класифікаційну похибку логістичної регресії для даного набору ваг $\mathbf{w}$.

Щоб прийняти рішення щодо окремого прикладу – після того, як ми навчили ваги під час тренування – класифікатор спершу множить кожну ознаку x_i на відповідну вагу w_i, підсумовує зважені ознаки та додає зсув b. Отримане число z виражає зважену суму доказів на користь певного класу [13]:

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b. \tag{3.1}$$

За допомогою (3.1) матриця ознак $\mathbf{X}_{\text{train}}$ перетворюється на вектор z, після чого застосовується сигмоїдна функція

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

щоб отримати прогнозовані ймовірності $\hat{y} = \sigma(z)$. Далі ймовірності переводяться у бінарні класи:

$$\tilde{y} = \begin{cases} 1, & \hat{y} \geq 0.5, \\ 0, & \hat{y} < 0.5. \end{cases}$$

Щоб вирахувати міру якості, вводимо метрику *accuracy* – частку правильно класифікованих прикладів:

$$\text{accuracy} = \frac{|\{i : \tilde{y}_i = y_i\}|}{N},$$

де N — кількість тренувальних прикладів.

Функція повертає $1 - \text{accuracy}$, тобто емпіричну частоту помилок. Тобто функцію на Python розроблено так, щоб оптимізатори мінімізовували результат. Далі на цій функції було запущно CHC, TBO та EJS по черзі. У таблиці 3.21 бачимо початкові параметри запуску. Було

Таблиця 3.21 – Початкові параметри запуску алгоритмів для класифікації

Кількість ітерацій	15000
Розмір популяції	50

обрано 15000 ітерацій, бо LR (методом Saga) виконувався за таку кількість ітерацій.

На 3.9 видно отримані результати точності. Видно, що TBO показав найкращий результат.

Для порівняння було використано результати роботи SVM та логістичної регресії з використанням методу SAGA з github-репозиторію [19]. Ці методи містяться в бібліотеці sklearn та є страндартними для

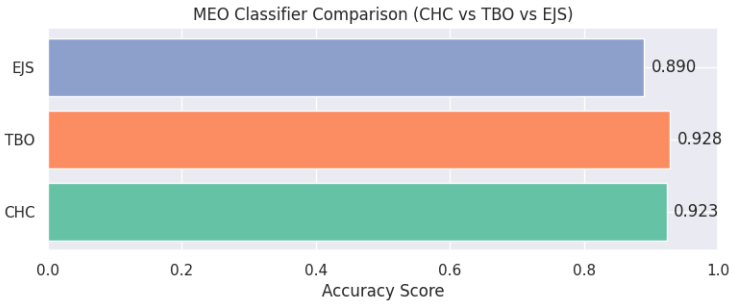


Рисунок 3.9 – Графіки точності класифікації для CHC, TBO та EJS

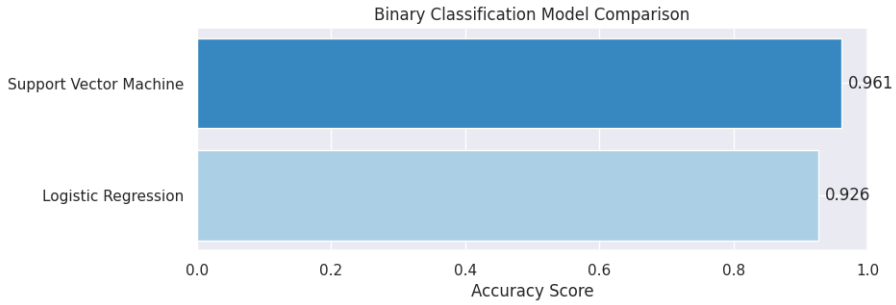


Рисунок 3.10 – Графіки точності класифікації для SVM та логістичної регресії (Saga метод)

вирішення задачі класифікації. Видно, що серед усіх SVM є найточнішим, а серед ройових алгоритмів найточнішим виявився TBO. По результатам точності TBO виявився кращим за SAGA на 0.2%. Варто зазначити, що усі ройові алгоритми показати точність >85%, що є хорошим результатом.

Також було побудовано ROC-криві і розраховано площу під ними (AUC).

У табл. 3.11 і 3.12 бачимо отримані результати. А в таблиці 3.22 – їхню інтерпретацію [5].

Таблиця 3.22 – Значення площі під кривою (AUC) та їх інтерпретація

Значення AUC	**Інтерпретація**
$0.9 \leqslant$ **AUC**	Відмінна
$0.8 \leqslant$ **AUC** < 0.9	Значна
$0.7 \leqslant$ **AUC** < 0.8	Задовільна
$0.6 \leqslant$ **AUC** < 0.7	Слабка
$0.5 \leqslant$ **AUC** < 0.6	Низька

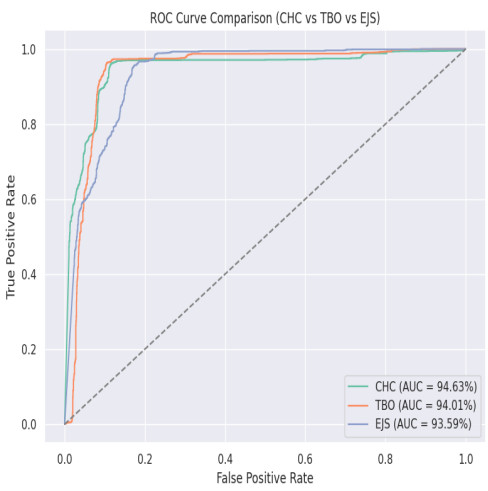


Рисунок 3.11 – ROC для CHC, TBO та EJS

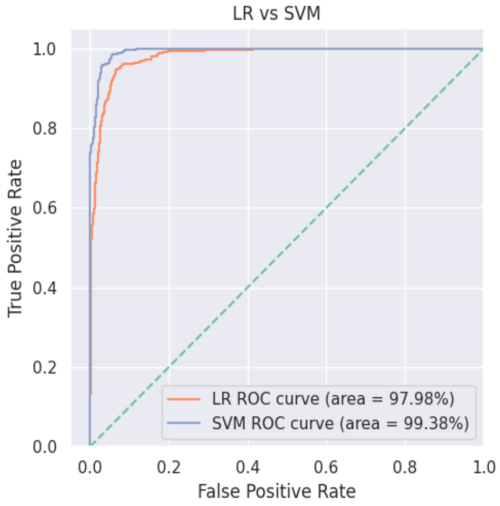
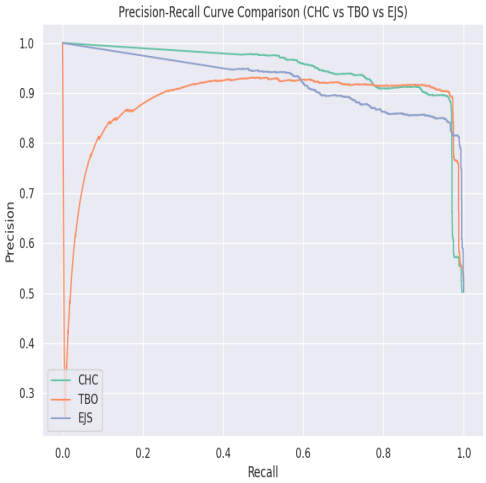


Рисунок 3.12 – ROC для SVM та LR (методом Saga)

По таблиці 3.22 кожне отримане значення можна інтерпретувати як відмінне. Тут бачимо, що найкращий AUC відсоток має CHC, TBO має відсоток на 0,62% нижче та EJS ще не 1,04% нижче. Отже краще за всіх серед ройових алгоритмів визначає CHC. Проте LR(Saga) та SVM показали значно кращі результати, майже наближені до 100%.

Далі досліджували таку метрику як точність-відтворювання. В інформаційному пошуку точність – це міра частки релевантних елементів серед фактично повернутих елементів, тоді як пригадування – це міра частки елементів, які були повернуті, серед усіх елементів, які мали бути повернуті. Крива «точність-відповідність» показує компроміс між точністю і відповідністю для різних порогових значень. Висока площа під кривою свідчить як про високу точність, так і про високий рівень запам'ятовування. Висока точність досягається за рахунок малої кількості хибнопозитивних результатів, а високий рівень пригадування – за рахунок малої кількості хибнонегативних результатів. Високі значення обох показників свідчать про те, що класифікатор повертає точні результати, а також про те, що він повертає більшість релевантних результатів [18].

На рисунках 3.13 та 3.14 бачимо криві «точності-відповідності».



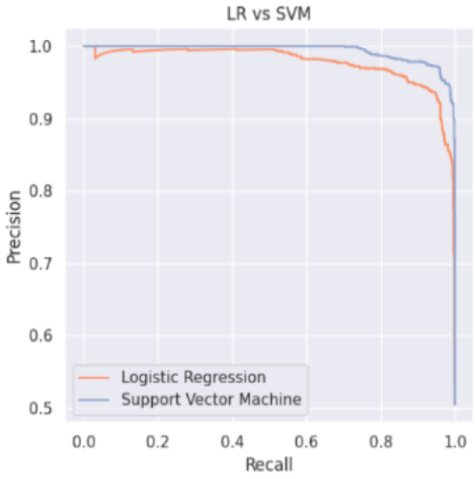


Рисунок 3.13 – PR для CHC, TBO та EJS

Рисунок 3.14 – PR для SVM та LR (методом Saga)

Одразу видно, що наші алгоритми демонструють не таку високу точність як SVM та LR (методом Saga). На низьких значеннях відповідностей, близьких до нуля, можна спостерігати, що TBO має нестабільну поведінку, але ближче до максимального він вже демонструє кращу точність серед ройових алгоритмів. CHC та EJS продемонстрували достатньо стабільну поведінку.

У матриці плутанини стовпці зазвичай представляють передбачені значення певного класу, а рядки – фактичні значення або навпаки. Ця таблична структура є зручним інструментом для візуалізації точності класифікації моделі, оскільки вона відображає кількість правильних і неправильних передбачень для всіх класів одночасно.

Стандартна форма матриці плутанини для бінарного класифікатора може виглядати так:

У верхньому лівому боксі вказується кількість істинно позитивних передбачень – кількість правильних передбачень для позитивного класу. Під ним – кількість хибно позитивних передбачень, тобто випадки негативного класу, які були помилково віднесені до позитивного. Їх також називають помилками першого роду. У верхньому правому боксі міститься кількість хибно негативних передбачень, тобто випадки

позитивного класу, які були помилково класифіковані як негативні. У нижньому правому боксі знаходиться кількість істинно негативних передбачень, тобто правильні передбачення для негативного класу. Сума всіх цих значень дорівнює загальній кількості передбачень, зроблених моделлю [10]. На рисунках 3.15–3.17 видно, матрицю плутанини для

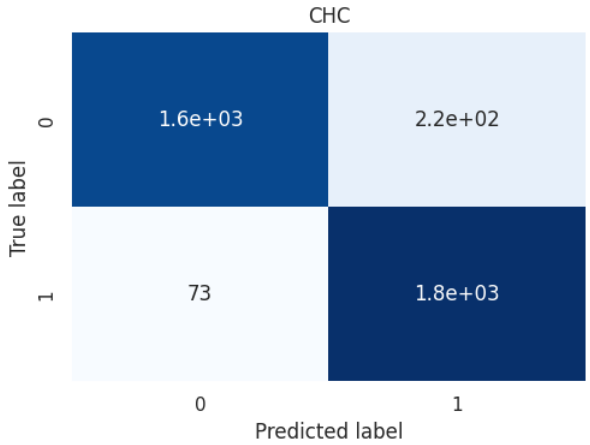


Рисунок 3.15 – Матриця плутанини для CHC

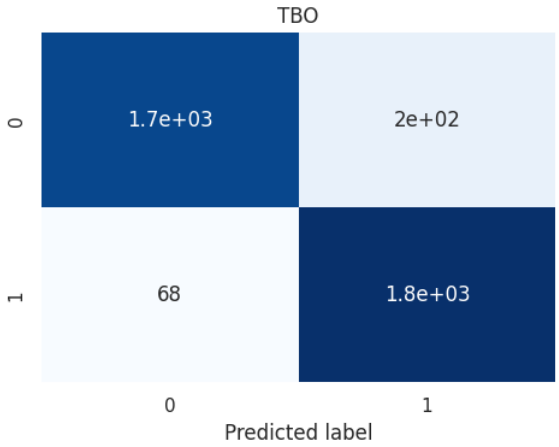


Рисунок 3.16 – Матриця плутанини для TBO

CHC, TBO та EJS. TBO демонструє найкращу збалансовану продуктивність серед цих трьох алгоритмів. CHC демонструє подібний рівень балансу, але трохи гірше за TBO. EJS має найбільше помилкових позитивів, він найчастіше помиляється, коли передбачає атаку, але це нормальний трафік.

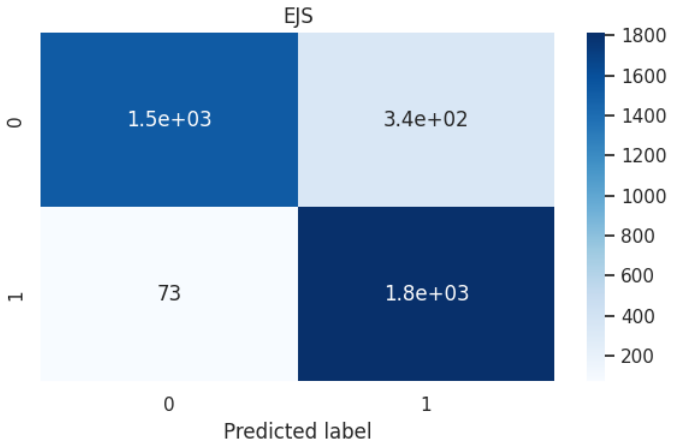


Рисунок 3.17 – Матриця плутанини для EJS

Висновки до розділу 3

З експериментів можна бачити, що в порівнянні із алгоритмами такими як GWO, RSA, WOA, GA і т. п. наші алгоритми здебільшого показали кращі результати на протестованих наборах функцій. Алгоритми показали високу швидкість збіжності до мінімумів та хороший рівень стабільності. Найкращу збіжність показав алгоритм CHC, також він продемонстрував високі здібності уникати локальні мінімуми. TBO показав трохи гіршу стабільність, але був найкращим на функціях з довгим шляхом до глобального мінімуму. Також алгоритм TBO був найповільнішим. Алгоритм EJS в свою чергу виявився найшвидшим, але найменш надійним в порівнянні з іншими досліджуваними алгоритмами. Усі три алгоритми показали достатньо високу точність на класифікації датасету, тож вони можуть конкурувати з такими оптимізаторами як SAGA у цьому аспекті. Проте досліджувані алгоритми мали і недоліки при розв'язку задачі класифікації такі як нижчий ніж у LR (SAGA) та SVM значення AUC, а також нестабільну точність алгоритму TBO.

ВИСНОВКИ

У даній бакалаврській роботі було проведено порівняльний аналіз таких ройових оптимізаторів як CHC, TBO та EJS. З метою визначення ефективності алгоритмів було виконано усі завдання досліджень та отримано відповіді на поставлені дослідницькі питання.

Спочатку було проведено огляд сучасних практичних задач, які потребують застосування ройового інтелекту для більш ефективного вирішення. На цьому етапі було встановлено, що для кожної оглянутої можна знайти підходящу метаевристику, яка б вирішувала оптимізаційну проблему для неї таким чином, щоб точність отриманих рішень задовольняла початкові умови.

Наступним кроком був огляд та аналіз публікацій за тематикою досліджень. На цьому етапі було розглянуто літературу де описано сучасні ройові методи, способи тестування нових алгоритмів, а також отримані результати тестування методів з оглянутих публікацій. Було також виявлено, що наразі існує тенденція покращувати існуючі методи та створювати гібриди алгоритмів. За результатами досліджень обидва ці підходи дають високі результати.

Далі стояло завдання ретельно проаналізувати алгоритми CHC, TBO та EJS. Було виявлено, що в публікації, де описано алгоритм CHC, не було наведено порівняння з алгоритмом CSO, який було взято автором для розробки CHC. Також було виявлено, що результати роботи з публікацій не порівняні одне з одним через неоднакові початкові умови експериментів. Отже, на цьому етапі було сформовано наступні дослідницькі питання:

1) Чи перевершує оптимізатор Cock-Chicken-Hen оригінальний Chicken Serach Optimizer за якістю результатів на стандартних функціях?

2) Які результати оптимізації демонструють досліджувані алгоритми на стандартних функціях?

3) Як ці три алгоритми показують себе у порівнянні з іншими відомими методами?

4) Який із трьох обраних алгоритмів є найефективнішим для вирішення практичної задачі?

На наступному етапі було проведено імплементацію та тестування досліджуваних алгоритмів. Було написано Python код за псевдокодом до кожного алгоритму. Алгоритми було протестовано спочатку на двох тестових наборах функцій, а потім їх було використано в якості оптимізаторів для методу логістичної регресії для класифікації датасету. У процесі запуску на функціях було виміряно середній мінімум протягом запусків, стандартне відхилення, медіанну збіжність та інші параметри оцінки роботи. Для оцінки якості класифікації датасету було зафіксовано такі параметри як точність, відношення точність-відтворення та AUC.

Останнім етапом був аналіз отриманих результатів та формулювання відповідей на дослідницькі питання.

1) Оптимізатор Cock-Chicken-Hen перевершує оригінальний Chicken Serach Optimizer за якістю результатів на стандартних функціях. CHC показав як кращу збіжність так і кращу стабільність. Винятками стали декілька функцій, проте на них обидва алгоритми показали погану ефективність, тож трохи кращий результат CSO в цих випадках не є показовим.

2) Найбільш універсальним алгоритм виявився CHC. Він виявився найстабільнішим та показав найвищу ефективність серед досліджуваних алгоритмів. Алгоритм TBO показав трохи нижчий, але теж показав високий рівень стабільності, а також хороші здібності досягати мінімуму, коли від алгоритму вимагається пройти довгий шлях до нього. Алгоритм EJS найкраще збігається на простих функціях, але іноді може вести себе неочікувано на складному шляху, що робить його найменш надійним.

3) У порівнянні з іншими алгоритмами в цілому досліджувані алгоритми показали здебільшого кращі результати. Особливо на простих функціях та в експериментах, де невелика кількість ітерацій. Але

алгоритми продемонстрували посередні результати відносно інших метаевристик в експерименті, де вимагалося брати 6000 ітерацій.

4) На задачі класифікації найкращим виявився алгоритм CHC. Хоч TBO і показав найвищу точність, яка навіть виявилася вищою за оптимізатор Saga на логістичній регресії, проте TBO виявився нестабільним та мав нижчу здібність розрізняти позитивні так негативні класи, ніж CHC. Загальна точність CHC лише на 0,05% нижча за точність TBO.

ПЕРЕЛІК ПОСИЛАНЬ

[1] Mohamed Abdel-Basset та ін. «Young's double-slit experiment optimizer: A novel metaheuristic optimization algorithm for global and constraint optimization problems». В: *Computer Methods in Applied Mechanics and Engineering* 403 (2023), с. 115652.

[2] Muhammad Firdaus Bin Azman, Azlan Mohd Zain та Nur Asyikin Mohamad Halimin. «Particle swarm optimization for optimal process parameters in injection molding». В: *Journal of Soft Computing and Decision Support Systems* 2.5 (2015), с. 10—15.

[3] A.D. Boursianis та ін. «Emerging Swarm Intelligence Algorithms and Their Applications in Antenna Design: The GWO, WOA, and SSA Optimizers». В: *Applied Sciences* 11.18 (2021), с. 8330. DOI: 10.3390/app11188330. URL: https://doi.org/10.3390/app11188330.

[4] Jason Brownlee. *SMOTE for Imbalanced Classification with Python*. Accessed: 2025-05-18. 2021. URL: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/.

[5] Şeref Kerem Çorbacıoğlu та Gökhan Aksel. «Receiver operating characteristic curve analysis in diagnostic accuracy studies: A guide to interpreting the area under the curve value». В: *Turkish journal of emergency medicine* 23.4 (2023), с. 195—198.

[6] David W. Corne та Michael A. Lones. *Evolutionary Algorithms*. arXiv:1805.11014 [cs.NE]. To appear in R. Martí, P. Pardalos, and M. Resende (eds.), *Handbook of Heuristics*, Springer. 2018. DOI: 10.48550/arXiv.1805.11014. URL: https://arxiv.org/abs/1805.11014.

[7] DataCamp. *Kaggle Datasets and Notebooks Tutorial.* https://www.datacamp.com/tutorial/tutorial-kaggle-datasets-tutorials-kaggle-notebooks. Accessed: 2025-05-16. 2023.

[8] Hu G. та ін. «EJS: Multi-Strategy Enhanced Jellyfish Search Algorithm for Engineering Applications». В: *Mathematics* (2023), с. 851. URL: https://doi.org/10.3390/math11040851.

[9] Ricardo García-Ródenas та ін. «A comparison of general-purpose optimization algorithms for finding optimal approximate experimental designs». В: *Computational Statistics & Data Analysis* 144 (2020). DOI: https://doi.org/10.1016/j.csda.2019.106844. URL: https://www.sciencedirect.com/science/article/pii/S0167947319301999.

[10] IBM. *What is a Confusion Matrix?* Accessed: 2025-05-19. n.d. URL: https://www.ibm.com/think/topics/confusion-matrix.

[11] IBM. *What Is Principal Component Analysis (PCA)?* Accessed: 2025-05-16. 2023. URL: https://www.ibm.com/think/topics/principal-component-analysis.

[12] Liang J.-J., Qu B.-Y. та Suganthan P. N. *Problem Definitions and Evaluation Criteria for the CEC 2014 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization.* Тех. звіт. Technical Report 201311. 2013.

[13] Daniel Jurafsky та James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models.* 3rd. Online manuscript released January 12, 2025. Dec, 2025. URL: https://web.stanford.edu/~jurafsky/slp3/.

[14] Gao Z. M. та Zhao J. «Cock-hen-chicken Optimizer: A Nature-inspired Algorithm for Real-world Engineering Optimization». В: *Computational Biomedicine* (2024).

[15] J. A. Tenreiro Machado, N. Özdemir та D. Baleanu. *Mathematical Modelling and Optimization of Engineering Problems*. Англ. 2020, с. 202 с. URL: https://link.springer.com/book/10.1007/978-3-030-37062-6.

[16] S. Masrom та ін. «Hybridization of Particle Swarm Optimization with Adaptive Genetic Algorithm Operators». В: *2013 13th International Conference on Intelligent Systems Design and Applications (ISDA)*. Salangor, Malaysia, 2013, с. 153—158. DOI: 10.1109/ISDA.2013.6920726.

[17] Nafiul Nawjis, mohammad Shafiul Alam та Mahzabeen Emu. «Hybridization of Evolutionary and Swarm Intelligence Algorithms for improved performance: A case study with TSP problem». В: *Proceedings of the International Conference on Computing Advancements*. ICCA 2020. Dhaka, Bangladesh: Association for Computing Machinery, 2020. ISBN: 9781450377782. DOI: 10 . 1145 / 3377049 . 3377060. URL: https://doi.org/10.1145/3377049.3377060.

[18] F. Pedregosa та ін. «Scikit-learn: Machine Learning in Python». В: *Journal of Machine Learning Research* 12 (2011), с. 2825—2830.

[19] Noushin Pervez. *Intrusion Detection (CIC-IDS2017)*. https : / / github . com / noushinpervez / Intrusion - Detection - CICIDS2017. GitHub repository. 2017. URL: https://github.com/noushinpervez/Intrusion-Detection-CICIDS2017.

[20] Noushin Pervez. *Intrusion Detection using CICIDS2017*. GitHub repository, Accessed: 2025-05-15. 2021. URL: https://github.com/noushinpervez/Intrusion-Detection-CICIDS2017.

[21] G. M. Platt, X.-S. Yang та A. J. Silva Neto. *Computational Intelligence, Optimization and Inverse Problems with Applications in Engineering*. Англ. Springer, 2019, с. 284 с. URL: https://link.springer.com/book/10.1007/978-3-319-96433-1.

[22] A Pourrousta та ін. «A multi-objective particle swarm optimization for production-distribution planning in supply chain network». В: *Management Science Letters* 2.2 (2012), с. 603—614.

[23] Franz Rothlauf. «Optimization Problems». В: *Design of Modern Heuristics: Principles and Application*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. DOI: 10.1007/978-3-540-72962-4_2. URL: https://doi.org/10.1007/978-3-540-72962-4_2.

[24] A. Saihood, M. A. Al-Shaher та M. A. Fadhel. «A New Tiger Beetle Algorithm for Cybersecurity, Medical Image Segmentation and Other Global Problems Optimization». Англ. В: *Mesopotamian Journal of CyberSecurity* 4.1 (2024), с. 17—46. DOI: 10.58496/MJCS/2024/003. URL: https://doi.org/10.58496/MJCS/2024/003.

[25] Yujie Tang та Steven H. Low. «Optimal placement of energy storage in distribution networks». В: *2016 IEEE 55th Conference on Decision and Control (CDC)*. 2016, с. 3258—3264. DOI: 10.1109/CDC.2016.7798759.

[26] Unaligned. *AI Algorithms and Swarm Intelligence*. https://www.unaligned.io/p/ai-algorithms-and-swarm-intelligence. Accessed: 2025-05-19. 2024.

[27] Plevris V. та Solorzano G. «A Collection of 30 Multidimensional Functions for Global Optimization Benchmarking». В: (2022), с. 46. URL: https://doi.org/10.3390/data7040046.

[28] G. Venter. «Review of Optimization Techniques». В: *Encyclopedia of Aerospace Engineering*. За ред. Richard Blockley та Wei Shyy. Chichester, UK: John Wiley та Sons, Ltd., 2010. URL: https://scholar.sun.ac.za/server/api/core/bitstreams/7a84c521-16b6-4844-8f2e-bfaf97296df4/content.

[29] Jie-Sheng Wang та Shu-Xia Li. «An Improved Grey Wolf Optimizer Based on Differential Evolution and Elimination Mechanism». В: *Scientific Reports* 9.1 (2019), с. 7181. ISSN: 2045-2322. DOI: 10 . 1038 / s41598 - 019 - 43546 - 3. URL: https://doi.org/10.1038/s41598-019-43546-3.

[30] Zhenlun Yang, Yunzhi Jiang та Wei-Chang Yeh. «Self-learning salp swarm algorithm for global optimization and its application in multi-layer perceptron model training». В: *Scientific Reports* 14.1 (2024), с. 27401. ISSN: 2045-2322. DOI: 10 . 1038 / s41598 - 024 - 77440 - 4. URL: https://doi.org/10.1038/s41598-024-77440-4.

[31] Shijie Zhao та ін. «Sea-horse optimizer: a novel nature-inspired meta-heuristic for global optimization problems». В: *Applied Intelligence* 53.10 (2023), с. 11833—11860.

[32] Zhuanzhe Zhao та ін. «A Modified Shuffled Frog Leaping Algorithm with Inertia Weight». В: *Scientific Reports* 14.1 (2024), с. 4146. ISSN: 2045-2322. DOI: 10.1038/s41598-024-51306-1. URL: https://doi.org/10.1038/s41598-024-51306-1.

[33] Фармацевтична енциклопедія. *Біореактор*. Accessed: 2025-05-21. n.d. URL: https://www.pharmencyclopedia.com.ua/article/1929/bioreaktor.

ДОДАТОК А ТЕКСТИ ПРОГРАМ

А.1 CHC алгоритм

```
import numpy as np
from scipy.special import gamma

def initialization(SearchAgents_no, dim, ub, lb):
    Positions = np.random.rand(SearchAgents_no, dim) * (np.array(ub)
    - np.array(lb)) + np.array(lb)
    return Positions

def levy_flight(d):
    beta = 1.5
    sigma = (
        gamma(1 + beta)
        * np.sin(np.pi * beta / 2)
        / (gamma((1 + beta) / 2) * beta * 2 ** ((beta - 1) / 2))
    ) ** (1 / beta)
    u = np.random.randn(d) * sigma
    v = np.random.randn(d)
    return u / (np.abs(v) ** (1 / beta))

def CHC(SearchAgents_no, Max_iter, lb, ub, dim, fobj):
    cocks, hens = 20, 5
    chickens = SearchAgents_no - cocks - hens

    positions_cocks = initialization(cocks, dim, ub, lb)
    positions_hens = initialization(hens, dim, ub, lb)
    positions_chickens = initialization(chickens, dim, ub, lb)
    Positions = np.vstack((positions_cocks, positions_hens, positions_chickens))

    fitness = np.array([fobj(pos) for pos in Positions])
    best_fit = np.min(fitness)
    best_position = Positions[np.argmin(fitness)]
    worst_position = Positions[np.argmax(fitness)]
    p1 = 0.97

    Convergence_curve = np.zeros(Max_iter)

    for it in range(Max_iter):
        for i in range(SearchAgents_no):
            Positions[i] = np.clip(Positions[i], lb, ub)
            fitness[i] = fobj(Positions[i])
            if fitness[i] < best_fit:
                best_fit = fitness[i]
                best_position = Positions[i]

        fit_index = np.argsort(fitness)
        position_cocks = Positions[fit_index[:cocks]]
        fit_cocks = fitness[fit_index[:cocks]]
        position_hens = Positions[fit_index[cocks:cocks+hens]]
        fit_hens = fitness[fit_index[cocks:cocks+hens]]
        position_chickens = Positions[fit_index[cocks+hens:]]
        fit_chickens = fitness[fit_index[cocks+hens:]]

        a = 2 - it * (2.0 / Max_iter)

        for i in range(cocks):
            current = position_cocks[i].copy()
            if np.array_equal(position_cocks[i], best_position):
                position_cocks[i] += np.random.uniform(-1, 1, dim) * 0.8
            else:
```

```
                LF = levy_flight(dim)
                position_cocks[i] = best_position * LF
            new_fit = fobj(position_cocks[i])
            if new_fit >= fit_cocks[i]:
                position_cocks[i] = current
            else:
                fit_cocks[i] = new_fit

        for i in range(hens):
            current = position_hens[i].copy()
            for j in range(dim):
                if np.random.rand() < 0.5:
                    position_hens[i][j] = best_position[j] +
                    a * np.random.rand() * (best_position[j] - position_hens[i][j])
                else:
                    k = np.random.randint(0, cocks)
                    position_hens[i][j] = position_cocks[k][j] +
                    a * np.random.rand() * (position_cocks[k][j] -
                    position_hens[i][j])
            new_fit = fobj(position_hens[i])
            if new_fit >= fit_hens[i]:
                position_hens[i] = current
            else:
                fit_hens[i] = new_fit

        for i in range(chickens):
            for j in range(dim):
                if np.random.rand() < 0.5:
                    if np.random.rand() < 0.5:
                        position_chickens[i][j] += a * np.random.rand() *
                        (position_chickens[i][j] - worst_position[j])
                    else:
                        position_chickens[i][j] += a * np.random.rand() *
                        (position_chickens[i][j] - best_position[j])
                else:
                    if np.random.rand() < p1:
                        k = np.random.randint(0, hens)
                        position_chickens[i][j] = position_hens[k][j]
                        + a * np.random.rand() *
                        (position_hens[k][j] - position_chickens[i][j])
                    else:
                        position_chickens[i][j] = np.random.rand() *
                        (ub[j] - lb[j]) + lb[j]

        Positions = np.vstack((position_cocks, position_hens, position_chickens))
        fitness = np.concatenate((fit_cocks, fit_hens, fit_chickens))

        Convergence_curve[it] = best_fit

    return best_fit, best_position, Convergence_curve
```

A.2 TBO алгоритм

```
def initialization(n_agents, dim, lb, ub):
    lb, ub = np.asarray(lb), np.asarray(ub)
    return np.random.uniform(lb, ub, (n_agents, dim))


def dig_holes(pos, R, SD):
    steps = np.where(np.random.rand(*pos.shape) > 0.5, 1, -1)
    return pos + steps * R * SD * np.random.rand(*pos.shape)
```

```python


def TBO(search_agents=30,
        max_iter=500,
        lb=-100, ub=100, dim=30,
        fobj=lambda x: np.sum(x**2),
        Hm=10, SD0=1.0, p=2):

    lb = np.asarray(lb if hasattr(lb, '__len__') else [lb]*dim, dtype=float)
    ub = np.asarray(ub if hasattr(ub, '__len__') else [ub]*dim, dtype=float)

    pos = initialization(search_agents, dim, lb, ub)
    fit = np.apply_along_axis(fobj, 1, pos)

    best_idx = np.argmin(fit)
    best_pos, best_fit = pos[best_idx].copy(), fit[best_idx]
    Convergence_curve = np.empty(max_iter)

    eps = 1e-12

    for t in range(max_iter):
        SD    = SD0 - (p/np.pi)*abs(np.arctan((t+1)*p/np.pi))
        alpha    = 1 - ((t+1)/max_iter)**2
        beta    = 1 - alpha
        Rtmp = 2*np.random.rand()

        worst_fit = fit.max() + eps

        for i in range(search_agents):
            ratio = fit[i] / (worst_fit + 1e-8)
            safe_exp = np.exp(np.clip(ratio, -50, 50))
            H_i = int(round((1 - safe_exp) * Hm))
            for _ in range(H_i):
                cand = dig_holes(pos[i], Rtmp, SD)
                cand = np.clip(cand, lb, ub)
                f_c  = fobj(cand)
                if f_c < fit[i]:
                    pos[i], fit[i] = cand, f_c

        B_bar   = pos.mean(axis=0)
        rnd_mat = np.random.rand(search_agents, dim)
        cand    = alpha*pos + beta*(best_pos - B_bar)*rnd_mat
        cand    = np.clip(cand, lb, ub)
        fit_c   = np.apply_along_axis(fobj, 1, cand)
        mask    = fit_c < fit
        pos[mask], fit[mask] = cand[mask], fit_c[mask]

        for i in range(search_agents):
            k, l = np.random.choice(search_agents, 2, replace=False)
            cand = pos[i] + np.random.rand(dim)*(pos[k] - pos[l])
            cand = np.clip(cand, lb, ub)
            f_c  = fobj(cand)
            if f_c < fit[i]:
                pos[i], fit[i] = cand, f_c

        best_idx = np.argmin(fit)
        if fit[best_idx] < best_fit:
            best_fit, best_pos = fit[best_idx], pos[best_idx].copy()

        Convergence_curve[t] = best_fit

    return best_fit, best_pos, Convergence_curve
```

A.3 EJS алгоритм

```
def logistic_init(n, dim, lb, ub, eta=3.9, burn=100):
    x = np.random.rand(n, dim)
    for _ in range(burn):
        x = eta * x * (1 - x)
    return lb + (ub - lb) * x

def sine_cosine_factors(t, T):
    w1 = 2 * np.sin((1 - t / T) * np.pi / 2)
    w2 = 2 * np.cos((1 - t / T) * np.pi / 2)
    return w1, w2

def local_escaping_operator(P_tmp, pop, best):
    n, dim = pop.shape
    u1, u2, u3 = np.random.rand(3)
    rho1 = np.random.rand()
    f1, f2 = np.random.rand(2)
    k = np.random.randint(0, n)
    r1, r2 = np.random.randint(0, n, size=2)
    p1, p2 = np.random.randint(0, n, size=2)
    term1 = f1 * (u1 * best - u2 * pop[k])
    term2 = f2 * rho1 * (u3 * (pop[p2] - pop[p1]))
    term3 = u2 * (pop[r1] - pop[r2]) / 2.0
    if np.random.rand() < 0.5:
        return P_tmp + term1 + term2 + term3
    else:
        return best + term1 + term2 + term3

def learning_strategy(P, lb, ub, p=0.5):
    P_opp = lb + ub - P
    mid = (lb + ub) / 2.0
    P_rand = np.random.uniform(low=mid, high=P_opp, size=P.shape)
    mask = np.random.rand(*P.shape) < p
    return np.where(mask, P, P_rand)

def repair(P, lb, ub):
    return np.clip(P, lb, ub)

def EJS(SearchAgents_no, Max_iter, lb, ub, dim, fobj, beta=0.5,
gamma=0.5, p_ls=0.5):
    lb = np.full(dim, lb) if np.isscalar(lb) else np.asarray(lb, dtype=float)
    ub = np.full(dim, ub) if np.isscalar(ub) else np.asarray(ub, dtype=float)
    P = logistic_init(SearchAgents_no, dim, lb, ub)
    fit = np.apply_along_axis(fobj, 1, P)
    best_idx = np.argmin(fit)
    best, best_fit = P[best_idx].copy(), fit[best_idx]
    curve = np.empty(Max_iter)
    for t in range(1, Max_iter + 1):
        C_t = abs((1 - t / Max_iter) * (2 * np.random.rand() - 1))
        mu = P.mean(axis=0)
        for i in range(SearchAgents_no):
            if C_t > 0.5:
                r2 = np.random.rand(dim)
                P_tmp = P[i] + r2 * (best - beta * r2 * mu)
                P_new = local_escaping_operator(P_tmp, P, best)
            else:
                if np.random.rand() > (1 - C_t):
                    r3 = np.random.rand(dim)
                    P_new = P[i] + gamma * r3 * (ub - lb)
                else:
                    w1, w2 = sine_cosine_factors(t, Max_iter)
                    j = np.random.randint(0, SearchAgents_no)
```

```
                    step = C_t * (P[j] - P[i])
                    P_new = w1 * (P[i] + step) + w2 * (best - P[i])
            P_new = learning_strategy(P_new, lb, ub, p=p_ls)
            P_new = repair(P_new, lb, ub)
            f_new = fobj(P_new)
            if f_new < fit[i]:
                P[i], fit[i] = P_new, f_new
                if f_new < best_fit:
                    best, best_fit = P_new.copy(), f_new
        curve[t-1] = best_fit
    return best_fit, best, curve
```

ДОДАТОК Б ПОРІВНЯЛЬНИЙ АНАЛІЗ РОБОТИ МЕТОДІВ Й АЛГОРИТМІВ РОЙОВОГО ІНТЕЛЕКТУ

Таблиця Б.1 – Нормалізована відстань у просторі змінних (Δx) для кожної функції

ID	GA	PSO	SQP	CHC	TBO	EJS
F01	4.17E-05	5.79E-05	4.34E-11	3.77E-31	3.87E-13	0.00E+00
F02	2.90E-05	7.83E-05	2.20E-10	1.17E-30	4.07E-13	0.00E+00
F03	1.02E-03	5.37E-04	6.45E-05	4.91E-19	7.19E-10	0.00E+00
F04	3.54E-02	3.54E-02	4.18E-02	1.33E-02	1.50E-02	1.27E-02
F05	4.55E-02	2.27E-02	2.74E-01	5.01E-11	2.27E-02	0.00E+00
F06	1.01E-03	4.46E-05	1.96E-01	7.79E-16	4.82E-11	0.00E+00
F07	6.80E-02	4.46E-03	2.11E-01	4.77E-32	3.33E-05	0.00E+00
F08	9.83E-05	4.93E-05	2.46E-01	3.18E-18	1.00E-11	0.00E+00
F09	4.63E-05	4.35E-02	2.56E-01	3.85E-11	1.59E-03	0.00E+00
F10	7.78E-04	6.28E-02	2.06E-01	1.72E-10	7.59E-05	7.91E-11
F11	2.05E-02	1.06E-02	3.04E-03	1.75E-02	1.14E-02	1.91E-02
F12	3.08E-02	2.24E-02	3.33E-02	3.33E-02	1.82E-02	3.03E-02
F13	3.95E-02	3.86E-02	3.26E-07	6.31E-02	7.12E-02	8.84E-02
F14	1.55E-04	8.67E-04	3.84E-10	1.02E-29	8.82E-12	0.00E+00
F15	4.31E-05	3.04E-04	2.43E-10	3.65E-30	9.21E-12	0.00E+00
F16	6.11E-05	7.89E-03	1.42E-10	1.01E-28	2.80E-11	0.00E+00
F17	2.55E-02	2.62E-02	2.12E-02	4.95E-02	5.24E-02	5.01E-02
F18	4.29E-04	6.98E-05	2.31E-01	9.10E-32	2.89E-12	0.00E+00
F19	6.07E-02	3.53E-02	2.95E-01	1.07E-11	1.03E-02	0.00E+00
F20	3.67E-05	7.00E-05	1.53E-10	4.18E-31	3.34E-13	0.00E+00
F21	3.24E-01	4.58E-01	5.28E-01	4.48E-01	3.93E-01	3.91E-01
F22	7.99E-04	5.23E-04	6.67E-05	1.06E-18	7.08E-10	0.00E+00
F23	9.77E-02	2.39E-01	4.95E-01	9.86E-02	9.43E-02	0.00E+00
F24	5.48E-04	1.52E-04	8.23E-10	3.47E-29	6.44E-13	0.00E+00
F25	2.27E-05	5.88E-05	2.10E-04	1.44E-32	6.34E-10	0.00E+00
F26	2.23E-02	1.12E-02	2.79E-01	5.71E-25	1.12E-02	0.00E+00
F27	3.25E-05	1.59E-04	5.89E-05	3.35E-16	3.39E-12	0.00E+00
F28	1.58E-03	4.14E-04	3.74E-09	2.93E-28	1.24E-12	0.00E+00
F29	3.87E-01	4.46E-01	3.34E-01	3.80E-01	3.82E-01	0.00E+00
F30	2.21E-01	2.28E-01	4.72E-01	2.02E-01	1.98E-01	0.00E+00

З таблиць Б.2 та Б.1 видно, що результати усіх трьох досліджуваних алгоритмів в більшості переважаються результати GA, PSO та SQP, а це

Таблиця Б.2 – Нормалізована помилка функції (Δf) для кожної функції

ID	GA	PSO	SQP	CHC	TBO	EJS
F01	1.35E-05	2.21E-07	4.10E-11	6.86E-61	7.25E-25	0.00E+00
F02	1.23E-05	1.31E-07	1.71E-10	4.93E-60	6.36E-25	0.00E+00
F03	6.05E-04	3.61E-04	1.45E-03	1.61E-97	6.06E-35	0.00E+00
F04	3.91E-02	3.55E-02	4.24E-02	1.01E-07	1.17E-07	3.73E-07
F05	6.45E-02	3.22E-02	2.91E-01	0.00E+00	6.38E-02	0.00E+00
F06	6.73E-03	1.11E-07	1.88E-01	0.00E+00	1.86E-07	0.00E+00
F07	5.13E-02	7.15E-02	2.10E-01	1.04E-32	6.36E-06	0.00E+00
F08	3.82E-05	1.88E-07	2.92E-01	2.00E-17	1.19E-10	2.00E-17
F09	1.39E-05	2.78E-02	2.54E-02	0.00E+00	5.30E-03	0.00E+00
F10	3.07E-02	8.69E-02	2.50E-01	0.00E+00	3.47E-06	0.00E+00
F11	3.52E-02	1.50E-02	1.08E-02	3.05E-03	2.44E-03	3.89E-03
F12	4.45E-02	2.87E-02	3.33E-02	4.69E-04	2.65E-04	4.50E-04
F13	3.37E-02	4.19E-02	2.44E-07	2.51E-04	2.32E-04	2.21E-04
F14	1.89E-04	7.21E-06	3.37E-10	1.76E-62	1.73E-26	0.00E+00
F15	3.57E-05	4.31E-07	2.64E-10	3.90E-64	2.56E-27	0.00E+00
F16	3.14E-05	3.29E-05	1.96E-10	7.20E-61	1.15E-25	0.00E+00
F17	4.45E-02	7.01E-02	7.28E-02	5.22E-13	2.90E-13	9.98E-13
F18	1.34E-03	3.97E-05	2.56E-01	6.69E-63	1.90E-23	0.00E+00
F19	1.09E-01	1.76E-01	2.96E-01	0.00E+00	9.11E-02	0.00E+00
F20	1.23E-05	1.55E-07	1.44E-10	6.30E-61	4.65E-25	0.00E+00
F21	3.34E-01	5.99E-01	5.07E-01	2.05E-01	1.65E-01	2.00E-01
F22	4.26E-04	3.27E-05	9.37E-05	1.74E-49	7.49E-18	0.00E+00
F23	1.84E-01	4.22E-01	4.74E-01	4.73E-05	3.62E-05	0.00E+00
F24	3.55E-03	5.72E-05	6.30E-10	1.20E-28	1.86E-12	0.00E+00
F25	1.12E-05	1.24E-07	5.75E-02	1.70E-39	5.78E-17	0.00E+00
F26	2.37E-02	1.58E-02	2.57E-01	8.71E-25	1.70E-02	0.00E+00
F27	1.20E-05	6.64E-06	6.25E-05	1.69E-08	2.31E-04	0.00E+00
F28	7.67E-03	6.42E-04	3.93E-09	1.57E-59	2.25E-28	0.00E+00
F29	3.25E-01	4.04E-01	2.88E-01	5.87E-01	5.87E-01	0.00E+00
F30	2.46E-01	2.51E-01	3.34E-01	2.39E-01	2.40E-01	0.00E+00

свідчить про перспективність цих методів. Проте слід мати на увазі, що перший тестовий набір мав недостатньо високу складність обчислень, щоб однозначно визначити міру ефективності кожного з алгоритмів. Тож наступним етапом дослідження (табл. Б.3–Б.6) було запустити алгоритми на другому тестовому наборі, де кожна з досліджуваних функцій була ускладнена оберненнями та зсувами.

Таблиця Б.3 – Інтегральна метрика якості (Δt)

ID	GA	PSO	SQP	CHC	TBO	EJS
F01	1.03E-09	2.76E-13	9.48E-21	2.66E-31	2.74E-13	0.00E+00
F02	6.26E-10	6.11E-14	1.18E-19	8.29E-31	2.88E-13	0.00E+00
F03	1.51E-18	1.63E-28	1.68E-23	3.47E-19	5.08E-10	0.00E+00
F04	3.95E-09	1.75E-11	1.04E-13	9.38E-03	1.06E-02	8.97E-03
F05	5.22E-01	2.14E-01	9.57E-01	3.54E-11	4.79E-02	0.00E+00
F06	2.06E-02	4.87E-05	3.93E-01	5.51E-16	1.31E-07	0.00E+00
F07	1.25E-05	7.94E-08	2.11E-08	3.46E-32	2.40E-05	0.00E+00
F08	4.68E-02	2.23E-06	8.85E-01	1.43E-17	8.43E-11	1.42E-17
F09	3.37E-07	4.43E-03	3.50E-03	2.72E-11	5.02E-03	0.00E+00
F10	3.10E-02	2.79E-02	2.05E-01	1.22E-10	5.37E-05	5.59E-11
F11	6.58E-03	1.58E-03	8.46E-04	1.26E-02	8.22E-03	1.37E-02
F12	5.56E-04	2.41E-04	3.11E-04	2.36E-02	1.28E-02	2.14E-02
F13	7.01E-07	1.04E-06	1.04E-17	4.46E-02	5.04E-02	6.25E-02
F14	1.39E-10	2.65E-14	1.67E-21	7.18E-30	6.24E-12	0.00E+00
F15	5.52E-11	1.81E-17	3.50E-23	2.58E-30	6.51E-12	0.00E+00
F16	2.72E-10	1.69E-13	2.37E-22	7.15E-29	1.98E-11	0.00E+00
F17	5.51E-21	6.46E-22	2.41E-21	3.50E-02	3.70E-02	3.54E-02
F18	5.42E-04	1.21E-05	2.15E-01	6.43E-32	2.05E-12	0.00E+00
F19	3.99E-01	4.35E-01	7.70E-01	7.59E-12	6.51E-02	0.00E+00
F20	5.84E-10	1.08E-13	8.07E-20	2.95E-31	2.36E-13	0.00E+00
F21	9.19E-02	1.50E-01	2.88E-01	3.47E-01	2.98E-01	3.08E-01
F22	2.07E-13	1.42E-21	2.95E-20	7.51E-19	5.00E-10	0.00E+00
F23	3.08E-08	9.47E-08	9.46E-08	6.98E-02	6.67E-02	0.00E+00
F24	1.13E-02	1.74E-04	2.32E-09	8.84E-29	1.39E-12	0.00E+00
F25	7.77E-22	1.18E-23	3.54E-18	1.02E-32	4.48E-10	0.00E+00
F26	4.00E-02	2.67E-02	4.34E-01	7.36E-25	1.44E-02	0.00E+00
F27	6.90E-03	2.99E-03	5.80E-04	1.20E-08	1.64E-04	0.00E+00
F28	1.65E-10	1.24E-12	4.40E-23	2.07E-28	8.73E-13	0.00E+00
F29	9.65E-01	9.65E-01	9.65E-01	4.95E-01	4.95E-01	0.00E+00
F30	2.37E-01	2.37E-01	2.37E-01	2.22E-01	2.20E-01	0.00E+00

Таблиця Б.4 – Параметри запуску алгоритмів на другому тестовому наборі на F1–F15

Розмір популяції	40
Кількість ітерацій	6000
Кількість запусків	30
Розмірність функцій	10

Таблиця Б.5 – Середні значення та стандартні відхилення результатів (GA-EJS) для F16–F30

Fn	Stat	GA	SCA	DA	SFO	TSA	ChOA	SHO	CHC	TBO	EJS
F16	Avg	1.61E+03	1.61E+03	1.61E+03	1.61E+03	1.61E+03	1.61E+03	1.61E+03	1.61E+03	1.61E+03	1.61E+03
F16	Std	3.67E-01	2.63E-01	3.05E-01	1.77E-01	3.13E-01	2.59E-01	3.70E-01	3.47E-01	2.13E-01	2.35E-01
F17	Avg	5.28E+07	1.30E+07	1.41E+07	4.21E+07	1.34E+07	3.44E+07	1.21E+07	4.85E+06	6.87E+06	3.22E+07
F17	Std	3.02E+07	6.76E+06	1.37E+07	2.03E+07	1.20E+07	1.92E+07	8.78E+06	5.44E+06	2.73E+06	1.39E+07
F18	Avg	1.39E+09	3.27E+08	2.54E+07	1.23E+09	5.85E+08	8.51E+08	3.81E+07	1.23E+06	9.65E+07	6.58E+08
F18	Std	6.05E+08	1.80E+08	6.27E+07	6.59E+08	1.11E+09	1.12E+09	4.42E+07	5.40E+06	3.11E+07	3.70E+08
F19	Avg	2.19E+03	2.04E+03	2.00E+03	2.16E+03	2.10E+03	2.18E+03	2.03E+03	9.34E+03	7.25E+05	2.06E+07
F19	Std	9.37E+01	2.79E+01	7.54E+01	5.82E+01	1.19E+02	1.28E+02	5.07E+01	3.25E+04	5.02E+05	2.19E+07
F20	Avg	2.84E+05	4.64E+04	1.63E+05	2.40E+05	5.26E+04	1.13E+05	3.62E+04	9.51E+05	3.19E+08	1.79E+11
F20	Std	3.28E+05	2.66E+04	1.45E+05	1.81E+05	6.36E+04	4.91E+04	1.55E+04	4.11E+06	2.12E+08	3.21E+11
F21	Avg	1.80E+07	3.64E+06	4.63E+06	1.47E+07	7.06E+06	1.27E+07	1.77E+06	8.05E+05	6.88E+06	2.22E+08
F21	Std	1.17E+07	2.85E+06	5.82E+06	1.07E+07	1.11E+07	2.07E+06	2.29E+06	8.50E+05	2.63E+06	8.85E+07
F22	Avg	3.79E+03	3.27E+03	3.32E+03	4.14E+03	3.56E+03	3.20E+03	2.96E+03	4.02E+03	2.10E+05	1.03E+09
F22	Std	3.02E+02	1.68E+02	2.78E+02	4.61E+02	8.69E+02	1.88E+02	2.53E+02	6.94E+02	3.39E+09	1.73E+01
F23	Avg	2.89E+03	2.72E+03	2.72E+03	2.93E+03	2.73E+03	2.76E+03	2.68E+03	2.53E+03	2.52E+03	2.50E+03
F23	Std	1.22E+02	2.72E+01	4.59E+01	1.27E+02	9.32E+01	5.33E+01	8.46E+01	1.03E+01	7.06E+00	0.00E+00
F24	Avg	2.74E+03	2.61E+03	2.66E+03	2.70E+03	2.61E+03	2.60E+03	2.60E+03	2.60E+03	2.60E+03	2.60E+03
F24	Std	2.74E+01	9.51E+00	9.64E+00	1.65E+01	1.76E+01	4.85E-02	1.66E-04	1.74E-04	5.30E-05	0.00E+00
F25	Avg	2.77E+03	2.74E+03	2.74E+03	2.74E+03	2.73E+03	2.71E+03	2.70E+03	2.70E+03	2.70E+03	2.70E+03
F25	Std	1.99E+01	1.02E+01	1.65E+01	9.07E+00	7.41E+00	1.34E+01	3.04E-13	0.00E+00	1.54E+00	0.00E+00
F26	Avg	2.71E+03	2.70E+03	2.73E+03	2.71E+03	2.78E+03	2.79E+03	2.77E+03	2.70E+03	2.70E+03	2.71E+03
F26	Std	1.62E+00	4.89E-01	4.62E+01	5.77E+00	5.62E+01	5.34E+01	4.70E+01	1.68E-01	8.31E-02	5.45E-01
F27	Avg	3.86E+03	3.85E+03	3.77E+03	3.91E+03	3.80E+03	3.98E+03	3.59E+03	3.77E+03	3.36E+03	2.90E+03
F27	Std	1.67E+02	2.59E+02	3.92E+02	2.19E+02	3.11E+02	7.74E+01	3.09E+02	9.10E+01	1.63E+02	0.00E+00
F28	Avg	6.42E+03	5.48E+03	6.64E+03	8.93E+03	7.53E+03	5.71E+03	5.68E+03	3.93E+03	3.95E+03	3.00E+03
F28	Std	1.12E+03	3.51E+02	1.06E+03	6.10E+02	9.82E+02	2.49E+02	5.31E+02	1.41E+02	8.75E+01	0.00E+00
F29	Avg	4.13E+07	3.03E+07	8.04E+07	2.98E+08	6.30E+07	5.33E+07	1.87E+07	7.99E+03	2.90E+07	3.10E+03
F29	Std	3.96E+07	1.28E+07	7.56E+07	9.05E+07	4.69E+07	3.55E+07	2.28E+07	2.63E+04	1.77E+07	0.00E+00
F30	Avg	6.98E+05	5.27E+05	4.72E+05	2.05E+06	5.07E+05	7.99E+05	1.69E+05	7.90E+06	1.95E+08	3.20E+03
F30	Std	4.27E+05	1.68E+05	3.78E+05	1.09E+06	5.27E+05	1.88E+05	1.49E+05	5.92E+06	8.84E+07	0.00E+00

Для функцій F1–F15 було порівняно роботу алгоритмів CHC, TBO та EJS з DE, MTDE, PSO, AVOA, CHIO, HGS, RSO, SCA, SMA, WOA, WSO та YDSE [1].

Нижче в експерименті для кожної з тридцяти функцій було зафіксовано середнє значення знайденого мінімуму та стандартне відхилення для п'ятдесяти запусків, а також побудовано графіки медіанної збіжності для відслідковування швидкості збіжності до мінімуму.

В цих випадках середнє значення та стандартне відхилення менше, ніж в CHC. Проте варто зазначити, що ця різниця незначна в порівнянні з тим, що на інших функціях CSO демонструє на порядок нижчу

продуктивність. Лише на функціях F17 та F21 CSO показав кращий середній мінумум. Проте результати алгоритмів в обох випадках значно гірші за оптимальні. У всіх інших випадках алгоритм CHC показав або краще значення, або таке саме.

Таблиця Б.6 – Середні значення та стандартні відхилення (DE–SCA)

Fn	Stat	DE	MTDE	PSO	AVOA	CHIO	HGS	RSO	SCA
F1	Avg	2.06E+05	1.00E+02	6.56E+05	2.57E+05	3.83E+05	7.17E+04	1.03E+07	2.77E+06
F1	Std	1.46E+05	1.12E-14	1.90E+06	2.00E+05	1.94E+05	7.86E+04	5.20E+06	1.20E+06
F2	Avg	9.55E+03	2.00E+02	1.18E+08	9.41E+03	1.16E+03	2.12E+04	2.56E+09	2.45E+09
F2	Std	8.43E+03	1.06E-14	2.71E+08	9.28E+03	1.32E+03	3.30E+04	1.83E+09	1.02E+09
F3	Avg	1.48E+03	3.00E+02	9.60E+02	1.43E+03	2.37E+03	1.69E+03	1.20E+04	4.07E+03
F3	Std	9.81E+02	2.36E-14	2.52E+03	1.11E+03	1.45E+03	1.09E+03	4.54E+03	1.58E+03
F4	Avg	4.03E+02	4.00E+02	4.18E+02	4.03E+02	4.01E+02	4.03E+02	6.35E+02	4.27E+02
F4	Std	1.20E+01	3.95E-14	3.89E+01	1.24E+01	1.32E+00	1.32E+01	1.13E+02	1.06E+01
F5	Avg	5.20E+02	5.20E+02	5.20E+02	5.20E+02	5.20E+02	5.20E+02	5.20E+02	5.20E+02
F5	Std	4.03E-02	2.22E-02	8.70E+00	6.22E-02	1.59E-03	1.59E-02	9.16E+00	2.76E-02
F6	Avg	6.05E+02	6.05E+02	6.01E+02	6.05E+02	6.02E+02	6.04E+02	6.09E+02	6.07E+02
F6	Std	1.49E+00	1.05E+00	1.41E+00	1.88E+00	7.25E-01	1.57E+00	1.32E+00	9.76E-01
F7	Avg	7.00E+02	7.00E+02	7.00E+02	7.00E+02	7.00E+02	7.00E+02	7.40E+02	7.03E+02
F7	Std	2.37E-01	9.79E-02	4.99E-02	2.74E-01	1.78E-02	7.09E-02	1.27E+01	1.11E+00
F8	Avg	8.05E+02	8.00E+02	8.02E+02	8.05E+02	8.00E+02	8.00E+02	8.00E+02	8.42E+02
F8	Std	2.77E+00	1.82E-01	2.01E+00	2.86E+00	5.98E-04	2.11E-14	7.76E+00	4.62E+00
F9	Avg	9.25E+02	9.08E+02	9.07E+02	9.23E+02	9.11E+02	9.15E+02	9.45E+02	9.30E+02
F9	Std	1.00E+01	3.48E+00	2.81E+00	8.95E+00	3.03E+00	8.29E+00	6.52E+00	6.53E+00
F10	Avg	1.19E+03	1.01E+03	1.28E+03	1.26E+03	1.00E+03	1.02E+03	2.15E+03	1.80E+03
F10	Std	1.62E+02	6.90E+01	1.67E+02	2.11E+02	1.14E-01	4.02E+01	2.37E+02	1.97E+02
F11	Avg	1.90E+03	1.40E+03	1.49E+03	1.95E+03	1.40E+03	1.60E+03	2.33E+03	2.37E+03
F11	Std	2.89E+02	1.67E+02	1.83E+02	2.52E+02	1.01E+02	2.14E+02	3.07E+02	1.57E+02
F12	Avg	1.20E+03	1.20E+03	1.20E+03	1.20E+03	1.20E+03	1.20E+03	1.20E+03	1.20E+03
F12	Std	2.03E-01	1.64E-01	1.12E-01	2.26E-01	5.07E-01	1.03E-01	2.83E-01	1.79E-01
F13	Avg	1.30E+03	1.30E+03	1.30E+03	1.30E+03	1.30E+03	1.30E+03	1.30E+03	1.30E+03
F13	Std	1.59E-01	5.14E-02	6.63E-02	1.47E-01	5.74E-02	1.60E-01	3.03E-01	1.04E-01
F14	Avg	1.40E+03	1.40E+03	1.40E+03	1.40E+03	1.40E+03	1.40E+03	1.41E+03	1.40E+03
F14	Std	1.34E-01	1.96E-02	3.19E-01	1.10E-01	4.40E-02	2.23E-01	7.46E-02	9.23E-02
F15	Avg	1.50E+03	1.50E+03	1.50E+03	1.50E+03	1.50E+03	1.50E+03	1.51E+03	1.50E+03
F15	Std	2.10E+00	3.07E-01	7.20E+00	1.76E+00	4.21E-01	9.86E-01	3.94E+02	1.57E+00