 https://doi.org/10.5281/zenodo.10451666

ПРІОРИТЕЗАЦІЯ ВИМОГ ПРИ РОЗРОБЦІ ПРОЕКТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОБ'ЄКТІВ КРИТИЧНОЇ ІНФРАСТРУКТУРИ

Я.Ю. Дорогий[1], О.О. Дорога-Іванюк[2]

[1] Department of Applied Mathematics and Informatics, Donetsk National Technical University, Lutsk, Ukraine
[2] Department of Computer Science and Software Engineering, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine

E-mail: yaroslav.dorohyi@donntu.edu.ua

Отримано 30.10.2023
Прийнято до публікації 09.11.2023
Опубліковано 01.04.2024

АНОТАЦІЯ

Мета роботи полягає в розробці алгоритму пріоритезації вимог при розроблюванні програмного забезпечення для проектів об'єктів критичної інфраструктури.

Розробка вимог є базовою фазою будь-якого проекту програмного забезпечення, оскільки ця фаза пов'язана з ідентифікацією вимог, обробкою та маніпулюванням. Основним джерелом цих вимог є зацікавлені сторони проекту з урахуванням обмежень та лімітів проекту. Кількість вимог різна для кожного проекту програмного забезпечення для об'єкту критичної інфраструктури, тому термін пріоритезації вимог стосується визначення пріоритетності порядку виконання вимог до програмного забезпечення відповідно до міркувань і рішень зацікавлених сторін. Для вирішення оптимізаційних задач використовуються різні запропоновані алгоритми оптимізації. В даній роботі наведено основні етапи базових алгоритмів оптимізації, деякі їх модифікації з метою покращення їх ефективності у розв'язанні такого типу задач. Також, у цій статті пропонується гібридний підхід, заснований на алгоритмах оптимізації WOA та GWO, утворений шляхом поєднання переваг кожного алгоритму з метою визначення пріоритетності вимог до програмного забезпечення ОКІ. Крім того, наведено набір даних з проекту SKUDA, який використано в даному дослідженні та відповідає вимогам реального програмного проекту для оцінки запропонованого методу.

Наукова новизна полягає в модифікації, застосуванні та поєднанні результатів відомих алгоритмів GWO та WOA для розв'язання задачі пріоритезації вимог для проектів програмного забезпечення об'єктів критичної інфраструктури.

Запропонований алгоритм дає точність 92% для запропонованого набору вимог.

***Ключові слова**: пріоритезація вимог, WOA, GWO, об'єкт критичної інфраструктури, ОКІ, гібридний підхід*

 https://doi.org/10.5281/zenodo.10451666

ВСТУП

Інженерія вимог (ІВ) є однією з найважливіших галузей у сфері інженерії програмного забезпечення. Крім того, вона вважається найважливішою фазою у життєвому циклі розробки програмного забезпечення. Ця фаза включає ідентифікацію та виборку вимог, аналіз та перевірку вимог і їх документацію. Практично в усіх проектах програмного забезпечення існують обмеження у процесі розробки, такі як бюджет та час до виходу продукції на ринок, що призводить до поетапної поставки проектів програмного забезпечення. Таким чином, у великих проектах існує більше одного зацікавленого в результатах учасника, що ускладнює процес вибору того, який випуск слід розробляти першим. Ця складність змушує інженерів програмного забезпечення ефективно визначати пріоритети вимог, щоб приймати правильні рішення щодо поставки та розробки проекту [1-2].

Отже, визначення пріоритетів вимог (RP) є найважливішою частиною ІВ, яка входить до стадії аналізу вимог. Визначення пріоритетів вимог вважається однією з найважливіших дій у процесі створення програмного проекту та надання якісної системи, яку потребує замовник. У випадках, коли у проекті існують жорсткий графік виконання, недостатні ресурси та високі очікування споживачів, важливо опублікувати найважливіші характеристики якнайшвидше. Саме це і вимагає визначення пріоритетів вимог.

Процес визначення пріоритетів вимог підвищує участь учасників проекту, включаючи їх у вирішення питань, які вимоги повинні бути включені в програмне забезпечення разом з їхньою важливістю та впливом на процес розробки. Ця участь допомагає учасникам розуміти обмеження та ліміти ресурсів проекту та вирішувати конфлікти між різними точками зору, які дійсно впливають на розробку програмного забезпечення. Ці конфлікти виникають з різних цілей та ролей учасників проекту [3]. Таким чином, у проектах з великою кількістю вимог визначення пріоритетів стає основою успіху чи невдачі програмного продукту відповідно до обмежень та лімітів проекту. Ця участь та взаємодія спрямовані на визначення пріоритетів вимог з урахуванням їх важливості ефективним способом та корисним порядком.

З цієї причини можна використовувати техніку кластеризації для класифікації вимог за їх важливістю. Кластеризація даних - одна з найважливіших функцій у сфері видобутку даних, яка востаннє привертає увагу багатьох авторів, дослідників та експертів. Кластеризація - це неконтрольоване упорядкування типів у групи [4]. Основна мета кластеризації даних полягає в ідеї групування об'єктів у групи на основі схожості та відмінності між ними [5-6].

Загалом, кластеризацію можна поділити на дві основні категорії: жорстку кластеризацію і м'яку кластеризацію. У жорсткій кластеризації об'єкти даних належать лише до одного кластеру, в той час як у м'якій кластеризації всі окремі об'єкти даних належать до окремих класів у певному діапазоні [5, 7]. Основна мета кластеризації даних полягає в тому, що вона оптимально сортує всі N даних у K кластерів так, щоб неявні типи даних стали видимими [8]. Таким чином, кожен кластер містить об'єкти даних, які схожі за характеристиками, а також кластери відрізняються один від одного. Одним із найважливіших методів, які використовуються у процесі дослідження даних, нейрокомп'ютерних технологій, сегментації зображень та інших інженерних задач, є кластерний аналіз [9]. Наразі багато методів кластеризації були запропоновані дослідниками та поділені на алгоритми кластеризації на основі моделі, ієрархічні алгоритми кластеризації, алгоритми кластеризації на основі розбиття, алгоритми кластеризації на основі сіток та алгоритми кластеризації на основі щільності [8, 10]. Дані, які розділяються на K кластерів, використовують відстань Евкліда як міру у алгоритмах кластеризації на основі розбиття, також в алгоритмах кластеризації на основі ієрархії створюється дерево груп.

Останнім часом багато дослідників запропонували безліч метаевристик і евристичних алгоритмів для вирішення проблем, які виникають у зв'язку з складними наборами даних. Проте більшість технік, які запропоновані для вирішення проблем оптимізації, базуються на метаевристичних алгоритмах [9, 11]. Основна мета метаевристи-чних алгоритмів – знайти оптимальні рішення для формування кластерів даних та зменшення проблеми локальних мінімумів [12-13]. Останні метаевристичні алгоритми оптимізації – це алгоритм GWO (Grey Wolf Optimization) та алгоритм WOA (Whale Optimization Algorithm).

Алгоритм GWO був запропонований Мірджалілі та іншими у 2014 році [14]. Цей алгоритм моделює поведінку сірих вовків у природі під час полювання. Сірі вовки є одними з найвідоміших хижаків в природі. Зазвичай вони живуть у групах розміром від 5 до 12 особин. У цих вовків існують стійкі правила в соціальній ієрархії. Згідно з [14], сірі вовки включають альфа, бета, омега та дельта вовків. Альфа-вовк є лідером стаї і відповідає за прийняття рішень щодо полювання та інших діяльностей. Бета-вовк допомагає вищому рівню приймати рішення. Омега-вовки відповідальні за передачу інформації на найвищі рівні. Усі інші вовки у стаї називаються дельтами.

ISSN 2786-9024

 https://doi.org/10.5281/zenodo.10451666

Алгоритм WOA був запропонований Мірджалілі та Льюїсом у 2016 році [10]. Основна його мета – знайти глобальне оптимальне рішення для будь-якої задачі оптимізації. Головна відмінність цього алгоритму від інших полягає у принципі, який розвиває рішення-кандидата на кожній ітерації оптимізації. Крім того, процес полювання китів за допомогою бульбашкової мережі відображає процес полювання горбатих китів на джерело їжі.

АНАЛІЗ ЛІТЕРАТУРНИХ ДАНИХ ТА ПОСТАНОВКА ПРОБЛЕМИ

Як вже зазначалося, фаза пріоритезації вимог - це операція, яка визначає пріоритет однієї вимоги порівняно з іншими. Крім того, це найбільш мотивуючий напрямок в області інженерії вимог. На даний момент було запропоновано багато технік для надання переваги вимозі програмного забезпечення порівняно з іншими вимогами у тому ж самому проекті. Техніки пріоритезації вимог класифікуються як порядкова шкала, ординальна шкала і відносна шкала з точки зору потужної пріоритетизації. Найпотужнішою є відносна шкала, яка показує, наскільки одна вимога важливіша за інші вимоги. Найменш потужною є ординальна шкала, яка відповідає за надання вимог в порядку пріоритету та визначає, яка вимога важливіша за інші, але без вказівки на те, наскільки саме важлива.

Аналітичний ієрархічний процес (AHP) є найпопулярнішою і традиційною технікою, яка відноситься до класу відносної шкали. Тому його згадують у великій кількості досліджень [15-17]. Крім того, його вважають системною технікою прийняття рішень, яка використовується для пріоритетизації вимог для конкретного проекту [18-19]. Він порівнює всі можливі пари вимог для їх упорядкування і визначення, яка має вищий пріоритет. AHP не підходить для проектів, у яких є велика кількість вимог [20-21]. Тому багато дослідників намагалися зменшити кількість порівнянь, наприклад, [22-23], і були запропоновані різні техніки для зменшення кількості порівнянь приблизно на 75%, такі як [24].

Кумулятивне голосування (Cumulative Voting) називають "тестом на 100 доларів", що є прямолінійною технікою, де стейкхолдерам надається 100 уявних одиниць, які слід розподілити серед наданих вимог до програмного забезпечення [25]. Результати пріоритетизації подаються у вигляді відносної шкали.

Механізм MoScoW вважається одним з видів числового призначення, який зображений у [26]. Крім того, ця техніка має чотири класи пріоритету: "повинно бути", "може бути", "повинно бути", "не повинно бути". "Повинно бути" означає, що вимоги, які потрапили в цей клас, повинні бути реалізовані спочатку, і так далі. "Може бути" означає, що вимога, яка потрапила в цей клас, існує, і це буде чудово для програмного продукту. "Повинно бути" означає, що вимоги, які знаходяться в цьому класі, повинні бути реалізовані і будуть чудовими для програмного продукту, і, нарешті, "не повинно бути" означає, що вимоги, які потрапили в цей клас, не можуть бути реалізовані на даному етапі, оскільки інші вимоги мають менший пріоритет.

Механізм "сортування бульбашкою" використовується для ранжування елементів, таких як вимоги. У цій техніці вибирають дві вимоги для порівняння між собою. У разі, якщо вимога не впорядкована, їх обмінюють місцями і потім порівнюють з іншою вимогою, і так продовжують, поки не отримають впорядкований список вимог у спадаючому порядку (від вищого до нижчого), як використовується в цих дослідженнях [27].

Техніка бінарного пошукового дерева - це ще один вид методу ранжування, який був запропонований у [27]. Крім того, ця техніка була вперше представлена [15] для пріоритетизації вимог. Кожен вузол у цій техніці вказує на вимогу програмного забезпечення, де всі вимоги, що знаходяться ліворуч від дерева, мають менший пріоритет порівняно з іншими вузлами, тоді як інші вимоги, що знаходяться праворуч від дерева, мають вищий пріоритет. Однак спочатку обирається одна вимога, яка стає кореневим вузлом, та порівнюється з несортованою вимогою. У разі, якщо ця вимога має менший пріоритет, ніж корінь, вона шукає ліворуч від дерева. В іншому випадку вона шукає праворуч від дерева. Операція повторюється, поки не отримаємо відсортоване дерево.

У техніці "десять найважливіших вимог" стейкхолдери вибирають десять найважливіших вимог за їхньою важливістю для них, не вказуючи внутрішнього рангу серед цих вимог. Це робить цю техніку відповідною для багатьох стейкхолдерів однакової важливості [28].

В роботі [11] автор використовував оптимізаційний алгоритм GWO, який є одним з найбільш актуальних метаевристичних алгоритмів, для пріоритетизації вимог програмного проекту. Крім того, цей алгоритм був оцінений і порівняний із механізмом AHP, де запропонована робота показала кращі результати, ніж традиційний метод AHP приблизно на 30%. З іншого боку, в [29] автор застосовував алгоритм WOA, який знедавна використовується для вирішення проблем оптимізації, оскільки він імітує поведінку кита, застосовуючи метод полювання за допомогою бульбашкової сітки. Цей метод також був оцінений за допомогою AHP, де результати показали, що запропонована робота перевершує механізм AHP приблизно на 40%.

Формулювання проблеми: в умовах постійно зростаючої потреби в міждисциплінарних компетенціях існує ряд прикладних задач з прогнозування, зокрема, економічного, серед яких визначення бажаного часу перебування покупців в торговельному залі для отримання найбільшої середньої суми чеку, а також визначення залежності сезонних коливань середньої суми чеку із урахуванням щорічного зростання цін, розв’язання яких можливе засобами машинного навчання.

Мета: розробка програмного коду на основі бібліотек Python для прогнозування зазначених вище явищ. Вибір мови продиктований її поширеністю, доступністю до вільного і безкоштовного використання навіть в комерційних цілях [7], відносною легкістю опанування, можливістю виконувати досить складні вправи з програмування без необхідності встановлення платного або складного середовища.

Задачі:

1. Аналіз тематичних наукових джерел і методичних матеріалів.
2. Експериментальна перевірка запропонованих в них прикладів, їхня адаптація під зазначені вище завдання з прогнозування.
3. Розробка і тестування програмного коду моделей прогнозування.
4. Аналіз результатів, отриманих від розроблених моделей.

МАТЕРІАЛИ ТА МЕТОДИ ДОСЛІДЖЕНЬ

В роботі використано методи структурного і порівняльного аналізу, з інформаційним і аналітичним підходом розглянуто наукову та методичну літературу, а також онлайн ресурси, застосовано експериментальні методи для перевірки програмного коду.

РЕЗУЛЬТАТИ ДОСЛІДЖЕНЬ

Далі розглянуті основні методи, які використані для розв’язання поставленої задачі.

А. ПРІОРИТЕТИЗАЦІЯ ВИМОГ

Значення пріоритетизації вимог розглядається з різних точок зору. Самервіль визначив пріоритетизацію вимог як одне з найважливіших завдань для прийняття рішень [15]. З іншого боку, Файрсміт визнає її основним процесом у сфері програмної інженерії, оскільки вона визначає ідеальний порядок реалізації вимог для планування версій програми та надання бажаної функціональності якнайшвидше. Це також процес визначення пріоритету вимог для зацікавлених сторін на основі їх важливості [16]. Отже, ми можемо зробити висновок, що пріоритетизація вимог передбачає визначення пріоритету за важливістю або за можливістю впровадження.

Реалізація найважливіших операцій, які дозволяють отримати приростовий зворотний зв'язок від клієнта, встановлює графік, виправляє помилки та усуває будь-які непорозуміння між клієнтом і корпорацією на ранніх етапах, призводячи до задоволеності клієнта. Крім того, це корисно для вилучення непотрібних вимог, які можуть бути неефективно витратні, і вибору найбільш підходящих вимог для кожної версії; це сприяє майбутньому плануванню, зменшує ризик скасування, оцінює переваги, розставляє пріоритети в інвестиціях та визначає фінансові наслідки щодо впровадження кожної вимоги [17].

Пріоритетизація вимог є найважливішою і критичною частиною аналізу вимог через обмеження ресурсів проекту. Іншими словами, важко реалізувати всі вимоги одночасно через обмеження ресурсів, таких як графік, персонал і бюджет. Більше того, для удосконалення деяких проектів можуть знадобитися кілька місяців або навіть років, тому важливо визначити вимогу, яка повинна бути реалізована спочатку. Крім того, бюджет відіграє важливу роль, особливо при взаємодії з процесом пріоритетизації вимог, оскільки бюджет вважається менш важливою діяльністю в контексті інженерії вимог порівняно з іншими аспектами програмної інженерії. Наведене вище вказує на те, що вимоги мають різні рівні важливості, і важко визначити, яка з них є найважливішою.

Як було зазначено вище, зацікавлені сторони проекту є основою процесу пріоритетизації з урахуванням бізнес- і регуляторних факторів, оскільки вони мають різні точки зору, і кожен з них повинен визначити найвищий пріоритет вимог, щоб змусити зацікавлені сторони чітко зібрати всі відносно важливі вимоги, що сприяє підвищенню комунікації між ними, надає розумну основу для взаємодії щодо вимог та дозволяє планувати інженерні активності в розумний спосіб.

Б. АЛГОРИТМ GWO

Алгоритм GWO – це один із методів оптимізації, натхненних природою, який був запропонований Мірджалілі та ін. у 2014 році [14]. Він імітує поведінку сірих вовків на полюванні. Основною метою алгоритму GWO є визначення оптимального рішення для задачі за допомогою популяції пошукових агентів.

Ці вовки зазвичай живуть у групах, кількість учасників яких коливається від п'яти до дванадцяти. Основна відмінність між алгоритмом GWO та іншими алгоритмами

ISSN 2786-9024

 https://doi.org/10.5281/zenodo.10451666

оптимізації полягає в соціальній ієрархії, яка розвиває рішення-кандидата на кожній ітерації оптимізації. На практиці GWO імітує поведінку вовків у пошуку та нападі на жертв на полюванні. Соціальна ієрархія у групі вовків показана на Рис. 1 [14].

Альфа відображає лідера, який є найкращим кандидатом на рішення. Крім того, альфа - домінуючі вовки, за якими йдуть інші. Бета представляє другого кандидата на рішення, який допомагає альфа в прийнятті рішень і є містком між лідером та рештою загону, що грає на найнижчих рівнях. Дельта відображає третього кандидата на рішення, який відповідає за надсилання інформації на два вищих рівні (альфі і беті). Омега представляє решту рішень і відповідає за надсилання інформації на три вищих рівні.

Фактично механізм полювання включає три етапи: відстеження, оточення і напад на здобич. Таким чином, GWO відображає математичний метод полювання сірих вовків, який використовується для вирішення складних задач оптимізації. В даному контексті, оптимальне рішення проблеми розглядається як жертва.

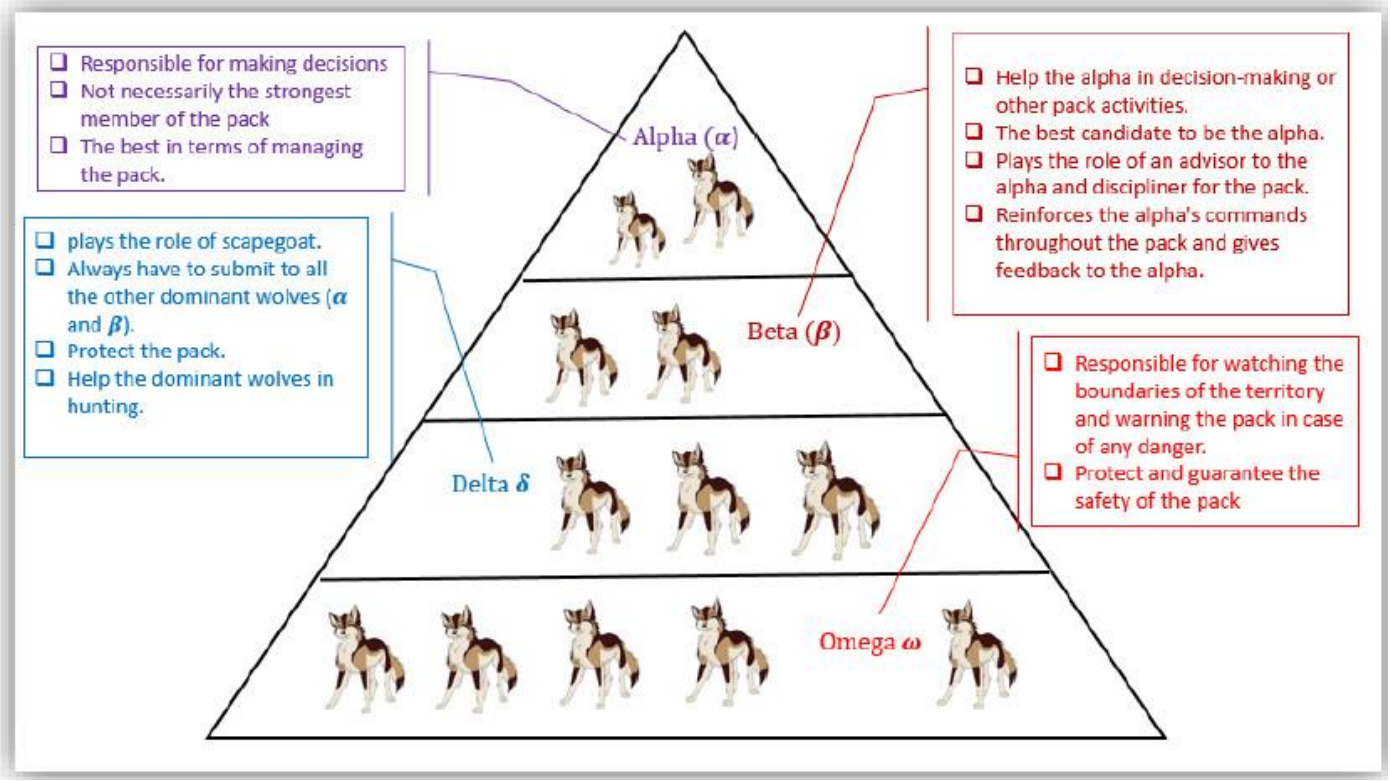

Рис.1. Ієрархія сірих вовків [14]

Рух трьох верхніх рівнів імітує оточення жертви сірими вовками, що відображено у наступній формулі (1):

$$\vec{D} = \left| \vec{C} \cdot \vec{X}_p(t) - \vec{X}(t) \right|, \vec{C} = 2\vec{r}_2 \quad (1)$$

де

t – поточна ітерація,

X_p – вказує на вектор позиції здобичі,

X – позиція сірого вовка,

C – вектор коефіцієнтів,

r_2 – обирається випадково з інтервалу [0, 1].

Таким чином, результат вектора D використовується для зміщення конкретного елементу в бік або від області, в якій знаходиться найкраще рішення, що представляє здобич, за допомогою наступного рівняння (2):

$$\vec{X}(t+1) = \left| \vec{X}_p(t) - \vec{A} \cdot \vec{D} \right|, \vec{A} = 2\vec{a}\vec{r}_1 - \vec{a} \quad , (2)$$

де

r_1 – обирається випадково з інтервалу [0, 1],

a – мінімізується від 2 до 0 протягом попередньо заданої кількості ітерацій.

Щоб побачити ефекти рівнянь (1) та (2), на рис. 2 (а) проілюстровано двовимірний вектор положення та деякі з можливих наступних положень вовка. Як видно на цьому рисунку, сірий вовк у положенні (X, Y) може оновити своє положення відповідно до положення здобичі $(X*, Y*)$. Додавши різні значення навколо найкращого агента до поточної позиції, можна налаштувати значення A та C вектори. Наприклад, $(X* - X, Y*)$ можна отримати, встановивши A = (0, 1) та C = (1, 1).

 https://doi.org/10.5281/zenodo.10451666

Можливі оновлені позиції сірого вовка в тривимірному просторі зображені на Рис. 2 (b). Зверніть увагу, що випадкові вектори і дозволяють вовкам досягти будь-якого положення між точками, зображеними на Рис. 2. Отже, сірий вовк може оновити своє положення всередині простору, та навколо здобичі у будь-якому випадковому місці, використовуючи рівняння (1) та (2).

Цю ж концепцію можна поширити на простір пошуку з n розмірами, і сірі вовки рухатимуться в гіперкубах (або гіперсферах) навколо найкращого рішення, отриманого за зданими умовами [14].

Б.1. ФАЗА ПОЛЮВАННЯ

Сірі вовки мають здатність розпізнавати місцезнаходження здобичі та оточувати їх. Полювання, як правило, керується альфою. Бета та дельта також можуть час від часу брати участь у полюванні. Однак в абстрактному просторі пошуку ми не маємо уявлення про розташування оптимуму (здобичі). Для математичного моделювання мисливської поведінки сірих вовків ми вважаємо, що альфа (найкращий варіант рішення) бета та дельта мають кращі знання про потенційне розташування здобичі. Тому ми зберігаємо перші три найкращі рішення, отримані на сьогодні, та зобов'язуємо інші пошукові агенти (включаючи омеги) оновити свої позиції відповідно до позиції найкращого пошукового агента.

Таким чином, три вищі рівні α, β і δ обчислюються за допомогою наступних математичних виразів (3-5):

$$\vec{D}_\alpha = \left|\vec{C}_1 \cdot \vec{X}_\alpha - X\right|, \vec{X}_1 = \vec{X}_\alpha - \vec{A}_\alpha \cdot (\vec{D}_\alpha), \quad (3)$$

$$\vec{D}_\beta = \left|\vec{C}_2 \cdot \vec{X}_\beta - X\right|, \vec{X}_2 = \vec{X}_\beta - \vec{A}_\beta \cdot (\vec{D}_\beta), \quad (4)$$

$$\vec{D}_\delta = \left|\vec{C}_3 \cdot \vec{X}_\delta - X\right|, \vec{X}_3 = \vec{X}_\delta - \vec{A}_\delta \cdot (\vec{D}_\delta). \quad (5)$$

Для математичного імітування процесу полювання сірого вовка припускається, що α, β і δ мають достатньо інформації про можливе розташування жертви. Більше того, перші три найкращі рішення, які отримані, зберігаються і змушують інших агентів оновлювати свої позиції відповідно до кращих агентів α, β і δ. Ця поведінка математично представлена наступним виразом (6), і псевдокод GWO показаний в Алгоритмі 1 [14].

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3}, \quad (6)$$

Алгоритм 1

Ініціалізація популяції **Xi (i = 1, 2, 3… n)**
Ініціалізація а**, A, C**
Розрахунок придатності для кожного агента
Xα = кращий пошуковий агент
Xβ = **2-**й кращий пошуковий агент
Xδ = **3-**й кращий пошуковий агент
WHILE (t < максимальна к**-**сть ітерацій**)**
  FOR EACH пошукового агента
    оновлення позицій для поточного пошукового агента за рівнянням **(6)**
  ENDFOR
  Оновлення а**, A, C**
  Розрахунок придатності для кожного агента
  Оновлення **X**α**, X**β**, X**δ
  t = t + 1
ENDWHILE
RETURN Xα

На Рис. 3 показано, як пошуковий агент оновлює свою позицію відповідно до альфи, бети та дельти у 2D пошуковому просторі. Можна помітити, що кінцева позиція знаходитиметься у випадковому місці в колі, яке визначається положеннями альфи, бети та дельти в просторі пошуку. Іншими словами, альфа, бета та дельта оцінюють положення здобичі, а інші вовки випадково оновлюють свої позиції навколо здобичі.

Б.2 ФАЗА НАПАДУ

Як згадувалося вище, сірі вовки закінчують полювання, атакуючи здобич, коли вона перестає рухатися. Для математичного моделювання наближення жертви ми зменшуємо значення a. Зверніть увагу, що діапазон коливань A також зменшується на a. Іншими словами A, це випадкове значення в інтервалі$[-a, a]$, де а зменшується з 2 до 0 протягом ітерацій. Коли випадкові значення значень знаходяться в [−1, 1], наступна позиція пошукового агента може знаходитись у будь-якій позиції між його поточною позицією та позицією здобичі.

У випадку, якщо |A| < 1, це відповідає стратегії експлуатації і симулює поведінку нападу на здобич. З іншого боку, якщо |A| > 1, це відповідає стратегії дослідження і імітує віддалення вовків від жертви.

ISSN 2786-9024

 https://doi.org/10.5281/zenodo.10451666

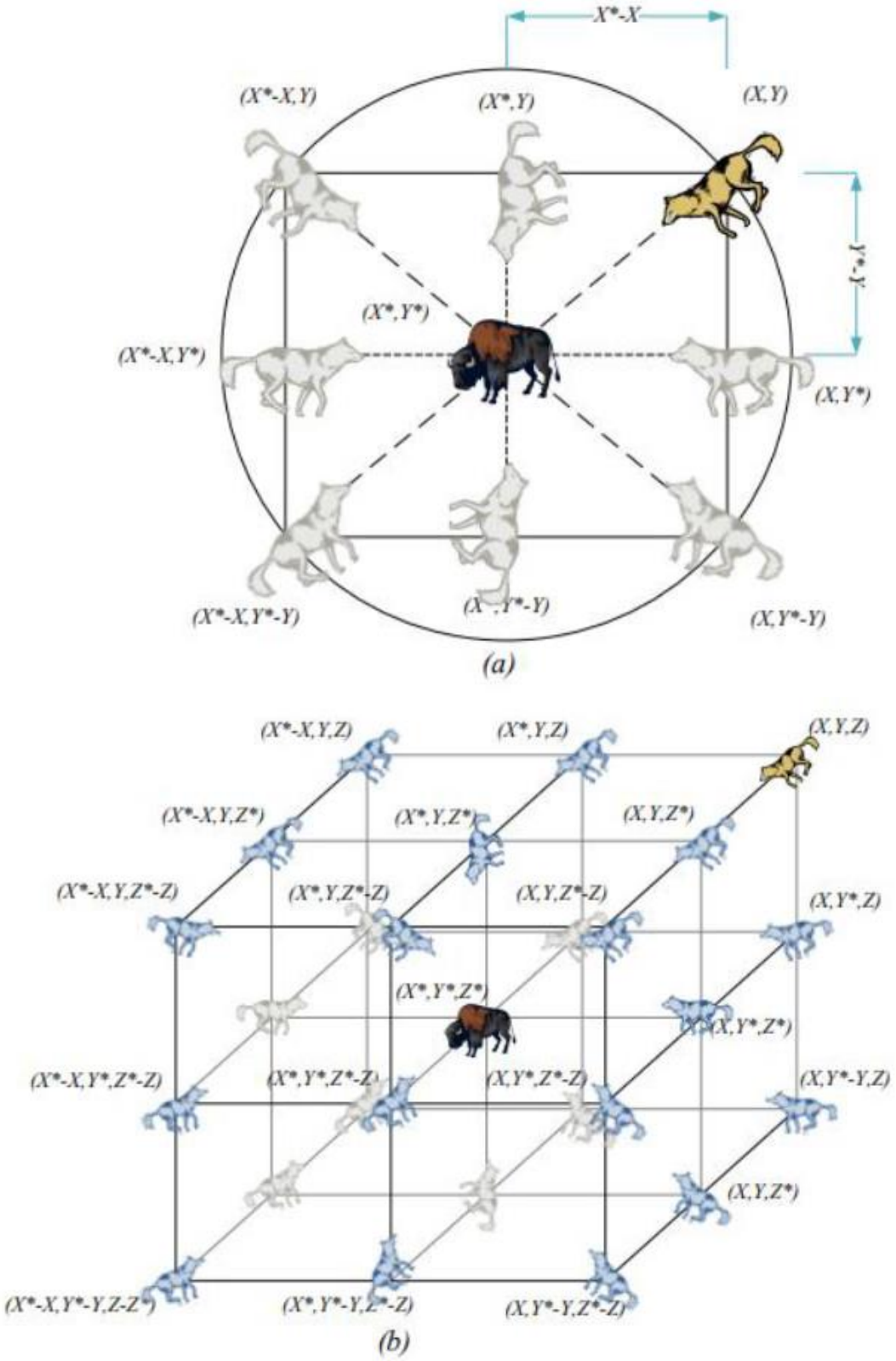

Рис. 2. a) 2D вектор розташування вовка, та одну з можливих наступних локацій [14]
b) 3D вектор розташування вовка, та одну з можливих наступних локацій [14]

 https://doi.org/10.5281/zenodo.10451666

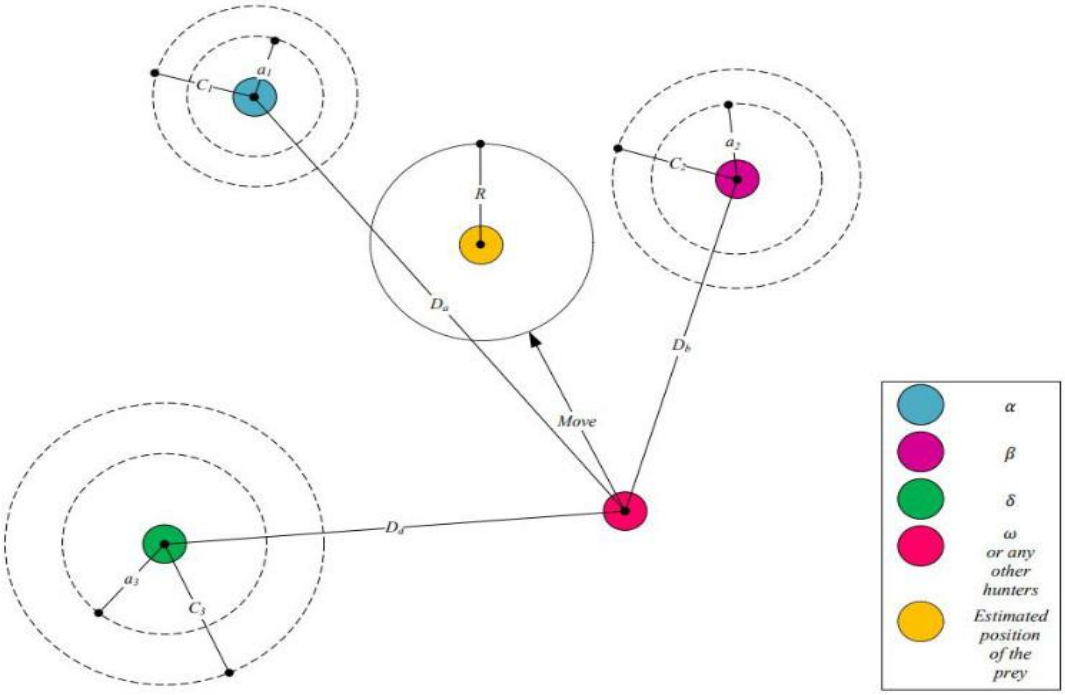

Рис.3. Оновлення позицій в GWO [14]

B. АЛГОРИТМ WOA

Алгоритм WOA є алгоритмом оптимізації, запропонованим Мірджалілі та Льюїсом у 2016 році [10]. Метою цього алгоритму є визначення глобального оптимуму для проблеми за допомогою популяції агентів пошуку (китів). Процес пошуку починається зі створення колекції рішень-кандидатів, які випадковим чином вибираються для задачі. Потім ця колекція поліпшується протягом багатьох ітерацій до досягнення задоволення кінцевої умови. Насправді, кити імітують особливий метод полювання, який називається методом "bubble-net feeding" (метод міхурової сітки), який показаний на Рис. 4 [10].

З Рис. 4 очевидно, що кит зганяє здобич, рухаючись за спіральним маршрутом навколо жертв, утворюючи бульбашки попереду. Таким чином, цей процес пошуку є основним натхненням для WOA. Крім того, ще одним імітованим методом у WOA є метод скорочення оточення.

Рис.4. Поведінка горбатих китів та метод міхурової сітки [10]

Кити-горбачі оточують жертв навколо себе, щоб розпочати полювання на них за допомогою методу ловлі "bubble-net" (метод бульбашкової сітки).

B.1. ФАЗА ОТОЧЕННЯ

Оскільки положення оптимальної схеми в просторі пошуку апріорі невідомо, алгоритм WOA передбачає, що поточне найкраще можливе рішення є цільовою жертвою або близько до оптимального. Після визначення кращого пошукового агента інші пошукові агенти спробують поновити свої позиції щодо кращого пошукового агента. Це поведінка представлено наступними рівняннями [10]:

$$\vec{D} = \left|\vec{C} \cdot \vec{X}^*(t) - \vec{X}(t)\right|, \vec{C} = 2\vec{r}_2 \quad (7)$$

$$\vec{X}(t+1) = \left|\vec{X}^*(t) - \vec{A} \cdot \vec{D}\right|, \vec{A} = 2\vec{a}\,\vec{r}_1 - \vec{a} \quad (8)$$

де

t – означає поточну ітерацію,

A – коефіцієнтний вектор,

C – коефіцієнтний вектор,

$X*(t)$ – вектор розташування найкращого рішення, отриманого на даний момент,

$X(t)$ – вектор розташування кита,

a – лінійно зменшується від 2 до 0 в процесі ітерацій,

r_1 – випадковий вектор з проміжку [0, 1],

r_2 – випадковий вектор з проміжку [0, 1].

Рис. 5 ілюструє обґрунтування рівняння (7) для двовимірної задачі. Положення (X, Y) пошукового агента

ISSN 2786-9024

 https://doi.org/10.5281/zenodo.10451666

може бути оновлено відповідно до положення поточної кращого запису ($X*$, $Y*$). Різні місця навколо кращого агента можуть бути досягнуті щодо поточної позиції, регулюючи значення A і C векторів.

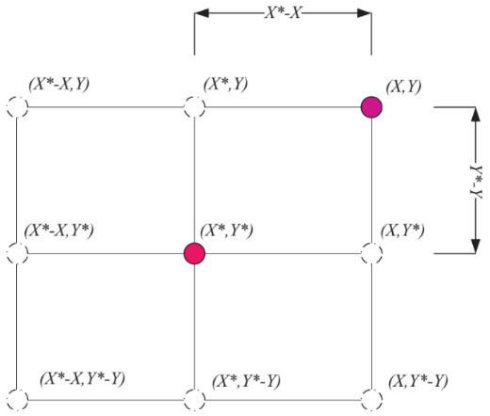

Рис.5. 2D вектори позицій і їх можливі наступні положення (X*, Y* - краще рішення, отримане на даний момент) [10]

Можлива позиція поновлення пошукового агента в тривимірному просторі також зображена на рис. 6.

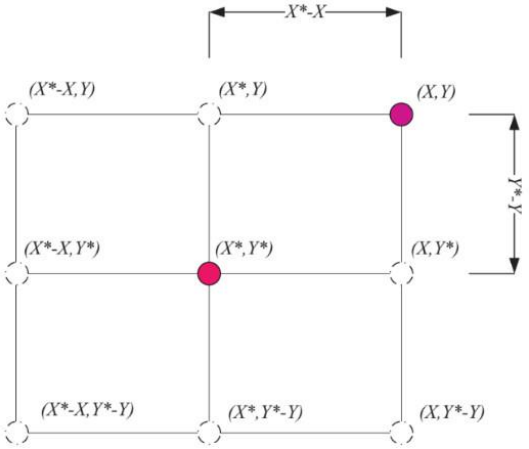

Рис.6. 3D вектори позицій і їх можливі наступні положення (X*, Y*, Z* - краще рішення, отримане на даний момент) [10]

Слід зазначити, що шляхом визначення випадкових векторів r_1 та r_2 можна досягти будь-якої позиції в пошуковому просторі, розташованої між ключовими точками, показаними на Рис. 5 та Рис. 6.

Отже, рівняння (8) дозволяє будь-якому пошуковому агенту оновлювати свою позицію навколо поточного кращого рішення, тим самим – імітує оточення жертви. Ту ж концепцію можна поширити на простір пошуку з n вимірами, і пошукові агенти будуть переміщатися у гіперкубі(ах) навколо кращого рішення, отриманого на даний момент.

Як згадувалося раніше, горбаті кити також атакують здобич, використовуючи стратегію пухирчастої сітки. Для математичного моделювання поведінки пухирчатої сітки горбатих китів розроблено два підходи: механізми скорочення оточення та пошуку за допомогою бульбашкової сітки.

Механізм скорочення оточення: така поведінка досягається зменшенням значення a в рівнянні (8). Зверніть увагу, що діапазон коливань A також зменшується на a. Іншими словами, A є випадковим значенням в інтервалі [$-a$, a], де a зменшується з 2 до 0 протягом ітерацій. Встановлюючи випадкові значення для A в діапазоні [-1,1], нову позицію пошукового агента можна визначити де завгодно, між початковою позицією агента та позицією поточного найкращого агента.

На Рис. 7 показано можливі положення від (X, Y) до (X∗, Y∗), яких можна досягти за допомогою 0≤ A ≤1 у двовимірному просторі.

Механізм спірального оновлення позицій. Як видно на Рис. 8, цей підхід спочатку обчислює відстань між китом, що знаходиться в (X, Y), і здобиччю, що знаходиться в (X∗, Y∗). Потім створюється спіральне рівняння між положенням кита і здобиччю, щоб імітувати спіральний рух горбатих китів, як показано нижче:

$$\vec{X}(t+1) = D' \cdot e^{bl} \cdot \cos(2\pi l) + \vec{X}*(t) \quad (9),$$

де

b – постійна, що визначає форму логарифмічної спіралі,

l – вказує на випадкове число в [−1, 1],

D' = |X∗ (t) – X (t)| - вказує на відстань між і-м китом та жертвою.

Зверніть увагу, що горбаті кити плавають навколо здобичі по звужуючому колі і по спіральній траєкторії одночасно. Щоб змоделювати цю одночасну поведінку, ми припускаємо, що існує 50%-а ймовірність вибору між механізмом скорочення оточення або спіральної моделі для оновлення положення китів в процесі оптимізації.

Це поведінка математично представлена наступними рівняннями (10):

$$\vec{X}(t+1) = \begin{cases} \vec{X}*(t) - \vec{A} \cdot \vec{D}, \text{якщо } p < 0,5 \\ D' \cdot e^{bl} \cdot \cos(2\pi l) + \vec{X}*(t), \text{якщо } p \geq 0,5 \end{cases} \quad (10)$$

де

p – це випадкове число в [0, 1],

t – представляє поточну ітерацію.

 https://doi.org/10.5281/zenodo.10451666

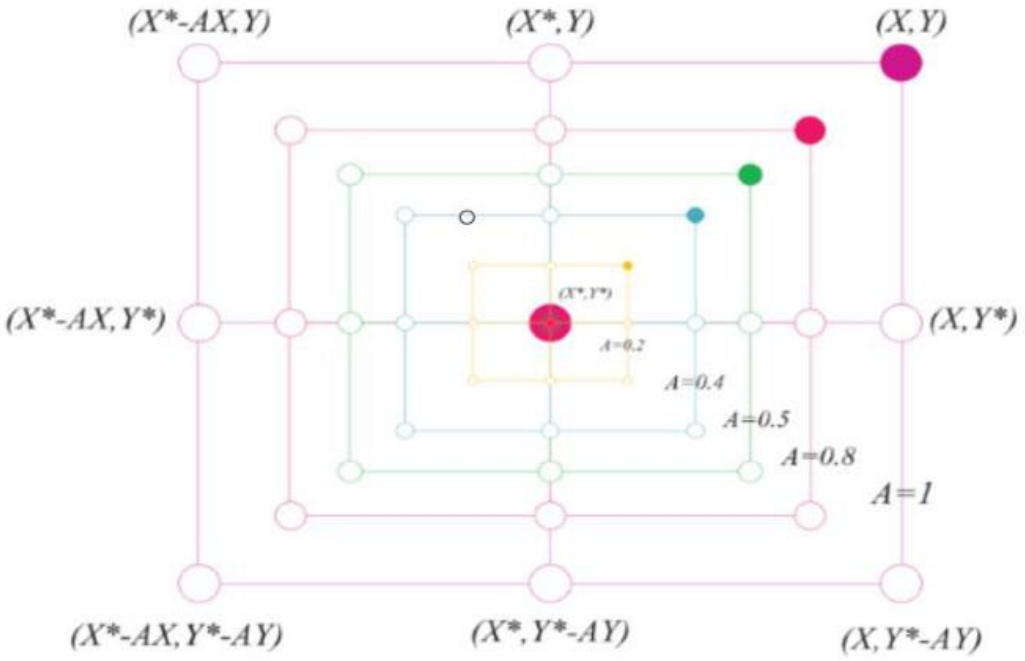

Рис.7. Механізм пошуку за допомогою бульбашкової сітки, реалізований в WOA (X* - найкраще рішення, отримане на даний момент), механізм скорочення оточення [10]

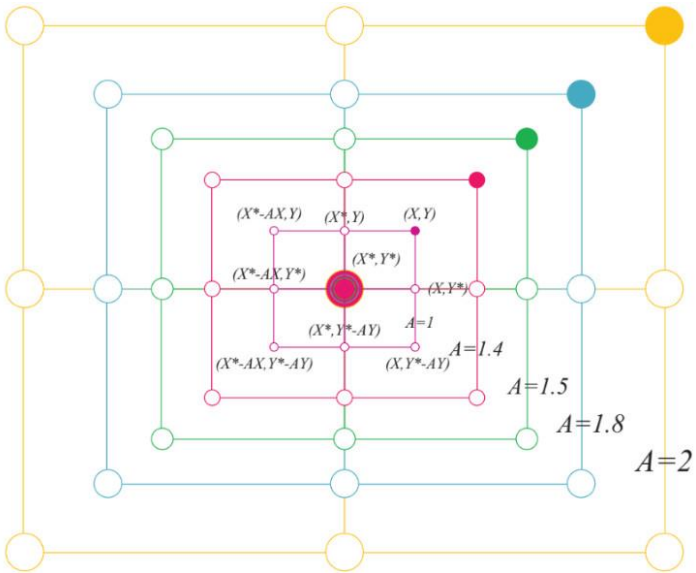

Рис.8. Механізм дослідження, реалізований в WOA (X* - це випадково обраний пошуковий агент)

B.2. ФАЗА РОЗВІДКИ

У разі, якщо задано проблему, WOA починає оптимізацію цієї проблеми, генеруючи колекцію випадкових рішень. На кожній ітерації агенти оновлюють своє розташування в залежності від випадково вибраного агента або найкращого агента, який виграв до цього часу. Для забезпечення фази дослідження інші агенти оновлюють свої позиції на основі найкращого рішення, яке опорною точкою, коли |A| > 1. У іншому випадку, коли |A| < 1, найкраще рішення виконує іншу роль з опорною точкою (Рис. 8).

Математична модель виглядає наступним чином (11-12):

ISSN 2786-9024

 https://doi.org/10.5281/zenodo.10451666

$$\vec{D} = \left| \vec{C} \cdot \vec{X}_{rand} - \vec{X}(t) \right| , (11)$$

$$\vec{X}(t+1) = \left| \vec{X}_{rand} - \vec{A} \cdot \vec{D} \right| , (12)$$

де

t – означає поточну ітерацію,

A – коефіцієнтний вектор,

$Xrand$ – вектор випадково вибраного кита (із поточної популяції).

Псевдокод WOA показано в Алгоритмі 2 [10].

Алгоритм 2

Ініціалізація китової популяції X_i $(i = 1, 2, \ldots n)$
Розрахунок придатності кожного агента
X^* = найкращий пошуковий агент
WHILE (*t* < максимальна к-сть ітерацій**)**
FOR EACH пошукового агенту
ОНОВЛЕННЯ ***a***, ***A***, ***C***, ***l*** та ***p***
IF1 (*p* < 0.5)
IF2 (|*A*| < 1)
Оновлення вектора позицій поточного пошукового агента за рівнянням **(7)**
ELSE IF2 (|*A*| ≥ 1)
Випадково вибрати пошукового агента **(xrand)**
Оновлення вектора позицій поточного пошукового агента за рівнянням **(12)**
ENDIF2
ELSE IF1 (*p* ≥ 0.5)
Оновлення вектора позицій поточного пошукового агента за рівнянням **(9)**
ENDIF1
ENDFOR
Перевірка кожного пошукового агента на вихід за границі
Розрахунок придатності кожного агента
Оновлення X*
t = t + 1
ENDWHILE
RETURN X^*

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

ЗАПРОПОНОВАНИЙ АЛГОРИТМ (HYGWWOA)

Підхід, запропонований у цьому дослідженні, в основному розроблений шляхом поєднання алгоритмів WOA та GWO з метою створення гібридного підходу, який називається "HyGWWOA". Процес поєднання здійснюється шляхом об'єднання переваг кожного з алгоритмів, які використовуються в цьому дослідженні. Основною перевагою GWO є те, що з кожного кластеру вибираються три найвищих значення, які вважаються α, β і δ, і ці значення представляють три найкращі рішення з цього кластеру, з урахуванням обмеження кількості учасників у групі в середньому від 5 до 12 вузлів. З іншого боку, у WOA немає обмежень для кількості учасників у групі, і, нарешті, у WOA немає топ-три найкращих рішень, а замість цього - одне. Враховуючи вищенаведене, HyGWWOA пропонується здійснити поєднання цих двох алгоритмів, застосовуючи алгоритм WOA для уникнення обмежень кількості учасників у групі та обираючи топ-три найвищих рішення, як це робить GWO.

Як і будь-який метаевристичний оптимізаційний алгоритм, запропонований алгоритм складається з двох етапів. Перший - це етап дослідження, тоді як другий - етап експлуатації.

У цьому дослідженні запропонований алгоритм HyGWWOA використовується для пошуку в усьому просторі пошуку з метою знаходження всіх можливих Xrand. Ці Xrand відіграють важливу роль в етапі дослідження. З іншого боку, якщо будь-який агент пошуку бачить здобич, вважається, що це Xrand, і цей процес пошуку здійснюється за допомогою створення бульбашок по шляху, який утворює форму спіралі, і його називають технікою полювання на мережу бульбашок. Таким чином, коли будь-який агент пошуку бачить бульбашки, це означає, що вони належать до цієї спіралі. У випадку, якщо одночасно є багато спіралей від різних Xrand, кожна з них розглядається як кластер в просторі пошуку, і в кожному з них є перші три найкращі рішення Xα, Xβ і Xδ. Ця перевага взята з алгоритму GWO, тоді як попередній сценарій представляє етап експлуатації.

На кожній ітерації, після визначення найкращих рішень Xα, Xβ і Xδ, інші агенти пошуку оновлюють свої позиції в кластері. Враховуючи значення абсолютного A, якщо воно більше одиниці, це означає, що агент пошуку виходить за межі кластера (спіралі), і йому потрібно шукати інший Xrand, щоб належати до нового кластера. В іншому випадку абсолютне значення A менше одиниці, що означає, що агент пошуку залишається в кластері і наближається до здобичі.

У цій роботі пропонується покращений підхід, який поєднує нещодавно запропоновані біоінспіровані оптимізаційні техніки, які були розроблені Мірджалілі та іншими, а саме алгоритми GWO і WOA, що імітують техніки полювання в природі. Потім для визначення пріоритетів вимог використовується алгоритм HyGWWOA. Для отримання пріоритету вимог застосовується алгоритм HyGWWOA. Алгоритм 3 представляє рекомендований псевдокод алгоритму HyGWWOA.

Алгоритм 3

BEGIN
Ініціалізувати популяцію агентів X_i $(i = 1, 2, 3 \ldots\ldots n)$

 https://doi.org/10.5281/zenodo.10451666

Ініціалізувати ***C***, ***r*** і ***a***
Випадковим чином вибрати X_{rand}
Розрахувати відстань між кожним китом **(*i*)** та усіма X_{rand} за рівнянням **(1)**
IF агент **(i)** не призначений
призначити агента **(i)** його найближчому X_{rand}
CALL FITNESS для кожного агента **(i)**
CALL CLUSTER
CALL RP
повернути найкраще рішення
ENDIF
END

A. ЕТАП ІНІЦІАЛІЗАЦІЇ

На початку ініціалізуйте популяцію агентів, як показано в Алгоритмі 3. Кількість агентів вибирається випадковим чином для генерації бульбашки, яка виглядає як конус (спіральна форма). Потім виміряйте відстань між усіма агентами та всіма Xrand, щоб призначити їх найближчому і з'єднати цю групу, щоб вона стала її членом.

B. ФІТНЕС-ФУНКЦІЯ

На основі рівняння (13), Xrand створює бульбашкову сітку, коли він бачить жертву, всі агенти, які бачать бульбашкову сітку, будуть асоціюватися з кластером (спіраллю). Отже, інші агенти, які приєдналися до кластера, повинні оновити свої розташування в напрямку розташування Xrand, як показано в Алгоритмі 4. З іншого боку, ці агенти оновлюють свої розташування в залежності від значення абсолютного A. У випадку, якщо |A| менше за одиницю, це означає, що агент все ще входить до кластера і оновлює своє розташування за допомогою рівняння (13). В іншому випадку, якщо |A| більше або дорівнює одиниці, це означає, що агент не належить до кластера, тому він шукає інший Xrand для асоціації, використовуючи рівняння (14). Ця поведінка представлена математично рівняннями (13, 14) та рівнянням (15).

$$\vec{X}(t+1) = D' \cdot e^{bl} \cdot \cos(2\pi l) + \vec{X}(t), \quad (13)$$

$$\vec{X}(t+1) = \vec{X}_{rand} - \vec{A} \cdot \vec{D}, \quad (14)$$

$$V = \frac{1}{3}\pi r h, \quad (15)$$

де

r – радіус бульбашкової сітки, який є постійною величиною (r = 12),

h – висота бульбашкової сітки, яка вибирається випадковим чином між 5 і 15.

Алгоритм 4. Функція FITNESS

PROCEDURE FITNESS
BEGIN
X_{rand} створює бульбашкову сітку за допомогою формули **(13)**
FOR EACH агента пошуку **(*i*)**
Оновити ***a***, ***A***, ***C*** та ***L***
Розрахувати відстань між кожним агентом **(*i*)** та X_{rand} за допомогою формули **(1)**
IF (A<1)
Оновити положення поточного агента **(*i*)** за допомогою формули **(13)**
ELSE
Вибрати новий X_{rand} випадковим чином
Оновити положення поточного агента **(*i*)** за допомогою формули **(14)**
ENDIF
ENDFOR
END

C. КЛАСТЕРИЗАЦІЯ

Як показано в Алгоритмі 5, кожний кластер K має Xrand, який вибирається випадковим чином. Функція пристосованості розраховується для кожного агента з метою перевірки, чи бачить цей агент бульбашку, яку створив Xrand. Іншими словами, цей агент все ще приєднаний до цієї спіралі. У випадку, якщо агент не бачить бульбашку, він шукає інший Xrand, до якого можна приєднатися. Таким чином, кожен кластер K отримує Xα, Xβ та Xδ, які представляють перші три найважливіші вимоги у цьому кластері.

Алгоритм 5. Функція CLUSTER

PROCEDURE CLUSTER
BEGIN
FOR EACH кластера *K*
Вибрати X_{rand} випадковим чином
Вибрати X_α, X_β та X_δ
WHILE (*t* ≤ *max_iteration*)
CALL FITNESS
CALL RP
ENDWHILE
RETURN X_α, X_β та X_δ
ENDFOR
END

D. ФУНКЦІЯ ПРІОРИТЕЗАЦІЇ ВИМОГ

Як вже обговорено на етапі кластеризації, кожний агент у просторі пошуку представляє вимогу з її важливістю; ця важливість відображає її значення в процесі розробки цільового проекту критичної інформаційної інфраструктури. Ця важливість розраховується відповідно до ранжування зацікавлених сторін для кожної вимоги з урахуванням важливості цієї зацікавленої сторони та її впливу на проект.

ISSN 2786-9024

 https://doi.org/10.5281/zenodo.10451666

Оскільки кожна зацікавлена сторона належить до ролі в цільовому середовищі, для якого будується система, ранг ролі має прямий вплив на важливість зацікавленої сторони. Таким чином, при розрахунках важливості зацікавленої сторони ранг ролі відіграє ключову роль.

Згідно з проектним середовищем, спершу буде розраховано важливість ролі на основі ранжування зацікавлених сторін за рівнем важливості зацікавленої сторони (16):

$$R_{\inf}(i) = \frac{RR_{\max} + 1 - Rank(R(i))}{\sum_{j=1}^{n} RR_{\max} + 1 - Rank(R(j))}, \quad (16)$$

де

$RR_{\max}$ - максимальний ранг в списку ролей,

$Rank(R(i)$ - ранг і-ї ролі,

n – кількість стейкхолдерів з однаковою роллю.

Далі важливість кожної зацікавленої сторони в кожній ролі розраховується для визначення впливу цієї зацікавленої сторони на проект (17):

$$ST_{\inf}(i) = \frac{RS_{\max} + 1 - Rank(i)}{\sum_{K=1}^{n} RS_{\max} + 1 - Rank(K)}, \quad (17)$$

де

і – представляє конкретну зацікавлену сторону,

RS_{max} – максимальний ранг зацікавлених сторін в даній ролі,

Rank(i) - ранг і-ої зацікавленої сторони у ролі,

n – загальна кількість зацікавлених сторін в тій же ролі.

Потім вплив цієї зацікавленої сторони на проект взагалі розраховується шляхом множення впливу ролі та впливу зацікавленої сторони, як показано в (18):

$$PR_{\inf}(i) = R_{\inf}(i) \cdot ST_{\inf}(i), \quad (18)$$

Оскільки кожна зацікавлена сторона ранжує список вимог, важливість цієї вимоги розраховується шляхом сумування впливу всіх зацікавлених сторін на проект, помноженого на її оцінку цієї вимоги (19):

$$R_{imp}(i) = \sum_{i=1}^{n} PR_{\inf}(i) \cdot r(i), \quad (19)$$

де

r(i) – представляє ранг і-ої зацікавленої сторони на цій вимозі,

n – загальна кількість зацікавлених сторін, які ранжують вимогу r(i).

Як показано в Алгоритмі 6, з кожного кластера будуть вибрані три найкращі значення. Цей процес буде виконуватися для кожного кластера ітеративно. Результатом цього процесу буде набір кращих рішень з кожного кластера, який буде збережений у тимчасовому списку. Отже, цей список містить α, β та δ з кластерів. Оскільки цей список містить найкращі рішення, три найкращі рішення будуть вибрані як перший результат процесу пріоритетизації. Вибрані результати будуть переміщені в остаточний відсортований список як три найкращі рішення. Оскільки вони вже вибрані та відсортовані за пріоритетом, ці вимоги будуть видалені зі створених кластерів.

Цей процес буде повторюватися до тих пір, поки в усіх кластерах не залишиться вимог. Як зазначалося вище, вибрані найкращі рішення будуть видалені зі створених кластерів, отже, кількість вимог у кластерах буде зменшуватися з кожною ітерацією.

Кінцевим результатом роботи запропонованого підходу буде повернення остаточного відсортованого набору вимог відповідно до їх важливості.

Алгоритм 6. Функція RP

PROCEDURE PR
BEGIN
  WHILE кожен кластер не порожній **(*1,2,3,...*K)**
    FOR EACH кластера ***K***
      Відсортувати вимоги на основі рівняння **(19)**
      Вибрати $\mathbf{X}_{\alpha}$, $\mathbf{X}_{\beta}$ і $\mathbf{X}_{\delta}$ та перемістити їх до **temp_list**
    ENDFOR
    Вибрати X_{α}, X_{β} і X_{δ} з відсортованого ***temp_list*** та перемістити їх до кінцевого відсортованого списку
    Видалити X_{α}, X_{β} і X_{δ} з оригінальних кластерів
  ENDWHILE
  Повернути кінцевий набір результатів
END

ЕКСПЕРИМЕНТАЛЬНІ РЕЗУЛЬТАТИ

Для оцінки запропонованого методу пріоритетизації вимог щодо точності було вибрано проект SKUDA як об'єкт вивчення в цій статті. SKUDA – це проект оновлення системи доступу до об'єктів критичної інфраструктури (ОКІ).

Багато будівель SKUDA потребують авторизованого доступу, таких як підстанції, серверні та комп'ютерні кластери.

Мета SKUDA полягає в заміні застарілих систем контролю доступу, об'єднанні різних існуючих засобів контролю доступу.

SKUDA – це досить великий проект. В ньому більше 20 груп зацікавлених осіб і приблизно 7 000 користувачів. До деяких з цих груп належали службовці з безпеки, розробники, менеджери та співробітники служб підтримки, таких як Департамент обслуговування та

 https://doi.org/10.5281/zenodo.10451666

управління майном, який відповідає за фізичний стан інфраструктури, Департамент людських ресурсів, що відповідає за інформацію про персонал, Департамент інформаційних технологій. У SKUDA близько 7 000 працівників та відвідувачів, які використовують систему для входу в будівлі та доступу на інфраструктурні об’єкти та отримання доступу до IT.

SKUDA має складну базу зацікавлених осіб, різні з яких мають протилежні вимоги.

У цьому дослідженні реалізовано роботу щодо вимог з урахуванням ранжування цілей проекту та відкинута робота щодо конкретних вимог, оскільки запропонований метод стосується пріоритетизації вимог.

Тестовий набір даних містить 53 вимоги, кожна з яких має свою важливість. Результати роботи показані в Таблиці 1, яка відображає впорядкований список вимог.

Запропонований підхід дає можливість впорядкувати вищевказаний набір вимог за важливістю з точністю 92%.

Таблиця 1. Пріоретизований список вимог

ID	Опис вимоги	Пріоритет
a.3	Все в 1 картці	1
a.1	Легший в використанні	2
a.2	Той самий спосіб контролю доступу для входу до ОКІ	3
c.3	Візуальний контроль	4
c.4	Контроль доступу, включаючи відстеження руху/журнали	5
c.5	Посилення контролю доступу до будівель	6
c.2	Контроль доступу до будівель ОКІ	7
c.1	Забезпечення відповідного доступу для кожної особи	8
d.5	Надання прав доступу	9
d.1	Швидкий випуск карток	10
d.2	Скорочення часу стояння в черзі	11
d.3	Статус ID-картки: можливість перевірити, чи отримав користувач ID-картку	12
d.6	Можливість створювати звіти про доступ	13
d.4	Легка заміна втрачених карток	14
d.7	Процедури боротьби з шахрайством	15
b.3	Картка має бути безпечною	16
b.1	Картка для включення даних користувача	17
b.4	Картка з брендом ОКІ	18
b.5	Легка ідентифікація/картка має чіткий вигляд	19
b.6	Карта має бути міцною/надійною	20
b.2	Картка з QR-кодом	21
b.7	Картка повинна бути привабливою	22
g.1	Централізоване управління інформацією доступу та ідентифікації	23
g.2	Експорт даних в інші системи	24
g.3	Імпорт даних з інших систем	25
g.4	Доступ до даних: можливість переглядати, оновлювати, видаляти віддалено та безпечно	26
g.5	Чіткі правила використання даних доступу	27
e.1	Економія на картках	28
e.2	Економія часу обробки	29
e.3	Скорочення паперових випробувань	30
f.5	Сумісність із поточною мережевою інфраструктурою	31
f.6	Вплив на інші системи	32
f.4	Сумісність з UPI	33
f.2	Сумісність з системами ОКІ	34
f.1	Сумісність з системою SKUDA	35
f.3	Сумісність з HR системою	36
h.3	Використання для входу в комп'ютер	37
h.2	Включення механізму оплати	38
h.4	Можливість оновлення (версії програмного забезпечення)	39
h.1	Додавання цифрового сертифікату	40

ISSN 2786-9024

h.5	Підвищення безпеки	41
j.2	Відповідність стандартам і законодавству	42
j.6	Наявність	43
j.5	Надійність	44
j.3	Технології	45
j.1	Безвідмовність	46
j.7	Мережева інфраструктура	47
j.4	Життєвий цикл	48
j.9	Вибраний виробник повинен мати перевірену історію відстеження в установах із технологіями контролю доступу, виготовлення ідентифікаційних карток, ODBC, смарт-карт.	49
j.8	Здатність прямого друку на обох сторонах картки, яка включатиме QR-код OKI	50
i.3	Діяльність з управління проектами	51
i.2	Технічна документація	52
i.1	Підтримка постачальника	53

ВИСНОВКИ

Інженерія вимог є найважливішою фазою у розробці проектів критичної інфраструктури, оскільки вона має справу з зацікавленими сторонами та іншими активностями. Оскільки кількість вимог змінюється для кожного проекту, процес пріоритизації вимог є важливим для встановлення порядку фаз проекту та задоволення зацікавлених сторін та кінцевих користувачів. У цій роботі було запропоновано гібридний підхід, який полягає в поєднанні переваг алгоритмів GWO та WOA як метаевристичних підходів для пріоритизації вимог проекту критичної інфраструктури.

Ця робота відкриває шлях для подальших досліджень в галузі пріоритизації вимог у сфері розробки проектів критичної інфраструктури та використання метаевристичних підходів для оптимізації процесів вирішення проблем у IT-галузі.

ЛІТЕРАТУРА

[1] M. S. Hasan, A.A. Mahmood, M.J.Alam, S.N.Hasan, & F.Rahman, “An evaluation of software requirement prioritization techniques”, *International Journal of Computer Science and Information Security* (IJCSIS), vol. 8, iss. 9, 2010.

[2] C.Duan, P.Laurent, J.Cleland-Huang, & C.Kwiatkowski, “Towards automated requirements prioritization and triage”, *Requirements engineering*, vol. 14, iss. 2, 73-89, 2009, doi: 10.1007/s00766-009-0079-7.

[3] R.Masadeh, A.Alzaqebah, & A.Sharieh, “Whale Optimization Algorithm For Solving The Maximum Flow Problem”, *Journal of Theoretical & Applied Information Technology*, vol. 96, iss. 8, 2018.

[4] A.K.Jain, M.N.Murty, & P.J.Flynn, “Data clustering: a review”, *ACM computing surveys* (CSUR), vol. 31, iss. 3, 264-323, 2019, doi: 10.1145/331499.331504.

[5] H.Emami, & F.Derakhshan, “Integrating fuzzy K-means, particle swarm optimization, and imperialist competitive algorithm for data clustering”, *Arabian Journal for Science and Engineering*, vol. 40, iss. 12, 3545-3554, 2015, doi: 10.1007/s13369-015-1826-3.

[6] F.Yang, T.Sun, & C.Zhang, “An efficient hybrid data clustering method based on K-harmonic means and Particle Swarm Optimization”, *Expert Systems with Applications*, vol. 36, iss. 6, 9847-9852, 2009, doi: 10.1016/j.eswa.2009.02.003

[7] J.Nayak, B.Naik, & H.S.Behera, “Fuzzy C-means (FCM) clustering algorithm: a decade review from 2000 to 2014”, *Computational intelligence in data mining*, vol. 2, 133-149,2015, doi: 10.1007/978-81-322-2208-8_14. Springer, New Delhi.

[8] Y.Kumar, & G.Sahoo, “Hybridization of magnetic charge system search and particle swarm optimization for efficient data clustering using neighborhood search strategy”, *Soft Computing*, vol. 19, iss. 12, 3621-3645, 2015, doi: 10.1007/s00500-015-1719-0.

[9] Q.H.Zhang, B.L.Li, Y.J.Liu, L.Gao, L.J.Liu, & X.L.Shi, “Data clustering using multivariant optimization algorithm”, *International Journal of Machine Learning and Cybernetics*, vol. 7, iss. 5, 773-782, 2016, doi: 10.1007/s13042-014-0294-5.

[10] S.Mirjalili, & A.Lewis, “The whale optimization algorithm”, *Advances in Engineering Software*, vol. 95, 51-67, 2016, doi: 10.1016/j.advengsoft.2016.01.008.

[11] A.Alzaqebah, R.Masadeh, & A.Hudaib, “Whale Optimization Algorithm for Requirements Prioritization”, *the 9th International Conference on Information and Communication Systems* (ICICS), 84-89, 2019, doi: 10.1109/IACS.2018.8355446.

[12] S.Chander, P.Vijaya, & P.Dhyani, “Multi kernel and dynamic fractional lion optimization algorithm for data clustering”, *Alexandria engineering journal*, vol. 57, iss. 1, 267-276, 2018, doi: 10.1016/j.aej.2016.12.013.

[13] E.Çomak, “A modified particle swarm optimization algorithm using Renyi entropy-based clustering”, *Neural Computing and Applications*, vol. 27, iss. 5, 1381-1390, 2016, doi: 10.1007/s00521-015-1941-9.

[14] S.Mirjalili, S.M.Mirjalili, & A.Lewis, "Grey wolf optimizer", *Advances in engineering software*, vol. 69, 46-61, 2014, doi: 10.1016/j.advengsoft.2013.12.007.
[15] J.Karlsson, C.Wohlin, & B.Regnell, "An evaluation of methods for prioritizing software requirements", *Information and software technology*, vol. 39, iss. 14-15, 939-947, 1998, doi: 10.1016/S0950-5849(97)00053-0.
[16] D.Leffingwell, D.Widrig, "Managing Software Requirements: A Unified Approach". Upper Saddle River:Addison- Wesley, 2009.
[17] S.Hatton, "Early prioritisation of goals", *International Conference on Conceptual Modeling*, 235-244, 2007, doi: 10.1007/978-3-540-76292-8_29. Springer, Berlin, Heidelberg.
[18] B.Regnell, M.Höst, J.Natt och Dag, P.Beremark, T.Hjelm, "An industrial case study on distributed prioritization in market-driven requirements engineering for packaged software", *Requirements Engineering*, vol. 6, iss. 1, 51-62, 2001, doi: 10.1007/s007660170015.
[19] T.L.Saaty, "The analytic hierarchy process". McGraw-Hill, New York, 1980.
[20] L.Lehtola, & M.Kauppinen, "Empirical evaluation of two requirements prioritization methods in product development projects", *European Conference on Software Process Improvement*, 161-170, doi: 10.1007/978-3-540-30181-3_15. Springer, Berlin, Heidelberg, 2004.
[21] L.Lehtola, "Suitability of requirements prioritization methods for market-driven software product development", *Softw. Process Improve. Pract.*, vol. 11, 7-19, 2006, doi: 10.1002/spip.249.
[22] Y.Shen, A.E.Hoerl, & W.Mcconnell, "An incomplete design in the analytic hierarchy process", *Mathematical and Computer Modelling: An International Journal*, vol. 16, iss. 5, 121-129, 1992.
[23] P.T.Harker, "Incomplete pairwise comparisons in the analytic hierarchy process" , *Mathematical Modelling*, vol. 9, iss. 11, 837-848, 1998, doi: 10.1016/0270-0255(87)90503-3.
[24] J.Karlsson, S.Olsson, & K.Ryan, "Improved practical support for large-scale requirements prioritising", *Requirements Engineering*, vol. 2, iss. 1, 51-60, 1997, doi: 10.1007/BF02802897.
[25] D.Leffingwell & D.Widrig, "Managing Software Requirements: A Unified Approach". Upper Saddle River:Addison- Wesley, 1999.
[26] DSDM Project Framework. [Online]. URL: http://surl.li/rlcbf. Accessed: 09.11.2023.
[27] A.V.Aho, J.E.Hopcroft, & J.Ullman, "Data structures and algorithms". Addison-Wesley Longman Publishing Co., Inc., 1993.
[28] S.Lauesen, "Software requirements – styles and techniques". Pearson Education, Essex, 2002.
[29] R.Masadeh, A.Alzaqebah & A.Hudaib, "Grey Wolf Algorithm for Requirements Prioritization", *Modern Applied Science*, vol. 12, iss. 2, 54, 2018, doi: 10.5539/mas.v12n2p.
[30] S.L.Lim, "Social networks and collaborative filtering for large-scale requirements elicitation", doctoral dissertation, University of New South Wales, 2011. [Online]. URL: http://surl.li/rlcck. Accessed: 09.11.2023.

REQUIREMENT PRIORITIZATION IN THE DEVELOPMENT OF SOFTWARE PROJECTS FOR CRITICAL INFRASTRUCTURE OBJECTS

Iaroslav Dorohyi, Olena Doroha-Ivaniuk

The objective of the study is to develop an algorithm for prioritizing requirements in the development of software for critical infrastructure object projects. Requirement development is a fundamental phase in any software project, as this phase involves the identification, processing, and manipulation of requirements. The primary source of these requirements is project stakeholders, taking into account project constraints and limits. The number of requirements varies for each software project for a critical infrastructure object, hence the term requirement prioritization pertains to determining the priority order of executing software requirements based on considerations and decisions of stakeholders.

Various proposed optimization algorithms are employed to address optimization tasks. This paper presents the main stages of basic optimization algorithms, some of their modifications aimed at enhancing their efficiency in solving such types of problems. Additionally, a hybrid approach based on WOA and GWO optimization algorithms is proposed, combining the advantages of each algorithm to determine the priority of requirements for critical infrastructure object software. Furthermore, a dataset from the SKUDA project is provided, utilized in this research, meeting the requirements of a real software project for evaluating the proposed method.

The scientific novelty lies in the modification, application, and combination of results from well-known GWO and WOA algorithms to address the requirement prioritization task for critical infrastructure object software projects. The proposed algorithm achieves an accuracy of 92% for the proposed set of requirements.

Keywords: *requirement prioritization, WOA (Whale Optimization Algorithm), GWO (Grey Wolf Optimization), critical infrastructure object, CI (Critical Infrastructure), hybrid approach.*

REFERENCES

[1] M. S. Hasan, A.A. Mahmood, M.J.Alam, S.N.Hasan, & F.Rahman, "An evaluation of software requirement prioritization techniques", *International Journal of Computer Science and Information Security* (IJCSIS), vol. 8, iss. 9, 2010.
[2] C.Duan, P.Laurent, J.Cleland-Huang, & C.Kwiatkowski, "Towards automated requirements prioritization and triage", *Requirements engineering*, vol. 14, iss. 2, 73-89, 2009, doi: 10.1007/s00766-009-0079-7.
[3] R.Masadeh, A.Alzaqebah, & A.Sharieh, "Whale Optimization Algorithm For Solving The Maximum Flow Problem", *Journal of Theoretical & Applied Information Technology*, vol. 96, iss. 8, 2018.

ISSN 2786-9024

[4] A.K.Jain, M.N.Murty, & P.J.Flynn, "Data clustering: a review", *ACM computing surveys* (CSUR), vol. 31, iss. 3, 264-323, 2019, doi: 10.1145/331499.331504.

[5] H.Emami, & F.Derakhshan, "Integrating fuzzy K-means, particle swarm optimization, and imperialist competitive algorithm for data clustering", *Arabian Journal for Science and Engineering*, vol. 40, iss. 12, 3545-3554, 2015, doi: 10.1007/s13369-015-1826-3.

[6] F.Yang, T.Sun, & C.Zhang, "An efficient hybrid data clustering method based on K-harmonic means and Particle Swarm Optimization", *Expert Systems with Applications*, vol. 36, iss. 6, 9847-9852, 2009, doi: 10.1016/j.eswa.2009.02.003

[7] J.Nayak, B.Naik, & H.S.Behera, "Fuzzy C-means (FCM) clustering algorithm: a decade review from 2000 to 2014", *Computational intelligence in data mining*, vol. 2, 133-149,2015, doi: 10.1007/978-81-322-2208-8_14. Springer, New Delhi.

[8] Y.Kumar, & G.Sahoo, "Hybridization of magnetic charge system search and particle swarm optimization for efficient data clustering using neighborhood search strategy", *Soft Computing*, vol. 19, iss. 12, 3621-3645, 2015, doi: 10.1007/s00500-015-1719-0.

[9] Q.H.Zhang, B.L.Li, Y.J.Liu, L.Gao, L.J.Liu, & X.L.Shi, "Data clustering using multivariant optimization algorithm", *International Journal of Machine Learning and Cybernetics*, vol. 7, iss. 5, 773-782, 2016, doi: 10.1007/s13042-014-0294-5.

[10] S.Mirjalili, & A.Lewis, "The whale optimization algorithm", *Advances in Engineering Software*, vol. 95, 51-67, 2016, doi: 10.1016/j.advengsoft.2016.01.008.

[11] A.Alzaqebah, R.Masadeh, & A.Hudaib, "Whale Optimization Algorithm for Requirements Prioritization", *the 9th International Conference on Information and Communication Systems* (ICICS), 84-89, 2019, doi: 10.1109/IACS.2018.8355446.

[12] S.Chander, P.Vijaya, & P.Dhyani, "Multi kernel and dynamic fractional lion optimization algorithm for data clustering", *Alexandria engineering journal*, vol. 57, iss. 1, 267-276, 2018, doi: 10.1016/j.aej.2016.12.013.

[13] E.Çomak, "A modified particle swarm optimization algorithm using Renyi entropy-based clustering", *Neural Computing and Applications*, vol. 27, iss. 5, 1381-1390, 2016, doi: 10.1007/s00521-015-1941-9.

[14] S.Mirjalili, S.M.Mirjalili, & A.Lewis, "Grey wolf optimizer", *Advances in engineering software*, vol. 69, 46-61, 2014, doi: 10.1016/j.advengsoft.2013.12.007.

[15] J.Karlsson, C.Wohlin, & B.Regnell, "An evaluation of methods for prioritizing software requirements", *Information and software technology*, vol. 39, iss. 14-15, 939-947, 1998, doi: 10.1016/S0950-5849(97)00053-0.

[16] D.Leffingwell, D.Widrig, "Managing Software Requirements: A Unified Approach". Upper Saddle River:Addison- Wesley, 2009.

[17] S.Hatton, "Early prioritisation of goals", *International Conference on Conceptual Modeling*, 235-244, 2007, doi: 10.1007/978-3-540-76292-8_29. Springer, Berlin, Heidelberg.

[18] B.Regnell, M.Höst, J.Natt och Dag, P.Beremark, T.Hjelm, "An industrial case study on distributed prioritization in market-driven requirements engineering for packaged software", *Requirements Engineering*, vol. 6, iss. 1, 51-62, 2001, doi: 10.1007/s007660170015.

[19] T.L.Saaty, "The analytic hierarchy process". McGraw-Hill, New York, 1980.

[20] L.Lehtola, & M.Kauppinen, "Empirical evaluation of two requirements prioritization methods in product development projects", *European Conference on Software Process Improvement*, 161-170, doi: 10.1007/978-3-540-30181-3_15. Springer, Berlin, Heidelberg, 2004.

[21] L.Lehtola, "Suitability of requirements prioritization methods for market-driven software product development", *Softw. Process Improve. Pract.*, vol. 11, 7-19, 2006, doi: 10.1002/spip.249.

[22] Y.Shen, A.E.Hoerl, & W.Mcconnell, "An incomplete design in the analytic hierarchy process", *Mathematical and Computer Modelling: An International Journal*, vol. 16, iss. 5, 121-129, 1992.

[23] P.T.Harker, "Incomplete pairwise comparisons in the analytic hierarchy process" , *Mathematical Modelling*, vol. 9, iss. 11, 837-848, 1998, doi: 10.1016/0270-0255(87)90503-3.

[24] J.Karlsson, S.Olsson, & K.Ryan, "Improved practical support for large-scale requirements prioritising", *Requirements Engineering*, vol. 2, iss. 1, 51-60, 1997, doi: 10.1007/BF02802897.

[25] D.Leffingwell & D.Widrig, "Managing Software Requirements: A Unified Approach". Upper Saddle River:Addison- Wesley, 1999.

[26] DSDM Project Framework. [Online]. URL: http://surl.li/rlcbf. Accessed: 09.11.2023.

[27] A.V.Aho, J.E.Hopcroft, & J.Ullman, "Data structures and algorithms". Addison-Wesley Longman Publishing Co., Inc., 1993.

[28] S.Lauesen, "Software requirements – styles and techniques". Pearson Education, Essex, 2002.

[29] R.Masadeh, A.Alzaqebah & A.Hudaib, "Grey Wolf Algorithm for Requirements Prioritization", *Modern Applied Science*, vol. 12, iss. 2, 54, 2018, doi: 10.5539/mas.v12n2p.

[30] S.L.Lim, "Social networks and collaborative filtering for large-scale requirements elicitation", doctoral dissertation, University of New South Wales, 2011. [Online]. URL: http://surl.li/rlcck. Accessed: 09.11.2023.