НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2026 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

на тему: «Порівняння метаевристичних методів оптимізації багатоекстремальних функцій»

Виконав:
Студент IV курсу, групи КА-21 _____
Ніколаєв Ігор Дмитрович
Керівник:
Доцент кафедри ММСА, д.т.н.
Савченко Ілля Олександрович _____
Консультант з економічного розділу:
Доцент кафедри ТПЕ, к.е.н.
Рощина Надія Василівна _____
Консультант з нормоконтролю:
Асистент кафедри ММСА, д-р філос.
Канцедал Георгій Олегович _____
Рецензент:
Доцент кафедри ШІ, к.т.н.
Шаповал Наталії Віталіївни _____

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

Київ – 2026 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 124 «Системний аналіз»
Освітня програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2026 р.

ЗАВДАННЯ
на дипломну роботу студенту
Ніколаєву Ігорю Дмитровичу

1. Тема роботи **«Порівняння метаевристичних методів оптимізації багатоекстремальних функцій»**, керівник роботи доцент кафедри ММСА, д.т.н. Савченко Ілля Олександрович, затверджені наказом по університету від 27.05.2026 р. №1944-с
2. Термін подання студентом роботи «10» червня 2026 р.
3. Вихідні дані до роботи: тестові багатоекстремальні функції, прикладна задача розміщення складів, методи метаевристичної оптимізації.
4. Зміст роботи: аналіз задач глобальної оптимізації багатоекстремальних функцій, огляд метаевристичних методів, математичний опис генетичного алгоритму та методу спірального пошуку, програмна реалізація алгоритмів, експериментальне порівняння на тестових функціях і задачі розміщення складів, функціонально-вартісний аналіз.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): графіки тестових функцій, графіки збіжності алгоритмів, таблиці результатів експериментів, візуалізація розміщення складів, таблиці та графіки до функціонально-вартісного аналізу, презентація.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Функціонально-вартісний аналіз	Рощина Н.В., доц., к.е.н.		

7. Дата видачі завдання 12.04.2026

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Вибір тематики дослідження	18.02.2026	Виконано
2	Аналіз актуальності та дослідження тематичної літератури	22.03.2026	Виконано
3	Формулювання завдання та огляд сучасних методів оптимізації	20.04.2026	Виконано
4	Математичний опис генетичного алгоритму та методу спірального пошуку	27.04.2026	Виконано
5	Розробка програмної реалізації алгоритмів	4.05.2026	Виконано
6	Проведення експериментів та аналіз результатів	11.05.2026	Виконано
7	Підготовка економічної частини	25.05.2026	Виконано
8	Оформлення пояснювальної записки	01.06.2026	Виконано
9	Підготовка презентації до дипломної роботи	04.06.2026	Виконано

Студент _____ Ігор НІКОЛАЄВ

Керівник _____ Ілля САВЧЕНКО

РЕФЕРАТ

Дипломна робота: 158 с., 18 рис., 11 табл., 3 додатки, 15 джерел.

ГЛОБАЛЬНА ОПТИМІЗАЦІЯ, БАГАТОЕКСТРЕМАЛЬНІ ФУНКЦІЇ, МЕТАЕВРИСТИЧНІ МЕТОДИ, ГЕНЕТИЧНИЙ АЛГОРИТМ, МЕТОД СПІРАЛЬНОГО ПОШУКУ, ФУНКЦІЯ ШВЕФЕЛЯ, ФУНКЦІЯ МІХАЛЕВИЧА, ЗАДАЧА РОЗМІЩЕННЯ СКЛАДІВ.

Об’єкт дослідження – процеси глобальної оптимізації складних багатоекстремальних функцій.

Предмет дослідження – метаевристичні методи глобальної оптимізації, зокрема генетичний алгоритм та метод спірального пошуку, а також програмні засоби їх реалізації для дослідження тестових і прикладних цільових функцій.

Мета дослідження – дослідити особливості задач глобальної оптимізації багатоекстремальних функцій, обґрунтувати доцільність використання метаевристичних методів, реалізувати генетичний алгоритм та метод спірального пошуку, провести їх експериментальне порівняння на тестових функціях Швефеля і Міхалевича та прикладній задачі розміщення складів.

За результатами виконаних досліджень узагальнено теоретичні відомості про задачі глобальної оптимізації, багатоекстремальні функції та обмеження традиційних детермінованих і градієнтних методів. Розглянуто математичний апарат генетичного алгоритму та методу спірального пошуку, визначено їх основні параметри й особливості застосування. Для експериментального дослідження використано тестові багатоекстремальні функції Швефеля та Міхалевича, а також сформульовано прикладну задачу розміщення складів як задачу мінімізації сумарних транспортних витрат.

Програмно реалізовано генетичний алгоритм із різними способами селекції, операторами кросоверу та мутації, а також метод спірального пошуку з урахуванням параметрів кута повороту, коефіцієнта наближення та кількості

точок-кандидатів. Проведена серія обчислювальних експериментів дозволила порівняти методи за якістю знайденого розв’язку, стабільністю результатів, успішністю оптимізації та кількістю обчислень цільової функції.

Практичне значення роботи полягає у можливості застосування отриманих результатів для вибору ефективних метаевристичних алгоритмів при розв’язанні складних задач глобальної оптимізації, зокрема задач логістичного типу.

Програмна реалізація розроблена мовою програмування Python.

ABSTRACT

Bachelor's thesis: 158 pages, 18 figures, 11 tables, 3 appendices, 15 references.

GLOBAL OPTIMIZATION, MULTIMODAL FUNCTIONS, METAHEURISTIC METHODS, GENETIC ALGORITHM, SPIRAL OPTIMIZATION ALGORITHM, SCHWEFEL FUNCTION, MICHALEWICZ FUNCTION, WAREHOUSE LOCATION PROBLEM.

Object of research – processes of global optimization of complex multimodal functions.

Subject of research – metaheuristic methods of global optimization, in particular the genetic algorithm and the spiral optimization algorithm, as well as software tools for their implementation in the study of test and applied objective functions.

Purpose of research – to investigate the features of global optimization problems for multimodal functions, justify the feasibility of using metaheuristic methods, implement the genetic algorithm and the spiral optimization algorithm, and conduct their experimental comparison on the Schwefel and Michalewicz test functions as well as on the applied warehouse location problem.

Based on the conducted research, theoretical information on global optimization problems, multimodal functions, and the limitations of traditional deterministic and gradient-based optimization methods was summarized. The mathematical apparatus of the genetic algorithm and the spiral optimization algorithm was considered, and their main parameters and application features were determined. For the experimental study, the Schwefel and Michalewicz test multimodal functions were used, and the applied warehouse location problem was formulated as a problem of minimizing total transportation costs.

A genetic algorithm with different selection methods, crossover and mutation operators, as well as the spiral optimization algorithm with parameters such as

rotation angle, convergence coefficient, and number of candidate points were implemented in software. A series of computational experiments made it possible to compare the methods in terms of the quality of the obtained solution, result stability, optimization success rate, and the number of objective function evaluations.

The practical significance of the work lies in the possibility of applying the obtained results to select effective metaheuristic algorithms for solving complex global optimization problems, in particular logistics-type problems.

The software implementation was developed in the Python programming language.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ГА – генетичний алгоритм.

СП – спіральний пошук.

ВСТУП

Сучасний розвиток інформаційних технологій, системного аналізу, штучного інтелекту, логістики, інженерного проєктування та обчислювальної математики супроводжується зростанням кількості складних задач оптимізації. У багатьох практичних випадках необхідно знайти такий набір параметрів, який забезпечує найкращу якість роботи системи, мінімальні витрати, максимальну ефективність або оптимальне використання наявних ресурсів.

Задачі оптимізації виникають у різних прикладних сферах: машинному навчанні, транспортній логістиці, економіці, автоматизованому проєктуванні, плануванні виробництва, управлінні складними системами та прийнятті рішень. Значна частина таких задач характеризується складною структурою цільової функції, великою кількістю параметрів, нелінійністю, обмеженнями та наявністю багатьох локальних екстремумів.

Особливу складність становлять задачі оптимізації багатоекстремальних функцій. Такі функції мають декілька локальних мінімумів або максимумів, що суттєво ускладнює процес пошуку глобального оптимуму. У процесі оптимізації алгоритм може передчасно збігатися до локального мінімуму, який не є найкращим розв’язком на всій області пошуку. Саме тому для таких задач важливо застосовувати методи, здатні виконувати глобальне дослідження простору розв’язків.

Традиційні детерміновані та градієнтні методи оптимізації часто демонструють недостатню ефективність при розв’язанні багатоекстремальних задач. Їх робота переважно базується на локальній інформації про поведінку функції, зокрема на значеннях похідних або градієнта. Через це такі методи суттєво залежать від вибору початкової точки та можуть завершувати роботу

в локальному мінімумі. Крім того, у багатьох практичних задачах цільова функція може бути недиференційованою або заданою неявно, що додатково ускладнює застосування класичних методів.

У зв’язку з цим значного поширення набули метаевристичні методи оптимізації. Вони використовують стохастичні або евристичні механізми пошуку, не потребують обчислення похідних і можуть застосовуватися до складних нелінійних, недиференційованих та багатоекстремальних функцій. До таких методів належать генетичні алгоритми, алгоритми рою частинок, метод імітації відпалу, мурашині алгоритми, диференціальна еволюція та метод спірального пошуку.

У даній роботі досліджуються два метаевристичні підходи: генетичний алгоритм та метод спірального пошуку [1]. Генетичний алгоритм базується на принципах еволюційного пошуку та використовує механізми селекції, кросоверу і мутації. Метод спірального пошуку використовує геометричну модель руху точок-кандидатів навколо поточного найкращого розв’язку за спіральною траєкторією.

Для оцінювання ефективності досліджуваних методів у роботі використовуються класичні тестові багатоекстремальні функції Швефеля та Міхалевича. Ці функції мають складний рельєф і велику кількість локальних екстремумів, що дозволяє перевірити здатність алгоритмів виконувати глобальний пошук і уникати передчасної збіжності.

Окрім тестових функцій, у роботі також розглядається прикладна задача розміщення складів. Вона формулюється як задача мінімізації сумарних транспортних витрат з урахуванням координат міст, попиту та відстаней до найближчих складів. Така задача має практичний логістичний зміст і може бути подана як багатоекстремальна цільова функція, оскільки різні конфігурації складів можуть утворювати локально оптимальні розв’язки.

Актуальність теми полягає у необхідності дослідження ефективності метаевристичних методів при розв’язанні задач глобальної оптимізації

багатоекстремальних функцій як тестового, так і прикладного характеру. Порівняння генетичного алгоритму та методу спірального пошуку дозволяє визначити їх переваги, недоліки, особливості збіжності та доцільність використання в різних типах задач.

Метою роботи є дослідження та порівняння ефективності генетичного алгоритму та методу спірального пошуку при оптимізації багатоекстремальних функцій, зокрема тестових функцій Швефеля і Міхалевича та прикладної цільової функції задачі розміщення складів.

Для досягнення поставленої мети необхідно розв’язати такі задачі:

1) проаналізувати особливості задач глобальної оптимізації багатоекстремальних функцій;
2) розглянути обмеження традиційних детермінованих та градієнтних методів оптимізації;
3) дослідити принципи роботи генетичного алгоритму;
4) дослідити математичну модель методу спірального пошуку;
5) розглянути тестові багатоекстремальні функції Швефеля та Міхалевича;
6) сформулювати прикладну задачу розміщення складів як задачу глобальної оптимізації;
7) реалізувати програмні засоби для дослідження алгоритмів;
8) провести експериментальне порівняння генетичного алгоритму та методу спірального пошуку;
9) оцінити ефективність алгоритмів за якістю знайденого розв’язку, стабільністю результатів та кількістю обчислень цільової функції.

Об’єктом дослідження є процеси глобальної оптимізації складних багатоекстремальних функцій.

Предметом дослідження є метаевристичні методи глобальної оптимізації, зокрема генетичний алгоритм та метод спірального пошуку, а

також програмні засоби їх реалізації для дослідження тестових і прикладних цільових функцій.

Методи дослідження включають методи математичного моделювання, чисельної оптимізації, еволюційні алгоритми, метод спірального пошуку, експериментальне моделювання та порівняльний аналіз результатів.

Практичне значення роботи полягає у можливості застосування отриманих результатів для вибору ефективних метаевристичних алгоритмів при розв’язанні складних задач глобальної оптимізації. Додавання задачі розміщення складів дозволяє продемонструвати застосування досліджених методів не лише на тестових математичних функціях, а й на прикладній задачі логістичного характеру.

РОЗДІЛ 1. ТЕОРЕТИЧНИЙ АПАРАТ МЕТАЕВРИСТИЧНОЇ ОПТИМІЗАЦІЇ БАГАТОЕКСТРЕМАЛЬНИХ ФУНКЦІЙ

1.1. Постановка задачі глобальної оптимізації мультимодальних функцій

Однією з важливих задач сучасної прикладної математики, комп'ютерних наук та інформаційних технологій є задача оптимізації. У загальному випадку оптимізація полягає у знаходженні найкращого розв'язку серед множини можливих варіантів відповідно до заданого критерію ефективності. Більшість практичних задач у галузях машинного навчання, автоматизованого проєктування, робототехніки, логістики та економіки можуть бути зведені до задач оптимізації.

З математичної точки зору задача оптимізації формулюється як задача пошуку такого вектора параметрів x, при якому цільова функція $f(x)$ набуває мінімального або максимального значення в межах області допустимих розв'язків D [2]:

$$min_{x \in D} f(x),$$

де $f(x) -$ цільова функція;

$x = (x_1, x_2, \dots, x_n) -$ вектор змінних;

$D -$ область допустимих значень параметрів.

У задачах глобальної оптимізації необхідно знайти глобальний екстремум функції, тобто найкраще значення на всій області пошуку. Глобальний мінімум визначається умовою:

$$f(x^*) \leq f(x), \qquad \forall x \in D,$$

де $x^* -$ точка глобального мінімуму функції.

На практиці значна кількість задач характеризується складною структурою цільових функцій. Такі функції можуть містити велику кількість локальних мінімумів та максимумів, що суттєво ускладнює процес пошуку глобального розв’язку. Функції такого типу називаються багатоекстремальними або мультимодальними.

Особливістю мультимодальних функцій є наявність декількох областей локальної оптимальності. У процесі оптимізації алгоритм може передчасно збігатися до локального мінімуму, який не є глобальним. Це призводить до отримання неоптимального результату та зниження ефективності методу оптимізації.

Основною проблемою задач глобальної оптимізації є необхідність забезпечення балансу між дослідженням нових областей простору пошуку та уточненням уже знайдених перспективних розв’язків. Надмірна локалізація пошуку підвищує ризик потрапляння до локального мінімуму, тоді як надмірна випадковість може значно збільшити час збіжності алгоритму.

Для розв’язання задач оптимізації мультимодальних функцій широкого застосування набули метаевристичні методи оптимізації. На відміну від класичних детермінованих підходів, метаевристичні алгоритми використовують стохастичні механізми дослідження простору пошуку, що дозволяє підвищити ймовірність знаходження глобального екстремуму.

У даній роботі досліджуються методи глобальної оптимізації багатоекстремальних функцій. Для цього розглядаються як класичні тестові функції, так і прикладна цільова функція задачі розміщення складів, яка дозволяє продемонструвати застосування метаевристичних методів у практичній задачі логістичного типу.

1.2. Особливості багатоекстремальних функцій

Багатоекстремальною або мультимодальною називають функцію, яка має декілька локальних мінімумів або максимумів у межах заданої області пошуку. У задачах мінімізації це означає, що функція може мати багато локальних мінімумів, але лише один або декілька з них є глобальними.

Багатоекстремальні функції становлять значну складність для алгоритмів оптимізації, оскільки наявність локальних мінімумів може призводити до передчасної збіжності алгоритму. Якщо метод орієнтується лише на локальну інформацію про функцію, він може зупинитися в локальному мінімумі та не знайти глобальний оптимум.

Основними властивостями багатоекстремальних функцій є:

1) наявність кількох локальних мінімумів або максимумів;
2) складна форма поверхні цільової функції;
3) висока залежність результатів від початкової точки;
4) можливість передчасної збіжності алгоритму;
5) складність побудови універсального методу пошуку глобального оптимуму.

У тестових задачах багатоекстремальність зазвичай створюється спеціально для перевірки ефективності алгоритмів. До таких функцій належать функції Швефеля, Міхалевича, Растригіна, Акерлі та інші [3]. Вони мають складний рельєф і дозволяють оцінити здатність алгоритму уникати локальних мінімумів.

У прикладних задачах багатоекстремальність може виникати природно. Наприклад, у задачах розміщення об’єктів, кластеризації, маршрутизації або планування функція якості може мати багато різних локально оптимальних конфігурацій. Кожна така конфігурація відповідає певному розподілу ресурсів або певному розміщенню об’єктів у просторі.

Для задач глобальної оптимізації важливим є баланс між двома процесами: глобальним дослідженням простору пошуку та локальним уточненням знайденого розв’язку. Глобальне дослідження дозволяє виявляти перспективні області простору, тоді як локальне уточнення забезпечує підвищення точності розв’язку. Надмірна локалізація може призвести до застрягання в локальному мінімумі, а надмірна випадковість — до повільної збіжності алгоритму.

Саме тому для оптимізації багатоекстремальних функцій часто використовують метаевристичні методи. Вони не гарантують знаходження точного глобального оптимуму в усіх випадках, однак завдяки стохастичному характеру пошуку дозволяють підвищити ймовірність знаходження якісного розв’язку в складних багатовимірних просторах.

1.3. Обмеження традиційних детермінованих та градієнтних методів оптимізації

Задачі глобальної оптимізації багатоекстремальних функцій характеризуються складною структурою простору пошуку та наявністю великої кількості локальних екстремумів. У таких умовах традиційні детерміновані та градієнтні методи оптимізації часто демонструють недостатню ефективність, оскільки їх робота переважно орієнтована на пошук локального оптимуму.

До традиційних методів оптимізації належать методи прямого перебору, координатного спуску, метод Ньютона, метод найшвидшого спуску та градієнтний спуск. Більшість таких методів використовують локальну інформацію про поведінку функції, зокрема значення похідних або градієнта, для визначення напряму подальшого пошуку [4].

У задачі мінімізації цільової функції градієнтний метод базується на послідовному оновленні параметрів за формулою:

$$x_{k+1} = \quad -\alpha \nabla f(x_k),$$

де $x_k -$ поточний вектор параметрів;

$\alpha -$ коефіцієнт навчання;

$\nabla f(x_k) -$ градієнт функції у точці x_k.

Основна ідея градієнтного спуску полягає у русі в напрямку найшвидшого зменшення значення функції. Проте ефективність такого підходу значною мірою залежить від структури цільової функції та вибору початкової точки.

Головним недоліком градієнтних методів є їх схильність до потрапляння у локальні мінімуми. Для багатоекстремальних функцій це є критичною проблемою, оскільки алгоритм може завершити роботу в локальному екстремумі, не досягнувши глобального оптимального розв’язку.

Крім проблеми локальних мінімумів, градієнтні методи мають й інші обмеження:

1) необхідність обчислення похідних;
2) чутливість до вибору початкової точки;
3) залежність від параметра швидкості навчання;
4) низька ефективність у високорозмірних задачах;
5) складність роботи з недиференційованими функціями.

У випадку складних нелінійних функцій обчислення градієнта може бути ресурсоємним або навіть неможливим. Також у багатьох практичних задачах цільова функція може бути задана неаналітично, наприклад у вигляді результатів комп’ютерного моделювання або експериментальних даних, що унеможливлює застосування класичних градієнтних методів.

Суттєвою проблемою є також залежність результатів оптимізації від вибору початкових параметрів. Для мультимодальних функцій різні початкові

точки можуть приводити алгоритм до різних локальних мінімумів. Це знижує стабільність та надійність роботи традиційних методів оптимізації.

Метод Ньютона та його модифікації дозволяють прискорити процес збіжності за рахунок використання другої похідної функції або матриці Гессе.

Ітераційна формула методу Ньютона має вигляд:

$$x_{k+1} = x_k - H^{-1}(x_k)\nabla f(x_k),$$

де $x_k -$ поточний вектор параметрів;

$\nabla f(x_k) -$ градієнт функції;

$H^{-1}(x_k) -$ обернена матриця Гессе.

Матриця Гессе визначається як матриця других частинних похідних функції:

$$H(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \, \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \, \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Однак для складних багатовимірних задач обчислення матриці других похідних вимагає значних обчислювальних ресурсів.

У багатьох випадках традиційні методи демонструють ефективну роботу лише для унімодальних функцій, які мають єдиний глобальний екстремум. Для багатоекстремальних задач їх ефективність суттєво знижується через обмежену здатність досліджувати глобальний простір пошуку.

На відміну від детермінованих методів, метаевристичні алгоритми використовують стохастичні механізми пошуку, що дозволяє частково уникати передчасної збіжності та підвищує ймовірність знаходження глобального оптимуму. Саме тому останніми роками значного поширення набули еволюційні алгоритми, методи колективного інтелекту та інші метаевристичні підходи.

1.4. Метаевристичні методи глобальної оптимізації

Метаевристичні методи оптимізації – це клас наближених алгоритмів, які застосовуються для пошуку якісних розв’язків складних оптимізаційних задач [5]. На відміну від класичних детермінованих методів, метаевристики не потребують повної інформації про структуру цільової функції та зазвичай не використовують похідні. Основою їх роботи є поєднання випадкового пошуку, евристичних правил та механізмів поступового покращення розв’язків.

Головна перевага метаевристичних методів полягає у здатності досліджувати складний простір пошуку та зменшувати ризик передчасного потрапляння в локальний мінімум. Такі методи особливо корисні для задач, у яких цільова функція має багато локальних екстремумів, є нелінійною, недиференційованою або заданою у вигляді результату моделювання.

У загальному випадку метаевристичний алгоритм виконує ітераційний пошук, у процесі якого формується один або декілька кандидатів на розв’язок. Для кожного кандидата обчислюється значення цільової функції, після чого алгоритм змінює поточні розв’язки відповідно до власної стратегії пошуку. Основною метою є поступове наближення до глобального оптимуму або достатньо якісного наближеного розв’язку.

До найбільш поширених метаевристичних методів належать [6]:

1) генетичні алгоритми;
2) метод імітації відпалу;
3) алгоритм рою частинок;
4) мурашині алгоритми;
5) диференціальна еволюція;
6) табу-пошук;
7) метод спірального пошуку.

Метаевристичні алгоритми можна умовно поділити на дві великі групи: одиночні та популяційні. Одиночні методи працюють з одним поточним розв’язком і поступово його покращують. До таких методів можна віднести, наприклад, метод імітації відпалу або табу-пошук. Популяційні методи одночасно працюють із множиною кандидатів на розв’язок. Саме до цієї групи належать генетичні алгоритми, алгоритми рою частинок та метод спірального пошуку.

Популяційний характер пошуку є важливою перевагою для багатоекстремальних задач. Якщо алгоритм одночасно досліджує декілька областей простору пошуку, зростає ймовірність того, що хоча б частина кандидатів потрапить у перспективну область, близьку до глобального мінімуму. Це дозволяє ефективніше працювати зі складними функціями, які мають багато локальних екстремумів.

У даній роботі основну увагу приділено двом метаевристичним підходам: генетичному алгоритму та методу спірального пошуку.

Генетичний алгоритм належить до класу еволюційних методів оптимізації. Він базується на ідеях природного відбору, спадковості, схрещування та мутації. У межах такого підходу кожен можливий розв’язок подається у вигляді особини або хромосоми, а сукупність розв’язків утворює популяцію. На кожному поколінні виконуються оцінювання якості особин, відбір батьків, кросовер і мутація. Завдяки цьому популяція поступово зміщується в напрямку кращих розв’язків.

Метод спірального пошуку також працює з множиною точок-кандидатів, однак використовує іншу ідею. Його основою є геометрична модель спірального руху точок навколо поточного найкращого розв’язку. Точки одночасно обертаються навколо центру пошуку та поступово наближаються до нього. Такий підхід дозволяє поєднати глобальне дослідження простору з локальним уточненням знайденого розв’язку.

Метаевристичні методи не гарантують знаходження точного глобального оптимуму для будь-якої задачі. Проте на практиці вони часто дозволяють отримати достатньо якісні результати за прийнятний час. Їх основна перевага полягає у гнучкості, простоті застосування до різних типів задач і здатності працювати зі складними цільовими функціями без необхідності обчислення похідних.

Саме тому у даній роботі генетичний алгоритм і метод спірального пошуку обрано для дослідження ефективності оптимізації багатоекстремальних функцій. Їх порівняння дозволяє оцінити переваги та недоліки різних стратегій метаевристичного пошуку.

1.5. Застосування метаевристичних методів у прикладних задачах

Метаевристичні методи широко застосовуються не лише для оптимізації тестових математичних функцій, а й для розв’язання прикладних задач. Це пояснюється тим, що багато реальних задач мають складну структуру, велику кількість параметрів, нелінійні залежності та значну кількість локально оптимальних розв’язків.

До типових прикладних задач, у яких використовуються метаевристичні алгоритми, належать:

1) задачі маршрутизації транспорту;
2) задачі розміщення об’єктів;
3) задачі планування виробництва;
4) задачі оптимального розподілу ресурсів;
5) задачі оптимізації параметрів технічних систем;
6) задачі машинного навчання;
7) задачі енергетики;

8) економічні та фінансові задачі.

У задачах маршрутизації метаевристичні методи використовуються для пошуку оптимальних або близьких до оптимальних маршрутів доставки. Такі задачі часто мають комбінаторний характер, а кількість можливих маршрутів дуже швидко зростає зі збільшенням кількості пунктів доставки. Через це повний перебір усіх варіантів є практично неможливим.

У задачах планування виробництва метаевристики застосовуються для визначення порядку виконання операцій, розподілу ресурсів, мінімізації часу простою обладнання та зменшення загальних витрат. У таких задачах можуть бути присутні численні обмеження, що ускладнює застосування класичних методів оптимізації.

В інженерії метаевристичні алгоритми використовуються для підбору параметрів технічних систем, оптимізації конструкцій, налаштування регуляторів і розв’язання задач автоматизованого проєктування. Часто цільова функція в таких задачах обчислюється за результатами комп’ютерного моделювання, тому її похідні можуть бути недоступними.

У машинному навчанні метаевристики можуть застосовуватися для підбору гіперпараметрів моделей, вибору ознак або оптимізації складних функцій помилки. Хоча для навчання багатьох моделей використовуються градієнтні методи, метаевристичні алгоритми можуть бути корисними у випадках, коли простір параметрів є складним, дискретним або змішаним.

Особливо важливою сферою застосування метаевристичних методів є логістика. У логістичних задачах необхідно приймати рішення щодо розміщення складів, вибору маршрутів, розподілу товарних потоків і мінімізації транспортних витрат. Такі задачі часто мають велику кількість можливих рішень і характеризуються складною структурою цільової функції.

Задачі розміщення об’єктів є одним із класичних прикладів прикладної оптимізації [7]. Вони полягають у виборі найкращих місць для розташування певних об’єктів: складів, виробничих центрів, лікарень, пожежних станцій,

базових станцій мобільного зв’язку або сервісних пунктів. Залежно від постановки, метою може бути мінімізація транспортних витрат, мінімізація середньої відстані до клієнтів, максимізація покриття території або мінімізація часу реагування.

У таких задачах часто існує декілька локально оптимальних конфігурацій. Наприклад, склади можуть бути розміщені так, що добре обслуговують одну групу міст, але не забезпечують мінімальних витрат для всієї системи загалом. Інша конфігурація може бути кращою з точки зору глобального критерію, але знайти її складно через велику кількість можливих варіантів.

Метаевристичні методи добре підходять для таких задач, оскільки вони не потребують аналітичного вигляду похідних і можуть працювати лише зі значенням цільової функції. Алгоритм може розглядати задачу як «чорну скриньку»: для кожного набору параметрів обчислюється значення функції, після чого метод приймає рішення про подальший напрям пошуку.

У межах даної роботи як приклад прикладної задачі розглядається задача розміщення складів. Вона дозволяє перейти від аналізу штучних тестових функцій до практичної задачі логістичного типу. При цьому цільова функція задачі розміщення складів також може бути розглянута як багатоекстремальна, оскільки різні конфігурації складів можуть утворювати різні локально оптимальні рішення.

Таким чином, застосування метаевристичних методів у прикладних задачах є доцільним тоді, коли класичні методи не забезпечують достатньої надійності або коли цільова функція має складну структуру. Саме такі умови характерні для задач глобальної оптимізації багатоекстремальних функцій, що розглядаються в даній роботі.

1.6. Задача розміщення складів як приклад прикладної багатоекстремальної оптимізації

Задача розміщення складів є однією з класичних задач логістики та дослідження операцій [8]. Її сутність полягає у визначенні такого розташування складів, яке забезпечує мінімальні витрати на обслуговування клієнтів або населених пунктів. У загальному випадку задача може включати вибір кількості складів, визначення їх місць розташування, розподіл клієнтів між складами та врахування витрат на відкриття або утримання складів.

У даній роботі розглядається спрощена неперервна постановка задачі. Кількість складів вважається фіксованою, а координати складів є змінними, які необхідно знайти в процесі оптимізації. Міста та їх попит задаються наперед. Кожне місто обслуговується найближчим складом, а цільова функція визначає сумарні транспортні витрати.

Саме тому для задачі розміщення складів доцільно використовувати метаевристичні методи. Генетичний алгоритм дозволяє досліджувати різні області простору пошуку завдяки роботі з популяцією розв’язків, а метод спірального пошуку забезпечує поступове уточнення розв’язку шляхом переміщення точок навколо поточного найкращого кандидата.

У межах даної роботи задача розміщення складів використовується як приклад практичної багатоекстремальної задачі, що доповнює дослідження на класичних тестових функціях. Це дозволяє оцінити ефективність генетичного алгоритму та методу спірального пошуку не лише на штучних математичних функціях, а й на прикладній задачі логістичного характеру.

1.7. Висновки до розділу 1

У першому розділі було розглянуто теоретичні основи задач глобальної оптимізації багатоекстремальних функцій. Було сформульовано загальну постановку задачі мінімізації цільової функції та наведено визначення глобального і локального мінімуму.

Показано, що багатоекстремальні функції характеризуються наявністю декількох локальних екстремумів, що ускладнює пошук глобального оптимуму. Основною проблемою таких задач є ризик передчасної збіжності алгоритму до локального мінімуму.

Було проаналізовано обмеження традиційних детермінованих та градієнтних методів оптимізації. Встановлено, що такі методи суттєво залежать від початкової точки, потребують обчислення похідних і часто демонструють недостатню ефективність при роботі зі складними багатоекстремальними функціями.

Розглянуто загальні особливості метаевристичних методів глобальної оптимізації. Показано, що вони не потребують інформації про похідні цільової функції та можуть застосовуватися до складних нелінійних, недиференційованих і багатовимірних задач.

Окрему увагу приділено прикладним задачам оптимізації, зокрема задачі розміщення складів. Було побудовано математичну модель цієї задачі, у якій вектор змінних містить координати складів, а цільова функція визначає сумарні транспортні витрати з урахуванням попиту міст.

Показано, що задача розміщення складів може бути розглянута як приклад прикладної багатоекстремальної оптимізації, оскільки зміна координат складів може змінювати розподіл міст між найближчими складами та формувати декілька локально оптимальних конфігурацій.

Таким чином, проведений теоретичний аналіз підтверджує доцільність використання метаевристичних методів, зокрема генетичного алгоритму та методу спірального пошуку, для дослідження задач глобальної оптимізації багатоекстремальних функцій як тестового, так і прикладного характеру.

РОЗДІЛ 2. МАТЕМАТИЧНІ МОДЕЛІ ТА МЕТОДИ ОПТИМІЗАЦІЇ

2.1. Концептуальні основи генетичного алгоритму

Генетичний алгоритм є одним із найбільш відомих та поширених метаевристичних методів глобальної оптимізації [9]. Даний підхід належить до класу еволюційних алгоритмів та базується на моделюванні природних процесів еволюції живих організмів, зокрема механізмів спадковості, природного відбору, схрещування та мутації [10].

Основна ідея генетичного алгоритму полягає у формуванні популяції можливих розв'язків задачі та її поступовому вдосконаленні протягом багатьох поколінь. У процесі еволюції більш пристосовані особини мають вищу ймовірність передати свої характеристики наступному поколінню, що забезпечує поступове покращення якості розв'язків.

Генетичний алгоритм належить до класу стохастичних методів оптимізації, оскільки використовує випадкові механізми генерації та модифікації розв'язків. На відміну від класичних градієнтних методів, генетичний алгоритм не потребує обчислення похідних функції та може застосовуватися до складних нелінійних, недиференційованих або багатоекстремальних функцій.

У межах генетичного алгоритму кожен можливий розв'язок задачі подається у вигляді окремої особини або хромосоми. Сукупність усіх особин утворює популяцію. Кожна хромосома містить набір параметрів задачі оптимізації:

$$x = (x_1, x_2, \ldots, x_n),$$

де x_i – окремі параметри розв'язку;

$n -$ кількість змінних задачі.

У задачах оптимізації кожна особина оцінюється за допомогою функції пристосованості (fitness function), яка визначає якість знайденого розв'язку. Оскільки у роботі розв'язується задача мінімізації, кращими вважаються особини з меншим значенням функції:

$$Fitness(x) = f(x),$$

де $f(x) -$ значення цільової функції;

$x -$ поточний розв'язок.

У загальному вигляді генетичний алгоритм складається з таких етапів.

1. Формування початкової популяції.
2. Обчислення функції пристосованості.
3. Відбір найкращих особин.
4. Схрещування (кросовер).
5. Мутація.
6. Формування нового покоління.
7. Перевірка критерію завершення алгоритму.

Основними параметрами генетичного алгоритму є:

1) розмір популяції P;
2) коефіцієнт відбору φ_{sel};
3) коефіцієнт кросоверу φ_{cross};
4) коефіцієнт мутації φ_{mut}.

Кількість особин, отриманих кожним способом, визначається формулами:

$$n_{sel} = \varphi_{sel} P$$
$$n_{cross} = \varphi_{cross} P$$
$$n_{mut} = \varphi_{mut} P$$

Сумарна кількість особин нового покоління повинна дорівнювати розміру популяції:

$$P = n_{sel} + n_{cross} + n_{mut}$$

2.1.1. Формування початкової популяції

Початкова популяція є стартовим набором рішень, з якого розпочинається робота генетичного алгоритму. Від способу її формування значною мірою залежить швидкість збіжності алгоритму та якість отриманого результату.

Найбільш поширеним методом є випадкова генерація особин. Координати індивідів формуються в межах області пошуку за формулою:

$$x_{i,d} = lb_d + r_{i,d}(ub_d - lb_d),$$

де $lb_d -$ нижня межа області пошуку;

$ub_d -$ верхня межа області пошуку;

$r_{i,d} \in [0; 1] -$ випадкове число;

$i -$ номер особини;

$d -$ номер координати.

Для більш рівномірного покриття простору пошуку можуть використовуватись квазівипадкові послідовності, зокрема послідовність Холтона. Вона належить до послідовностей із низькою розбіжністю та дозволяє уникати надмірного скупчення точок у певних областях. Для багатовимірного простору використовуються взаємно прості основи генерації послідовності.

Також можуть застосовуватись:

1) регулярні сітки;
2) послідовності Соболя;
3) послідовності Хаммерслі;

4) комбіновані способи генерації.

2.1.2. Оператори селекції

Селекція призначена для відбору найбільш пристосованих особин, які братимуть участь у формуванні нового покоління [11]. Основна мета селекції полягає у підвищенні середньої якості популяції.

Відбір еліти. При елітному відборі до нового покоління без змін переходять особини з найкращими значеннями функції пристосованості.

Рулеточний відбір. Ймовірність вибору особини пропорційна її пристосованості:

$$\hat{f}_i = \frac{f_i^*}{\sum_{i=1}^{P} f_i^*},$$

де $f_i^* -$ значення функції пристосованості;

$\hat{f}_i -$ нормалізована ймовірність вибору.

Для задач мінімізації використовується перетворення:

$$f_i^* = \frac{f_{max} - f_i}{f_{max} - f_{min}},$$

де: $f_{max} -$ найбільше значення функції;

$f_{min} -$ найменше значення функції.

Ранговий відбір. Індивіди ранжуються відповідно до якості розв’язку, а ймовірність вибору залежить від рангу особини, а не від абсолютного значення функції пристосованості.

Турнірний відбір. Із випадково обраної групи особин вибирається найкраща. Розмір турніру є параметром алгоритму та впливає на інтенсивність селекції.

2.1.3. Способи кросоверу

Кросовер є оператором комбінування генетичної інформації батьківських особин для формування нових рішень.

Кросовер у двійковому кодуванні.

Для двійкового кодування генеруються точки розриву:

$$a_1, a_2, \dots, a_n$$

Після сортування батьківські хромосоми обмінюються відповідними фрагментами бітових рядків.

Рівномірний кросовер у природному кодуванні.

Дочірня особина формується за формулою:

$$z_i = \beta_i x_i + (1 - \beta_i) y_i,$$

де $x_i, y_i -$ координати батьківських особин;

$\beta_i \in [-d; 1 + d] -$ випадковий коефіцієнт;

$d -$ параметр розширення області пошуку.

Нерівномірний кросовер.

Дочірня особина формується за формулою: $z_i = x_i + \rho_i(y_i - x_i)$, де

$$\rho_i = \begin{cases} -d + (1 + 2d)\beta_i^s, & f(X) < f(Y), \\ -d + (1 + 2d)\beta_i, & f(X) = f(Y), \\ -d + (1 + 2d)\beta_i^{1/s}, & f(X) > f(Y). \end{cases}$$

Параметр $s > 1$ регулює ступінь наближення дочірньої особини до кращого з батьків.

2.1.4. Способи мутації

Мутація використовується для підтримання різноманітності популяції та запобігання передчасній збіжності алгоритму до локального екстремуму.

Мутація у двійковому кодуванні.

Для кожного біта хромосоми випадково перевіряється умова мутації. Якщо випадково згенероване число менше за задану ймовірність мутації, відповідний біт інвертується. Тобто значення біта змінюється на протилежне.

Мутація у природному кодуванні.

До координат особини додається нормально розподілений випадковий вектор:

$$Y = X + N(0, \sigma),$$

де $X -$ батьківська особина;

$N(0, \sigma) -$ нормальний розподіл;

$\sigma -$ параметр мутації.

Значення σ визначає силу мутації та інтенсивність дослідження області пошуку.

2.1.5. Критерії завершення алгоритму

Критерій завершення визначає момент припинення роботи генетичного алгоритму.

Найчастіше використовуються такі критерії.

1. Досягнення фіксованої кількості поколінь.
2. Відсутність покращення найкращого розв’язку протягом G поколінь.

3. Виродження популяції, коли більшість особин стають подібними.
4. Досягнення заданої кількості обчислень цільової функції.
5. Перевищення допустимого часу роботи алгоритму.

У даній роботі основним критерієм завершення є досягнення фіксованої кількості поколінь.

Також для порівняння алгоритмів використовується кількість обчислень цільової функції. Для генетичного алгоритму вона приблизно дорівнює:

$$N^{GA} = n_{pop} \cdot n_{iter}.$$

Якщо враховувати початкове обчислення найкращого розв'язку, тоді:

$$N^{GA} = 1 + n_{pop} \cdot n_{iter},$$

де N^{GA} – кількість обчислень цільової функції генетичним алгоритмом;

n_{pop} – розмір популяції; n_{iter} – кількість поколінь.

2.2. Геометрична концепція та математична модель Методу спірального пошуку

Метод спірального пошуку (Spiral Optimization Algorithm, SPO) належить до класу метаевристичних алгоритмів глобальної оптимізації та базується на геометричній концепції спірального руху в просторі пошуку. Основною ідеєю методу є поступове наближення множини точок-кандидатів до оптимального розв'язку шляхом виконання обертального руху зі зменшенням радіуса пошуку [12].

На відміну від еволюційних алгоритмів, метод спірального пошуку не використовує операторів селекції, схрещування або мутації. Замість цього

пошук виконується за допомогою геометричних перетворень, які дозволяють поєднати глобальне дослідження простору пошуку та локальне уточнення розв'язку.

Основою алгоритму є поняття спіральної траєкторії. У процесі оптимізації точки-кандидати переміщуються навколо поточного найкращого розв'язку за спіральною траєкторією, поступово зменшуючи відстань до центру пошуку. Такий підхід дозволяє алгоритму ефективно досліджувати область пошуку та уникати передчасного потрапляння до локального мінімуму.

Нехай: $x_i(k) -$ положення i-ї точки на ітерації k; $x^*(k) -$ поточний найкращий розв'язок; $r -$ коефіцієнт наближення; $R(\theta) -$ матриця повороту; $\theta -$ кут повороту.

Тоді загальна формула оновлення положення точки має вигляд:

$$x_i(k+1) = x^*(k) + rR(\theta)(x_i(k) - x^*(k))$$

Дана формула описує одночасно два процеси.

1. Обертання точки навколо центру пошуку.
2. Поступове наближення точки до найкращого розв'язку.

У двовимірному випадку матриця повороту задається виразом:

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Для багатовимірного випадку матриця повороту може бути побудована як блочно-діагональна матриця, що складається з двовимірних блоків повороту $R(\theta)$. Якщо розмірність простору дорівнює $n = 2k$, то така матриця має вигляд:

$$R_n(\theta) = \begin{bmatrix} R(\theta) & 0 & \cdots & 0 \\ 0 & R(\theta) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R(\theta) \end{bmatrix}$$

У такому випадку кожен блок $R(\theta)$ виконує поворот відповідної пари координат. Наприклад, у задачі розміщення k складів розмірність простору

становить $n = 2k$, оскільки кожен склад задається двома координатами (x_j, y_j). Тому матриця повороту містить k двовимірних блоків.

Матриця повороту забезпечує циклічний рух точок у просторі пошуку, тоді як коефіцієнт r визначає швидкість звуження області пошуку.

Однією з основних особливостей методу спірального пошуку є використання множини точок-кандидатів, які незалежно досліджують простір пошуку. Після кожної ітерації визначається найкраща точка:

$$x^*(k) = \arg\min_{x_i(k)} f(x_i(k))$$

де $f(x_i(k))$ – значення цільової функції для i-ї точки.

Найкращий розв'язок використовується як новий центр спірального руху для наступної ітерації алгоритму.

Початкова множина точок генерується випадковим чином у межах області пошуку:

$$x_i \in [lb, ub],$$

де lb – нижня межа області пошуку; ub – верхня межа області пошуку.

Основними параметрами методу спірального пошуку є:

1) кількість точок-кандидатів;
2) коефіцієнт стискання r;
3) кут повороту θ;
4) максимальна кількість ітерацій.

Вибір параметра r суттєво впливає на характер роботи алгоритму. При великих значеннях r алгоритм виконує ширше дослідження простору пошуку, тоді як менші значення забезпечують швидше локальне уточнення розв'язку.

Кут повороту θ визначає форму спіральної траєкторії та впливає на рівномірність дослідження простору пошуку. Невдалий вибір параметрів може призвести до надмірного локального пошуку або нестабільності алгоритму.

До основних переваг методу спірального пошуку належать:

1) простота математичної моделі;
2) невелика кількість параметрів;
3) ефективне поєднання глобального та локального пошуку;
4) відсутність необхідності обчислення похідних;
5) можливість роботи зі складними багатоекстремальними функціями.

Водночас метод має певні недоліки:

1) чутливість до параметрів r та θ;
2) можливість передчасної збіжності;
3) зниження ефективності у високорозмірних задачах.

Метод спірального пошуку демонструє високу ефективність при оптимізації неперервних мультимодальних функцій та добре підходить для задач, у яких необхідне одночасне поєднання глобального дослідження простору пошуку та локального уточнення розв’язку.

Кількість обчислень цільової функції для методу спірального пошуку приблизно дорівнює:

$$N^{SP} = n_{points} \cdot (n_{iter} + 1).$$

2.3. Характеристика та математичне формулювання тестових багатоекстремальних функцій

Для оцінювання ефективності алгоритмів глобальної оптимізації використовуються спеціальні тестові функції, які дозволяють досліджувати поведінку алгоритмів у складних умовах багатовимірного простору пошуку. Такі функції називаються тестовими багатоекстремальними функціями.

Основною особливістю багатоекстремальних функцій є наявність великої кількості локальних мінімумів та максимумів. Це суттєво ускладнює

процес пошуку глобального оптимуму, оскільки алгоритм може передчасно збігатися до локального екстремуму. Використання тестових функцій дозволяє оцінити:

1) здатність алгоритму уникати локальних мінімумів;
2) швидкість збіжності;
3) стабільність результатів;
4) ефективність глобального пошуку.

У даній роботі для дослідження генетичного алгоритму та методу спірального пошуку використовуються функція Швефеля та функція Міхалевича.

2.3.1 Функція Швефеля

Функція Швефеля (Schwefel Function) є однією з найбільш складних тестових багатоекстремальних функцій для задач глобальної оптимізації. Її особливістю є велика кількість локальних мінімумів та значна віддаленість глобального мінімуму від початку координат (рис 2.1).

Математичне формулювання функції Швефеля має вигляд:

$$f(x) = 418.9829n - \sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|})$$

Для двовимірного випадку функція записується як:

$$f(x, y) = 837.9658 - x \sin\left(\sqrt{|x|}\right) - y \sin(\sqrt{|y|})$$

Область пошуку для функції: $x, y \in [-500; 500]$

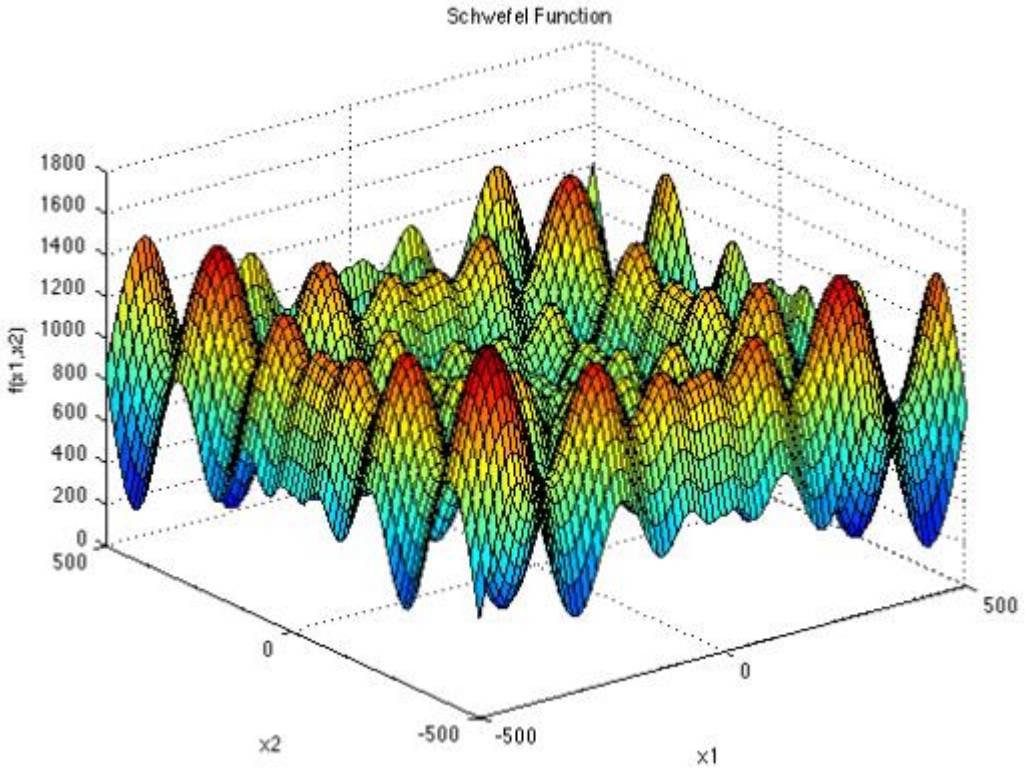


Рисунок 2.1 Функція Швефеля

Глобальний мінімум функції досягається у точці [13]:

$$x_i = 420.9687$$

Для двовимірного випадку:

$$f(420.9687,420.9687) = 0$$

Поверхня функції Швефеля характеризується великою кількістю локальних екстремумів та складним хвилеподібним рельєфом. Значна кількість локальних мінімумів створює складні умови для алгоритмів оптимізації та дозволяє оцінити їх здатність виконувати глобальний пошук.

Особливістю функції є те, що глобальний мінімум знаходиться поблизу меж області пошуку, що додатково ускладнює процес оптимізації.

2.3.2 Функція Міхалевича

Функція Міхалевича (Michalewicz Function) є складною багатоекстремальною тестовою функцією, яка використовується для перевірки здатності алгоритмів працювати в умовах великої кількості вузьких локальних мінімумів (рис 2.2).

Математичне формулювання функції має вигляд:

$$f(x) = -\sum_{i=1}^{n} \sin(x_i) \sin^{2m}\left(\frac{i{x_i}^2}{\pi}\right),$$

де $m -$ параметр, який визначає складність функції; найчастіше використовується значення: $m = 10$.

Для двовимірного випадку функція записується як:

$$f(x, y) = -\sin(x) \sin^{20}\left(\frac{x^2}{\pi}\right) - \sin(y) \sin^{20}\left(\frac{2y^2}{\pi}\right)$$

Область пошуку: $x, y \in [0; \pi]$.

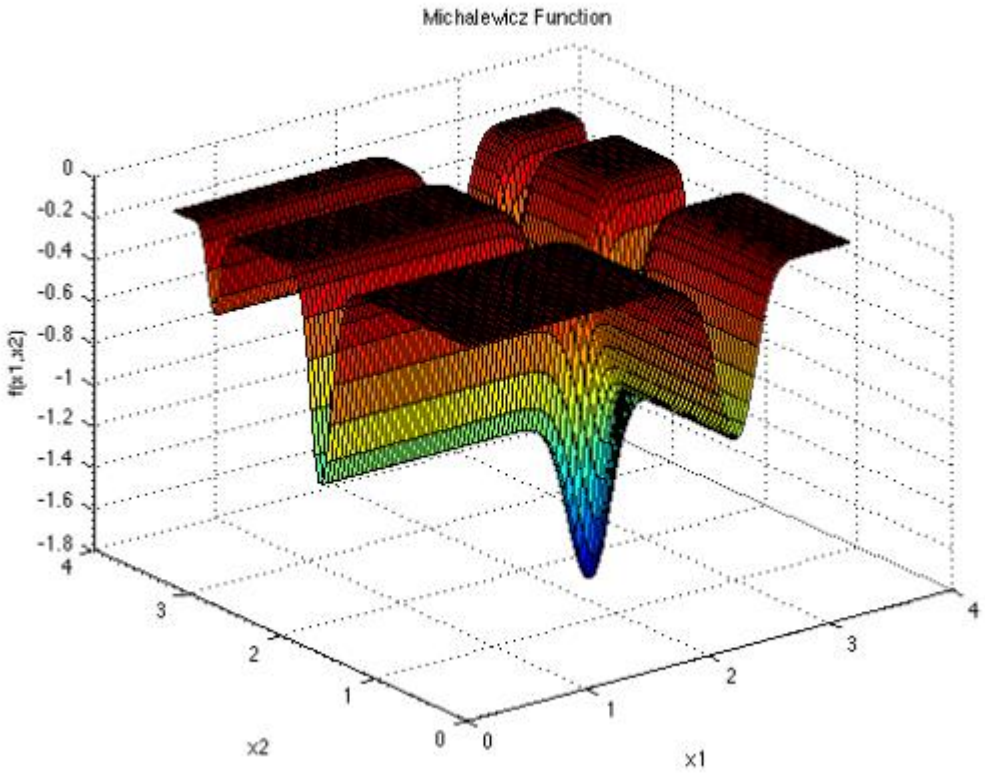


Рисунок 2.2 Функція Міхалевича

Функція Міхалевича має велику кількість вузьких локальних мінімумів, що значно ускладнює процес знаходження глобального оптимуму. Поверхня функції характеризується складною структурою та високою чутливістю до координат точок.

Для двовимірного випадку глобальний мінімум приблизно знаходиться у точці: $f(2.20,1.57) \approx -1.8013$.

Функція Міхалевича є особливо складною для алгоритмів оптимізації через вузькі області глобального мінімуму та велику кількість локальних екстремумів. Вона дозволяє оцінити здатність алгоритму виконувати точний локальний пошук та уникати передчасної збіжності.

2.3.3. Порівняльна характеристика тестових функцій

Функція Швефеля та функція Міхалевича мають різні властивості та створюють різні типи складності для алгоритмів глобальної оптимізації.

Функція Швефеля характеризується:

1) великою областю пошуку;
2) значною кількістю локальних мінімумів;
3) глобальним мінімумом поблизу меж області;
4) складним хвилеподібним рельєфом поверхні.

Функція Міхалевича характеризується:

1) вузькими локальними мінімумами;
2) високою чутливістю до координат;
3) складним рельєфом поверхні;
4) меншою областю пошуку порівняно з функцією Швефеля.

Використання двох різних типів тестових функцій дозволяє більш об’єктивно оцінити ефективність генетичного алгоритму та методу спірального пошуку при розв’язанні задач глобальної оптимізації.

Функція Швефеля краще демонструє здатність алгоритму виконувати глобальний пошук у великій області, тоді як функція Міхалевича дозволяє оцінити точність локального уточнення в складній поверхні з вузькими областями мінімумів.

2.4. Математична постановка задачі розміщення складів

Окрім класичних тестових функцій, у даній роботі розглядається прикладна задача розміщення складів. Вона використовується як приклад багатоекстремальної цільової функції, що виникає у логістиці.

Задача полягає у виборі координат складів таким чином, щоб мінімізувати сумарні транспортні витрати для обслуговування заданої множини міст. Кожне місто має координати та попит. Кожен склад також має координати, які потрібно визначити в процесі оптимізації.

На відміну від тестових функцій, задача розміщення складів має практичний зміст. Вона може використовуватися для моделювання задач логістики, планування складської інфраструктури, розміщення сервісних центрів або інших об’єктів обслуговування.

У даній роботі розглядається спрощена неперервна постановка задачі. Кількість складів вважається фіксованою, а координати складів є змінними. Міста та значення попиту задаються наперед. Кожне місто обслуговується найближчим складом.

2.4.1. Опис вхідних даних задачі

Нехай задано N міст. Координати $i-$ го міста позначимо як: $c_i = (a_i, b_i)$.

Попит міста позначимо як: D_i.

Попит показує умовний обсяг товару, який необхідно доставити до відповідного міста. Чим більший попит міста, тим більший вплив має його відстань до складу на значення цільової функції.

Також задається кількість складів: k.

У даній постановці кількість складів є фіксованою. Це означає, що алгоритм не обирає, скільки складів відкривати, а шукає найкраще розташування вже заданої кількості складів.

Вхідними даними задачі є:

1) кількість міст N;
2) координати міст $c_i = (a_i, b_i)$;
3) попит кожного міста D_i;
4) кількість складів k;
5) область допустимих координат складів.

У програмній реалізації область пошуку задається у вигляді квадратної області:

$$[0{,}100] \times [0{,}100].$$

Це означає, що кожна координата складу повинна належати проміжку від 0 до 100.

2.4.2. Формування вектора параметрів

Кожен склад має дві координати. Координати $j-$ го складу позначимо так: $w_j = (x_j, y_j)$.

Тоді вектор параметрів задачі має вигляд:

$$X = (x_1, y_1, x_2, y_2, \dots, x_k, y_k).$$

Розмірність задачі дорівнює: $2k$.

Це дозволяє перейти від двовимірних тестових функцій до багатовимірної прикладної задачі. Таким чином, задача розміщення складів є зручною для дослідження поведінки метаевристичних алгоритмів у просторах різної розмірності.

2.4.3. Побудова цільової функції сумарних транспортних витрат

Відстань між містом $c_i = (a_i, b_i)$ та складом $w_j = (x_j, y_j)$ визначається за евклідовою метрикою:

$$d(c_i, w_j) = \sqrt{(a_i - x_j)^2 + (b_i - y_j)^2}.$$

Оскільки кожне місто обслуговується найближчим складом, для міста iобирається мінімальна відстань до одного зі складів:

$$d_i(X) = \min_{j=1,\dots,k} d(c_i, w_j).$$

Тоді цільова функція має вигляд:

$$F(X) = \sum_{i=1}^{N} D_i \, d_i(X).$$

У повному вигляді:

$$F(X) = \sum_{i=1}^{N} D_i \cdot \min_{j=1,\dots,k} \sqrt{(a_i - x_j)^2 + (b_i - y_j)^2}.$$

Задача оптимізації формулюється так:

$$\min_{X \in \Omega} F(X),$$

де $\Omega -$ область допустимих значень координат складів.

Якщо координати складів обмежені квадратною областю:

$$[0,100] \times [0,100],$$

то для k складів область пошуку має вигляд:

$$\Omega = [0,100]^{2k}.$$

Таким чином, алгоритм повинен знайти такі координати складів, щоб сумарна зважена відстань від міст до найближчих складів була мінімальною.

Попит D_i у цій моделі виконує роль вагового коефіцієнта. Якщо місто має великий попит, його віддаленість від складу сильніше впливає на значення цільової функції. Тому алгоритм прагне розміщувати склади ближче до міст або груп міст із більшим попитом.

У деяких постановках задачі також враховується вартість відкриття складів. У такому випадку цільова функція може бути записана так:

$$F(X) = C_{open} \cdot k + \sum_{i=1}^{N} D_i \, d_i(X),$$

де C_{open} – вартість відкриття одного складу.

Однак у даній роботі кількість складів є фіксованою. Тому величина:

$$C_{open} \cdot k$$

є сталою і не впливає на координати мінімуму. Через це в експериментальній частині використовується спрощена цільова функція сумарних транспортних витрат без додаткового члена вартості відкриття складів.

2.4.4. Обґрунтування багатоекстремальності цільової функції задачі розміщення складів

Цільова функція задачі розміщення складів має складну структуру через наявність оператора мінімуму:

$$\min_{j=1,\dots,k}.$$

Для кожного міста обирається найближчий склад. Якщо координати складів змінюються, то може змінитися й те, який склад є найближчим для конкретного міста. У результаті змінюється розподіл міст між складами.

Для одного міста відстань до найближчого складу можна записати так:

$$d_i(X) = \min\{d(c_i, w_1), d(c_i, w_2), \dots, d(c_i, w_k)\}.$$

Це означає, що простір пошуку поділяється на області, які відповідають різним варіантам закріплення міст за складами. У кожній області цільова

функція має свою структуру. Різні конфігурації складів можуть бути локально оптимальними, але не всі вони забезпечують глобально мінімальні транспортні витрати.

Наприклад, певне розташування складів може добре обслуговувати одну групу міст, але бути неефективним для іншої. Інша конфігурація може мати трохи гірше локальне розташування для окремих міст, але забезпечувати менші сумарні транспортні витрати загалом.

Саме тому задача розміщення складів може розглядатися як приклад прикладної багатоекстремальної оптимізації. Вона є особливо корисною для даної роботи, оскільки дозволяє перевірити генетичний алгоритм і метод спірального пошуку не лише на штучних тестових функціях, а й на задачі з практичним змістом.

2.5. Критерії порівняння ефективності алгоритмів

Для порівняння ефективності генетичного алгоритму та методу спірального пошуку необхідно визначити критерії оцінювання. У даній роботі використовуються такі критерії:

1) найкраще знайдене значення цільової функції;
2) середнє значення результату за серією запусків;
3) найгірше значення результату;
4) час роботи;
5) успішність знаходження розв’язку із заданою точністю.

Найкраще значення функції показує, наскільки якісний розв’язок зміг знайти алгоритм у найуспішнішому запуску. Середнє значення характеризує типову поведінку алгоритму. Найгірше значення дозволяє оцінити ризик отримання неякісного розв’язку.

Для тестових функцій, де відомий глобальний мінімум, можна також оцінювати абсолютну або відносну похибку. Абсолютна похибка визначається як:

$$\Delta = | f - f^* |,$$

де $f -$ знайдене значення функції;

$f^* -$ теоретичне значення глобального мінімуму.

Відносна похибка може бути визначена як:

$$\delta = \frac{| f - f^* |}{| f^* |} \cdot 100\%.$$

Для функції Швефеля, мінімум якої дорівнює нулю, доцільно використовувати абсолютну похибку. Для функції Міхалевича можна використовувати відносну похибку, оскільки глобальний мінімум має ненульове значення.

Для задачі розміщення складів точне глобальне значення функції зазвичай невідоме, тому основними критеріями є найкраще, середнє та найгірше значення цільової функції за серією запусків, а також кількість обчислень цільової функції.

2.6. Висновки до розділу 2

У другому розділі було розглянуто математичні моделі та методи оптимізації, які використовуються в даній роботі.

Було описано генетичний алгоритм як популяційний метаевристичний метод глобальної оптимізації. Розглянуто спосіб бітового кодування розв’язків, процедуру декодування, оператори селекції, кросоверу та мутації, а також критерії завершення роботи алгоритму.

Також було розглянуто математичну модель методу спірального пошуку. Показано, що цей метод базується на переміщенні множини точок навколо поточного найкращого розв’язку за допомогою матриці повороту та коефіцієнта наближення.

Описано тестові багатоекстремальні функції Швефеля та Міхалевича, які використовуються для перевірки здатності алгоритмів знаходити глобальний мінімум у складному просторі пошуку.

Окрему увагу приділено математичній постановці задачі розміщення складів. Було сформовано вектор параметрів, побудовано цільову функцію сумарних транспортних витрат і пояснено, чому така задача може розглядатися як приклад прикладної багатоекстремальної оптимізації.

Також визначено критерії порівняння ефективності алгоритмів, серед яких основними є якість знайденого розв’язку, стабільність результатів і час. Саме ці критерії будуть використані в експериментальній частині роботи.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

3.1. Опис програмної реалізації

Для проведення експериментального дослідження було розроблено програмну реалізацію методів оптимізації мовою програмування Python. У межах роботи реалізовано генетичний алгоритм, метод спірального пошуку, а також допоміжні функції для обчислення значень тестових і прикладних цільових функцій.

Програмна реалізація призначена для дослідження ефективності метаевристичних методів при розв’язанні задач глобальної оптимізації багатоекстремальних функцій. У роботі розглядаються як класичні тестові функції, так і прикладна задача розміщення складів.

Для реалізації програмного продукту використано такі засоби [14] [15]:

1) мова програмування Python;
2) бібліотека NumPy для роботи з масивами та числовими обчисленнями;
3) модуль Math для використання математичних функцій;
4) бібліотека Matplotlib для побудови графіків та візуалізації результатів.

У програмі реалізовано можливість дослідження таких цільових функцій:

1) функція Швефеля;
2) функція Міхалевича;
3) цільова функція задачі розміщення складів.

Функції Швефеля та Міхалевича використовуються як класичні тестові багатоекстремальні функції. Вони мають складний рельєф поверхні та містять локальні екстремуми, що дозволяє перевірити здатність алгоритмів уникати передчасної збіжності та знаходити глобальний мінімум.

Задача розміщення складів використовується як приклад прикладної багатоекстремальної оптимізації. У цій задачі необхідно знайти такі координати складів, за яких сумарні транспортні витрати для обслуговування множини міст будуть мінімальними.

У програмній реалізації передбачено такі основні етапи обчислень:

1) задання області пошуку;
2) формування початкових розв’язків;
3) обчислення значення цільової функції;
4) виконання ітераційного процесу оптимізації;
5) збереження найкращого знайденого розв’язку;
6) побудова графіків збіжності;
7) виведення координат знайденого розв’язку;
8) порівняння результатів роботи алгоритмів.

Для генетичного алгоритму реалізовано бітове кодування розв’язків, процедуру декодування бітових рядків у дійсні значення, оператори селекції, кросоверу та мутації. У роботі розглядаються різні способи селекції: турнірний, рулетковий та ранговий відбір.

Для методу спірального пошуку реалізовано генерацію множини початкових точок, визначення найкращої точки на кожній ітерації та оновлення координат точок за допомогою матриці повороту і коефіцієнта наближення. Для задачі розміщення складів метод спірального пошуку використовується у багатовимірному просторі, де кожна точка описує координати всіх складів.

Результати роботи алгоритмів подаються у вигляді таблиць і графіків. Таблиці використовуються для порівняння числових значень, а графіки – для

візуального аналізу збіжності алгоритмів і розташування знайдених розв’язків.

3.2. Дослідження ефективності традиційних методів оптимізації

Перед дослідженням метаевристичних алгоритмів доцільно розглянути роботу традиційних методів оптимізації. Це дозволяє показати, чому для багатоекстремальних функцій виникає потреба у використанні генетичного алгоритму та методу спірального пошуку.

У роботі розглядаються два традиційні методи:

1) градієнтний спуск;
2) метод Ньютона.

Обидва методи належать до локальних методів оптимізації. Вони можуть ефективно працювати для гладких унімодальних функцій, але для багатоекстремальних функцій їх ефективність суттєво залежить від вибору початкової точки.

Основною проблемою таких методів є те, що вони використовують локальну інформацію про поведінку функції. У випадку багатоекстремальної функції це може призвести до збіжності не до глобального мінімуму, а до одного з локальних мінімумів.

3.2.1. Дослідження роботи градієнтного спуску

Для функції Швефеля будемо позначати успіх роботи алгоритму, якщо абсолютна похибка буде менша за 1.

Таблиця 3.1 Результат градієнтного спуску для функціїї Швефеля

Стартова точка	Знайдена точка	Значення функції	Абсолютна похибка	Час роботи, с	Успіх
[0, 0]	[5.0801, 5.0801]	830.0854	830.0854	0.021566	Ні
[100, 100]	[68.936, 68.936]	713.7468	713.7468	0.021592	Ні
[-100, -100]	[-122.5975, 122.5975]	593.4936	593.4936	0.021562	Ні
[-300, 300]	[-302.3246, 273.4497]	738.9087	738.9087	0.020877	Ні
[300, -300]	[273.4497, -302.3246]	738.9087	738.9087	0.021504	Ні
[380, 380]	[417.2915, 417.2915]	3.405367	3.405367	0.021309	Ні
[400, 400]	[419.2349, 419.2349]	0.758097	0.758097	0.020803	Так
[420, 420]	[420.8913, 420.8913]	0.001539	0.001539	0.020761	Так
[430, 430]	[421.6899, 421.6899]	0.131305	0.131305	0.022055	Так
[450, 450]	[423.3516, 423.3516]	1.433853	1.433853	0.021174	Ні

Для функції Міхалевича будемо позначати успіх роботи алгоритму, якщо відносна похибка буде менша за 1%.

Таблиця 3.2 Результат градієнтного спуску для функціїї Міхалевича

Стартова точка	Знайдена точка	Значення функції	Відносна похибка (%)	Час роботи, с	Успіх
[0.2, 0.2]	[0.2, 0.2]	0	100	0.024407	Ні
[0.5, 0.5]	[0.5, 0.5]	0	100	0.032076	Ні
[1.0, 1.0]	[1.0, 1.0107]	-0.000037	99.9979	0.02749	Ні
[2.5, 1.0]	[2.2029, 1.0107]	-0.80134	55.5132	0.02342	Ні
[3.0, 3.0]	[3.0, 2.9995]	0	100	0.023032	Ні
[2.0, 1.4]	[2.2029, 1.5708]	-1.801303	0.0002	0.02396	Так
[2.1, 1.5]	[2.2029, 1.5708]	-1.801303	0.0002	0.023474	Так
[2.2, 1.57]	[2.2029, 1.5708]	-1.801303	0.0002	0.024384	Так
[2.3, 1.6]	[2.2029, 1.5708]	-1.801303	0.0002	0.023817	Так
[2.4, 1.7]	[2.2029, 1.5708]	-1.801303	0.0002	0.023761	Так

Отримані результати показують, що градієнтний спуск суттєво залежить від вибору початкової точки. Якщо початкова точка розташована поблизу глобального мінімуму, алгоритм може знайти точний або близький до точного розв’язок. Однак для стартових точок, віддалених від глобального мінімуму, метод часто збігається до локальних екстремумів.

Таким чином, градієнтний спуск не забезпечує надійного глобального пошуку для багатоекстремальних функцій. Це підтверджує доцільність використання метаевристичних методів, які здатні досліджувати різні області простору пошуку.

3.2.2. Дослідження роботи методу Ньютона

Аналогічно, для функції Швефеля будемо позначати успіх роботи алгоритму, якщо абсолютна похибка буде менша за 1.

Таблиця 3.3 Результат методу Ньютона для функціїї Швефеля

Стартова точка	Знайдена точка	Значення функції	Абсолютна похибка	Час роботи, с	Успіх
[0, 0]	[5.2392, 5.2392]	830.075197	830.075197	0.001021	Ні
[100, 100]	[65.5479, 65.5479]	710.695836	710.695836	0.001188	Ні
[-100, -100]	[-124.8294, -124.8294]	592.213453	592.213453	0.001161	Ні
[-300, 300]	[-302.5249, 203.8143]	335.578029	335.578029	0.001553	Ні
[300, -300]	[203.8143, -302.5249]	335.578029	335.578029	0.001383	Ні
[380, 380]	[420.9687, 420.9687]	0.000025	0.000025	0.001054	Так
[400, 400]	[420.9688, 420.9688]	0.000025	0.000025	0.000817	Так
[420, 420]	[420.9688, 420.9688]	0.000025	0.000025	0.000371	Так
[430, 430]	[420.9687, 420.9687]	0.000025	0.000025	0.000686	Так
[450, 450]	[420.9687, 420.9687]	0.000025	0.000025	0.000926	Так

Також для функції Міхалевича будемо позначати успіх роботи алгоритму, якщо відносна похибка буде менша за 1%.

Таблиця 3.4 Результат методу Ньютона для функціїї Міхалевича

Стартова точка	Знайдена точка	Значення функції	Відносна похибка (%)	Час роботи, с	Успіх
[0.2, 0.2]	[0.2, 0.2]	0	100	0.000366	Ні
[0.5, 0.5]	[0.5, 0.5]	0	100	0.000204	Ні
[1.0, 1.0]	[1.0, 1.5708]	-1	44.4845	0.001345	Ні
[2.5, 1.0]	[2.2029, 1.5708]	-1.801303	0.0002	0.00161	Так
[3.0, 3.0]	[3.0, 3.0]	0	100	0.000171	Ні
[2.0, 1.4]	[2.2029, 1.5708]	-1.801303	0.0002	0.001222	Так
[2.1, 1.5]	[2.2029, 1.5708]	-1.801303	0.0002	0.000864	Так
[2.2, 1.57]	[2.2029, 1.5708]	-1.801303	0.0002	0.000496	Так
[2.3, 1.6]	[2.2029, 1.5708]	-1.801303	0.0002	0.00122	Так
[2.4, 1.7]	[2.2029, 1.5708]	-1.801303	0.0002	0.001476	Так

Результати показують, що метод Ньютона може забезпечувати високу швидкість збіжності при старті поблизу глобального мінімуму. Проте, незважаючи на швидкість, метод залишається локальним і також залежить від вибору початкової точки.

Для багатоекстремальних функцій метод Ньютона не гарантує знаходження глобального оптимуму. Якщо початкова точка потрапляє в область притягання локального мінімуму, алгоритм може завершити роботу саме в цій точці.

3.2.3. Аналіз результатів традиційних методів

Проведені експерименти показали, що традиційні методи оптимізації мають суттєві обмеження при роботі з багатоекстремальними функціями.

Основними недоліками градієнтного спуску та методу Ньютона є:

1) залежність від початкової точки;
2) орієнтація на локальний пошук;
3) схильність до потрапляння у локальні мінімуми;
4) обмежена здатність досліджувати глобальний простір пошуку;
5) складність застосування до недиференційованих або неаналітичних функцій.

Метод Ньютона зазвичай демонструє швидшу збіжність, ніж градієнтний спуск, однак це не усуває його основного недоліку – локального характеру пошуку. Для складних багатоекстремальних функцій швидка збіжність може означати швидке потрапляння не до глобального, а до локального мінімуму.

Таким чином, результати традиційних методів підтверджують необхідність використання метаевристичних алгоритмів, які можуть досліджувати декілька областей простору пошуку та мають вищу ймовірність знаходження глобального оптимуму.

3.3. Дослідження роботи генетичного алгоритму на тестових функціях

3.3.1 Програмна реалізація генетичного алгоритму і опис параметрів

У межах даної роботи для дослідження генетичного алгоритму використовувались такі параметри:

1) розмір популяції: $n_{pop} = 100$;

2) кількість поколінь: $n_{iter} = 100$;

3) коефіцієнт кросоверу: $r_{cross} \in \{0.5; 0.9\}$;

4) коефіцієнт мутації: $r_{mut} \in \{0.03125; 0.1\}$.

Початкова популяція формується випадковим чином. Для кожної особини генерується двійкова хромосома довжиною $n_{bits} \cdot m$, де m— кількість змінних задачі. У даній роботі використовуються двовимірні функції, тому кожна хромосома кодує дві координати точки.

Значення коефіцієнта мутації $r_{mut} = 0.0315$ обрано як базове, оскільки воно відповідає поширеному підходу, за яким імовірність мутації одного біта визначається як величина, обернена до довжини хромосоми:

$$r_{mut} = \frac{1}{n_{bits} \cdot m}$$

У роботі для кодування однієї змінної використовується: n_{bits}=16, а кількість змінних для двовимірних функцій дорівнює: $m = 2$.

Відповідно: $r_{mut} = 0.03125$.

Такий вибір означає, що в середньому за одну операцію мутації змінюється приблизно один біт у хромосомі. Це дозволяє підтримувати різноманітність популяції, але не руйнувати повністю вже знайдені якісні розв’язки.

Після генерації популяції виконується декодування хромосом у дійсні значення координат. Для цього двійковий рядок перетворюється у ціле число, після чого масштабується відповідно до меж області пошуку.

Далі для кожної особини обчислюється значення цільової функції. На основі отриманих значень виконується селекція батьківських особин. У роботі досліджуються три способи селекції:

1) турнірний відбір;

2) рулетковий відбір;

3) ранговий відбір.

Після відбору батьків застосовується одноточковий кросовер. Його сутність полягає в тому, що дві батьківські хромосоми обмінюються частинами бітових рядків після випадково обраної точки розриву. Після цього до отриманих нащадків застосовується мутація, яка випадковим чином інвертує окремі біти хромосоми.

Критерієм завершення алгоритму є досягнення фіксованої кількості (100) поколінь.

У даному дослідженні для кожного набору параметрів виконувалось 100 запусків алгоритму, що дозволяє оцінити не лише найкращий результат, а й стабільність роботи методу. Успішність будемо оцінювати при абсолютній похибці ≤ 0.01 у функції Швефеля і відносній похибці $\leq 1\%$ у функції Міхалевича.

3.3.2 Результати екпериментів ГА

Продемонструємо результати роботи ГА при двох наборах параметрів:

Функція – Швефеля, спосіб відбору – турнірний ($k = 3$), коефіцієнт кросоверу – 0.9, коефіцієнт мутації – 0.03125.

Отримуємо.

1. Середнє f(x): 0.029909.
2. Найкраще f(x): 0.000034.
3. Найгірше f(x): 0.207357.
4. Середній час роботи: 0.284381 с.
5. Успішність алгоритму: 72.00% запусків дали похибку ≤ 0.01.

Розглянемо графік значень (рис. 3.1).

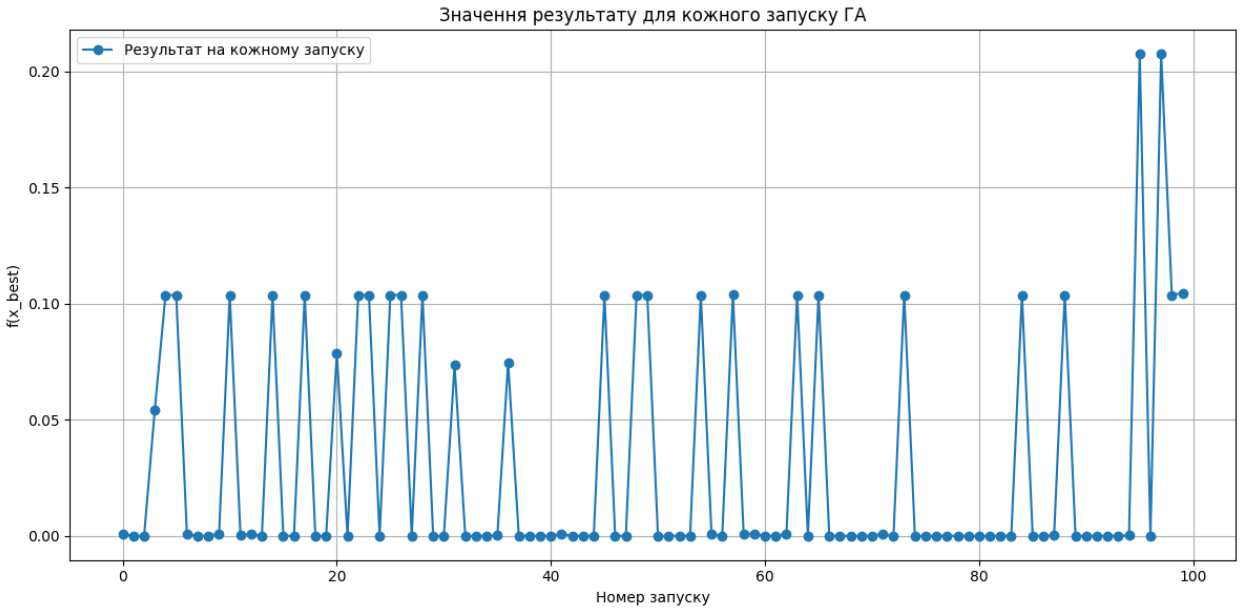


Рисунок 3.1 Приклад графіку експерименту для функції Швефеля

Функція – Міхалевича, спосіб відбору – турнірний ($k = 3$), коефіцієнт кросоверу – 0.9, коефіцієнт мутації – 0.03125.

Отримуємо.

1. Середнє f(x): -1.801299.
2. Найкраще f(x): -1.801303.
3. Найгірше f(x): -1.800863.
4. Середній час роботи: 0.300782 с.
5. Успішність алгоритму: 100.00% запусків дали похибку $\leq$ 1%.

Розглянемо графік значень (рис. 3.2).

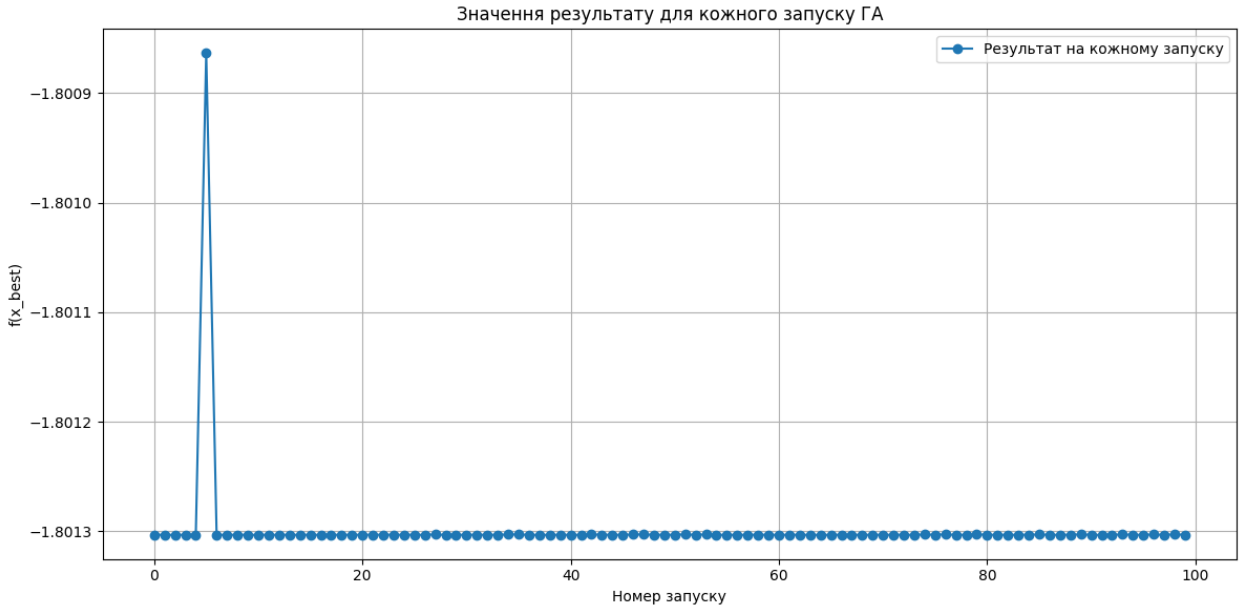


Рисунок 3.2. Приклад графіку експерименту для функції Міхалевича

Отже, розглянемо всі результати (рис. 3.3).

Функція	Спосіб відбору	r_cross	r_mut	Середній результат	Найкращий результат	Найгірший результат	Середній час, с	Успішність %
Функція Швефеля (мінімум: 0)	Турнірний	0.9	0.03125	0.029300	0.000034	0.207357	0.284381	72
		0.9	0.1	0.034601	0.000465	0.258639	0.296516	32
		0.5	0.03125	0.030463	0.000034	0.207357	0.293661	51
		0.5	0.1	0.049103	0.000301	0.316145	0.286745	24
	Рулеточний	0.9	0.03125	0.050733	0.000251	0.364414	0.293269	23
		0.9	0.1	0.515204	0.000093	2.693009	0.293939	4
		0.5	0.03125	0.076348	0.000282	1.551655	0.285810	27
		0.5	0.1	0.637349	0.008382	3.047488	0.287237	3
	Ранговий	0.9	0.03125	0.013841	0.000034	0.161535	0.385892	78
		0.9	0.1	0.190213	0.000112	1.521968	0.388109	8
		0.5	0.03125	0.014852	0.000034	0.240102	0.379276	85
		0.5	0.1	0.253836	0.002259	0.966439	0.382039	6
Функція Міхалевича (мінімум: -1.8013)	Турнірний	0.9	0.03125	-1.801299	-1.801303	-1.800863	0.300782	100
		0.9	0.1	-1.801202	-1.801303	-1.800748	0.296084	100
		0.5	0.03125	-1.801268	-1.801303	-1.800712	0.286368	100
		0.5	0.1	-1.801225	-1.801303	-1.800917	0.287100	100
	Рулеточний	0.9	0.03125	-1.801255	-1.801303	-1.800841	0.295330	100
		0.9	0.1	-1.801072	-1.801302	-1.799406	0.296638	100
		0.5	0.03125	-1.801230	-1.801302	-1.800541	0.285227	100
		0.5	0.1	-1.800978	-1.801303	-1.799020	0.286605	100
	Ранговий	0.9	0.03125	-1.801296	-1.801303	-1.800857	0.388671	100
		0.9	0.1	-1.800913	-1.801301	-1.798754	0.381535	100
		0.5	0.03125	-1.801302	-1.801303	-1.801289	0.360242	100
		0.5	0.1	-1.800814	-1.801303	-1.797043	0.365067	100

Рисунок 3.3 Результати всіх есперименті ГА

Результати експериментальних досліджень показують, що ефективність генетичного алгоритму суттєво залежить від способу селекції, коефіцієнта

кросоверу та коефіцієнта мутації. Аналіз проводився окремо для функцій Швефеля та Міхалевича.

Для функції Швефеля найбільш складною задачею є пошук глобального мінімуму в умовах великої кількості локальних екстремумів. Отримані результати показують, що найкращу ефективність продемонстрував ранговий спосіб селекції при параметрах:

$$r_{cross} = 0.5, r_{mut} = 0.03125$$

У цьому випадку:

1) середній результат становив 0.014852;
2) найкращий результат дорівнював 0.000034;
3) успішність алгоритму становила 85%.

Також високі результати продемонстрував ранговий відбір при:

$$r_{cross} = 0.9, r_{mut} = 0.03125,$$

де успішність становила 78%.

Турнірний відбір показав дещо нижчу стабільність. Найкращий результат для цього способу селекції було отримано при:

$$r_{cross} = 0.9, r_{mut} = 0.03125$$

У цьому випадку успішність становила 72%, а середній результат дорівнював 0.029300.

Рулетковий відбір продемонстрував найгірші результати серед усіх досліджених методів селекції. При збільшенні коефіцієнта мутації до:

$$r_{mut} = 0.1$$

успішність алгоритму знижувалась до $3 - 4\%$, а середні результати значно погіршувались. Наприклад, для параметрів:

$$r_{cross} = 0.5, r_{mut} = 0.1$$

середній результат становив 0.637349, а найгірший результат досягав 3.047488.

Отримані результати свідчать про те, що надто велике значення коефіцієнта мутації негативно впливає на збіжність генетичного алгоритму. При високому рівні мутації популяція втрачає стабільність, а процес пошуку набуває надмірно випадкового характеру.

Для функції Міхалевича результати виявились значно стабільнішими. Незалежно від способу селекції та параметрів алгоритму успішність у всіх експериментах становила 100%. Це пояснюється менш складною структурою функції порівняно з функцією Швефеля.

Найкращі результати для функції Міхалевича були отримані при використанні рангового відбору та параметрів:

$$r_{cross} = 0.5, r_{mut} = 0.03125$$

де середній результат становив:

$$f(x) = -1.801302$$

що практично співпадає з теоретичним глобальним мінімумом функції.

Також слід зазначити, що середній час роботи алгоритму залишався стабільним для всіх конфігурацій параметрів та становив приблизно:

1) $0.28 - 0.30$ секунди для турнірного та рулеткового відбору;
2) $0.36 - 0.39$ секунди для рангового відбору.

Збільшення часу роботи при ранговому відборі пояснюється необхідністю додаткового сортування особин за значенням функції пристосованості.

Таким чином, проведені експерименти показали, що генетичний алгоритм демонструє значно вищу ефективність порівняно з традиційними локальними методами оптимізації. Алгоритм здатний уникати локальних мінімумів та виконувати глобальний пошук навіть для складних багатоекстремальних функцій. Найбільш ефективним способом селекції у межах даного дослідження виявився ранговий відбір при помірному значенні коефіцієнта мутації.

3.4. Дослідження роботи алгоритму спірального пошуку на тестових функціях

3.4.1. Програмна реалізація методу спірального пошуку та опис параметрів

У межах даної роботи для дослідження методу спірального пошуку використовувались такі параметри:

1) кількість точок: $n_{points} \in \{100; 200; \}$;
2) кількість ітерацій: $n_{iter} = 100$;
3) кут обертання: $\theta \in \left\{\frac{\pi}{3}; \frac{\pi}{4}; \frac{\pi}{6}\right\}$;
4) коефіцієнт наближення: $r \in \{0.98; 0.95; 0.9\}$.

Початкова множина точок генерується випадковим чином у межах області пошуку. Після генерації початкової множини точок для кожної з них обчислюється значення цільової функції. Найкращою вважається точка з мінімальним значенням функції. Основна ідея методу полягає у поступовому наближенні всіх точок до поточного найкращого розв’язку за спіральною траєкторією. У програмній реалізації використовувався бар’єрний метод обмеження області пошуку. Якщо точка виходила за межі допустимої області, її значення функції прирівнювалось до $+\infty$.

При цьому координати точки не обрізались штучно, що дозволило зберегти природну геометрію спірального руху та уникнути скупчення точок на межах області пошуку.

Критерієм завершення алгоритму є досягнення фіксованої кількості (100) ітерацій.

У межах дослідження для кожного набору параметрів виконувалось 100 запусків алгоритму, що дозволяє оцінити не лише найкращий результат, а й стабільність роботи методу. Успішність будемо оцінювати при абсолютній похибці ≤ 0.01 у функції Швефеля і відносній похибці $\leq 1\%$ у функції Міхалевича.

3.4.2 Результати екпериментів спірального пошуку

Розглянемо зміну положень точок під час роботи СП з параметрами: Функція – Швефеля, кількість точок-100, кут обертання $-\frac{\pi}{3}$, коефіцієнт наближення -0.95. (рис. 3.4).

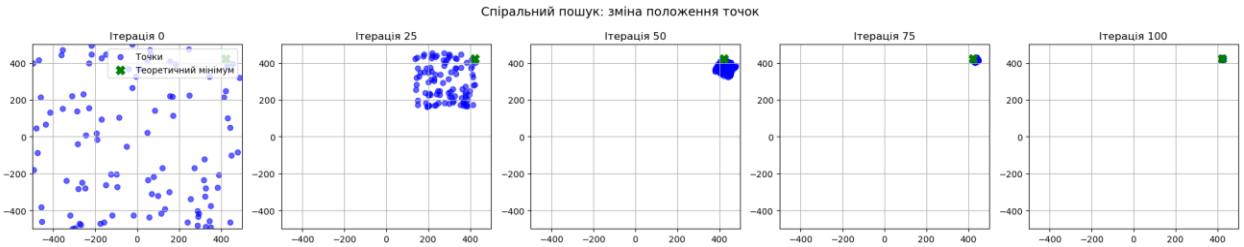


Рисунок 3.4. Зміна положень точок під час ітерацій

Продемонструємо результати роботи СП при двох наборах параметрів:

Функція – Швефеля, кількість точок-100, кут обертання $-\frac{\pi}{3}$, коефіцієнт наближення -0.95.

Отримуємо.

1. Середнє f(x): 35.533839.
2. Найкраще f(x): 0.000067.
3. Найгірше f(x): 236.880996.
4. Середній час роботи: 0.059161с.

5. Успішність алгоритму: 70.00% запусків дали похибку ≤ 0.0.

Розглянемо графік значень (рис. 3.5).

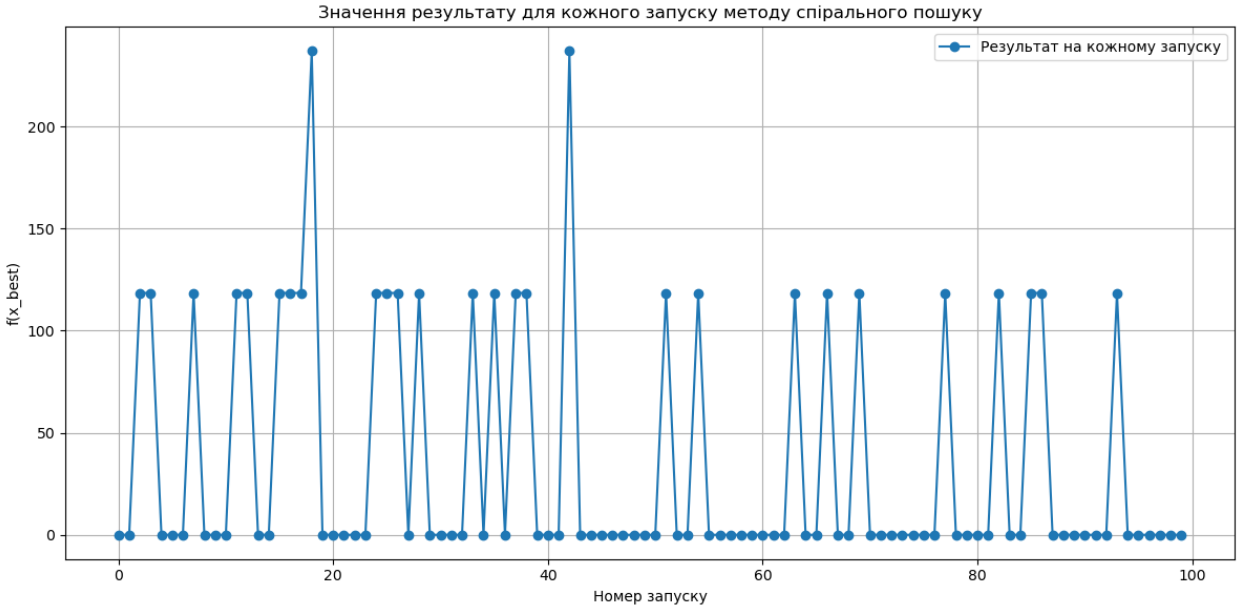


Рисунок 3.5 Приклад графіку експерименту для функції Швефеля

Функція – Міхалевича, кількість точок-100, кут обертання $-\frac{\pi}{3}$, коефіцієнт наближення $-$ 0.95.

Отримуємо.

1. Середнє f(x): -1.801298.
2. Найкраще f(x): -1.801303.
3. Найгірше f(x): -1.801269.
4. Середній час роботи: 0.065320 с.
5. Успішність алгоритму: 100.00% запусків дали похибку ≤ 1%.

Розглянемо графік значень (рис. 3.6).

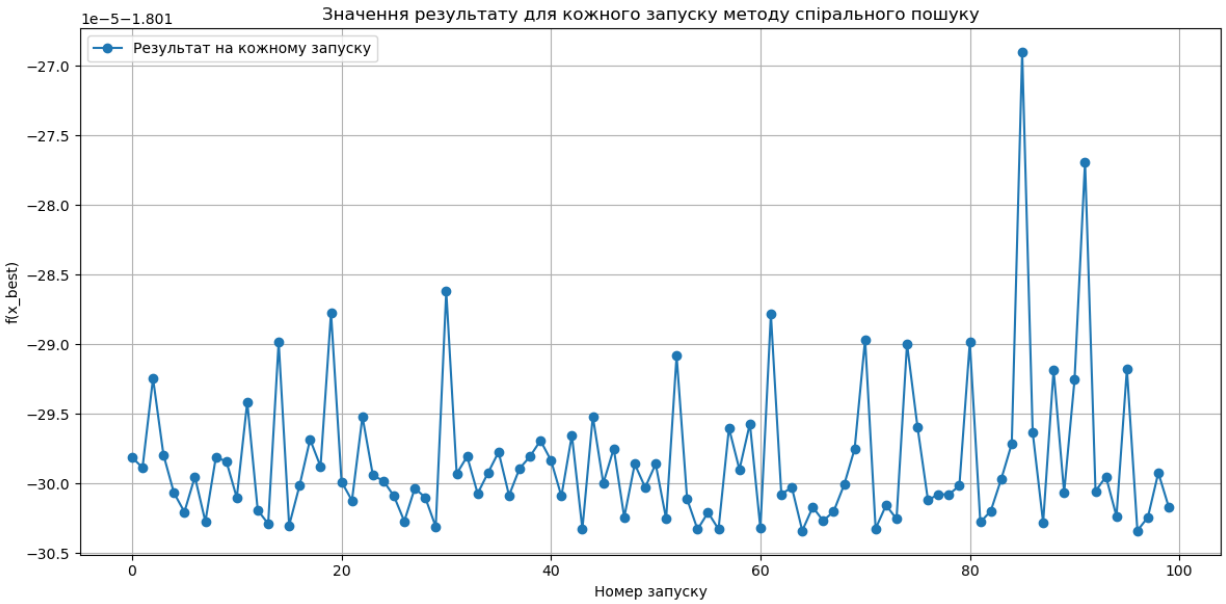


Рисунок 3.6. Приклад графіку експерименту для функції Міхалевича

Отже, розглянемо всі результати: (Рис. 3.7).

Функція	Кількість точок	theta	r	Середній результат	Найкращий результат	Найгірший результат	Середній час, с	Успішність %
Функція Швефеля (мінімум: 0)	100	π/3	0.98	13.387233	0.000119	119.932817	0.056777	1
		π/3	0.95	35.533839	0.000067	236.880996	0.059161с	70
		π/3	0.90	46.190976	0.000025	236.876695	0.061111	62
		π/4	0.98	11.944156	0.019953	120.127952	0.055276	0
		π/4	0.95	37.902771	0.000102	236.877622	0.057585	66
		π/4	0.90	58.034810	0.000025	236.876695	0.058778	56
		π/6	0.98	18.021620	0.018597	237.736173	0.053301	0
		π/6	0.95	52.119625	0.000867	236.886625	0.058142	49
		π/6	0.90	53.297395	0.000026	236.876713	0.060789	59
	200	π/3	0.98	6.585722	0.006336	120.417721	0.112336	2
		π/3	0.95	13.029322	0.000047	236.876802	0.113946	90
		π/3	0.90	27.240842	0.000025	236.876695	0.117892	78
		π/4	0.98	11.320786	0.006472	120.259012	0.110684	1
		π/4	0.95	11.845229	0.000092	118.443851	0.117005	90
		π/4	0.90	34.347143	0.000025	118.438360	0.122276	71
		π/6	0.98	7.948646	0.026362	121.838537	0.110079	0
		π/6	0.95	31.981215	0.000411	236.881606	0.123854	74
		π/6	0.90	22.503314	0.000025	118.438758	0.121418	81
Функція Міхалевича (мінімум: -1.8013)	100	π/3	0.98	-1.798976	-1.801155	-1.787591	0.065151	100
		π/3	0.95	-1.801298	-1.801303	-1.801269	0.065320	100
		π/3	0.90	-1.793290	-1.801303	-1.000000	0.069722	99
		π/4	0.98	-1.799257	-1.801269	-1.792883	0.066520	100
		π/4	0.95	-1.801298	-1.801303	-1.801269	0.068943	100
		π/4	0.90	-1.793290	-1.801303	-1.000000	0.066826	99
		π/6	0.98	-1.799244	-1.801243	-1.791019	0.066331	100
		π/6	0.95	-1.801296	-1.801303	-1.801260	0.065568	100
		π/6	0.90	-1.801303	-1.801303	-1.801297	0.065859	100
	200	π/3	0.98	-1.800074	-1.801302	-1.794582	0.133938	100
		π/3	0.95	-1.801302	-1.801303	-1.801290	0.133200	100
		π/3	0.90	-1.801303	-1.801303	-1.801303	0.133358	100
		π/4	0.98	-1.800327	-1.801298	-1.797045	0.133911	100
		π/4	0.95	-1.801302	-1.801303	-1.801292	0.133177	100
		π/4	0.90	-1.801303	-1.801303	-1.801303	0.134175	100
		π/6	0.98	-1.800197	-1.801267	-1.792557	0.133230	100
		π/6	0.95	-1.801300	-1.801303	-1.801287	0.133280	100
		π/6	0.90	-1.801303	-1.801303	-1.801303	0.137381	100

Рисунок 3.7. Результати всіх есперименті СП

Отримані результати демонструють суттєвий вплив параметрів методу спірального пошуку на ефективність глобальної оптимізації.

Для функції Швефеля найкращі результати були отримані при використанні:

$$r = 0.95$$

та кількості точок:

$$n_{points} = 200$$

У даному випадку успішність алгоритму досягала 90% при кутах обертання:

$$\theta = \frac{\pi}{3} \text{ та } \theta = \frac{\pi}{4}$$

Отримані результати свідчать про те, що збільшення кількості точок з 100 до 200 дозволяє покращити стабільність глобального пошуку та підвищити ймовірність знаходження глобального мінімуму. Це пояснюється більш щільним покриттям області пошуку та кращим дослідженням простору розв’язків.

Водночас збільшення коефіцієнта наближення до: $r = 0.98$, призводило до значного зниження успішності алгоритму. У цьому випадку спостерігалось надто повільне звуження області пошуку, через що алгоритм не встигав достатньо точно наблизитись до глобального мінімуму за фіксовану кількість ітерацій.

При: $r = 0.90$, навпаки відбувалось надто швидке стискання траєкторій пошуку, що інколи призводило до передчасної збіжності та погіршення стабільності результатів.

Для функції Міхалевича результати виявились значно стабільнішими. Практично для всіх конфігурацій параметрів успішність алгоритму становила 100%.

Це пояснюється менш складною структурою функції порівняно з функцією Швефеля та меншою кількістю локальних екстремумів.

Найкращі результати для функції Міхалевича були отримані при:

$$r = 0.90,$$

де середнє значення функції практично співпадало з теоретичним глобальним мінімумом:

$$f(x) = -1.801303.$$

Середній час роботи алгоритму залишався стабільним для всіх конфігурацій параметрів та становив приблизно:

1) 0.05−0.12 с для функції Швефеля;
2) 0.06−0.13 с для функції Міхалевича.

Таким чином, проведені експерименти показали, що метод спірального пошуку є ефективним інструментом глобальної оптимізації багатоекстремальних функцій. Алгоритм демонструє високу швидкість роботи, просту реалізацію та здатність успішно знаходити глобальний мінімум навіть у складних просторах пошуку. Найбільш ефективними параметрами в межах проведеного дослідження виявились:

$$n_{points} = 200, \theta = \frac{\pi}{3} \text{ і } r = 0.95,$$

що забезпечило найкращий баланс між точністю пошуку та стабільністю роботи алгоритму.

3.5. Дослідження роботи алгоритмів на задачі розміщення складів

Окрім класичних тестових функцій, у даній роботі розглядається прикладна задача розміщення складів. Її використання дозволяє перевірити роботу генетичного алгоритму та методу спірального пошуку на цільовій функції, що має практичний зміст.

Задача полягає у знаходженні таких координат складів, за яких сумарні транспортні витрати для обслуговування заданої множини міст будуть мінімальними. Кожне місто має координати та значення попиту. Кожен склад також має координати, які визначаються алгоритмом оптимізації.

3.5.1 Програмна реалізація алгоритмів і опис параметрів

У межах даної роботи для дослідження алгоритмів використовувались такі параметри:

1) розмір популяції/кількість точок: $n_{pop}/n_{points} \in \{100; 200\}$;
2) кількість ітерацій: $n_{iter} = 100$;
3) кількість складів змінювалася так: $k \in \{2; 3; 5\}$.

Таблиця 3.5 – Приклад вхідних даних задачі розміщення складів

№ міста	Координата a_i	Координата b_i	Попит D_i
1	10	20	120
2	20	80	80
3	30	50	150
4	40	30	100
5	50	90	90

Повний набір міст використовується можна побачити в ДОДАТКУ А. Під час кожного запуску алгоритмів координати міст і значення попиту залишаються сталими. Для n міст в експерименті вибираються перші n міст із набору.

Для перевірки ефективності генетичного алгоритму та методу спірального пошуку було сформовано 12 експериментів. Вони відрізняються

кількістю початкових точок, кількістю міст, кількістю складів і відповідною розмірністю простору пошуку.

Для кожного алгоритму вибиралися найкращі параметри із результатів на тестових функціях.

Тобто для ГА: $r_{cross} = 0.5$; $r_{mut} = \frac{1}{n_{bits} \cdot m}$, де $n_{bits} = 16$, $m = 2k$, $k -$ кількість складів; спосіб селекції – ранговий відбір.

Для СП: $\theta = \frac{\pi}{3}$ і $r = 0.95$.

Критерієм завершення алгоритму є досягнення фіксованої кількості (100) ітераці. У даному дослідженні для кожного набору параметрів виконувалось 100 запусків алгоритму, що дозволяє оцінити не лише найкращий результат, а й стабільність роботи методу. Стандартне відхилення використовується для оцінювання стабільності алгоритму.

Стандартне відхилення визначається як:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(F_i - \bar{F})^2},$$

де $F_i -$ результат i-го запуску;

$\bar{F} -$ середнє значення цільової функції за всіма запусками;

$n -$ кількість запусків.

Чим менше значення стандартного відхилення, тим стабільнішою є робота алгоритму.

3.5.2 Результати екпериментів алгоритмів.

Продемонструємо результати роботи алгоритмів для першого експеримента:

Розмір популяції/кількість точок – 100, кількість міст – 10, кількість складів – 2.

Генетичний алгоритм.

1. Середній результат: 26400.458923027087.
2. Найкращий результат: 26246.004159370284.
3. Найгірший результат: 26775.186183382713.
4. Стандартне відхилення: 189.93101652891437.
5. Середній час виконання: 1.2146013830001174.
6. Середня кількість обчислень ЦФ: 10001.0.
7. Координати складів для найкращого запуску:
   [[29.00695801, 43.60046387]
   [80.00030518, 39.99938965]]

Спіральний пошук.

1. Середній результат: 26266.08297189202.
2. Найкращий результат: 26245.97585561559.
3. Найгірший результат: 26335.79562561885.
4. Стандартне відхилення: 32.8684370239648.
5. Середній час виконання: 0.6448374469999544.
6. Середня кількість обчислень ЦФ: 10100.0.
7. Координати складів для найкращого запуску:
   [[79.99942977, 39.9992378]
   [28.94146447, 43.4787967]]

Розглянемо графік середньої збіжності алгоритмів (Рис. 3.8).

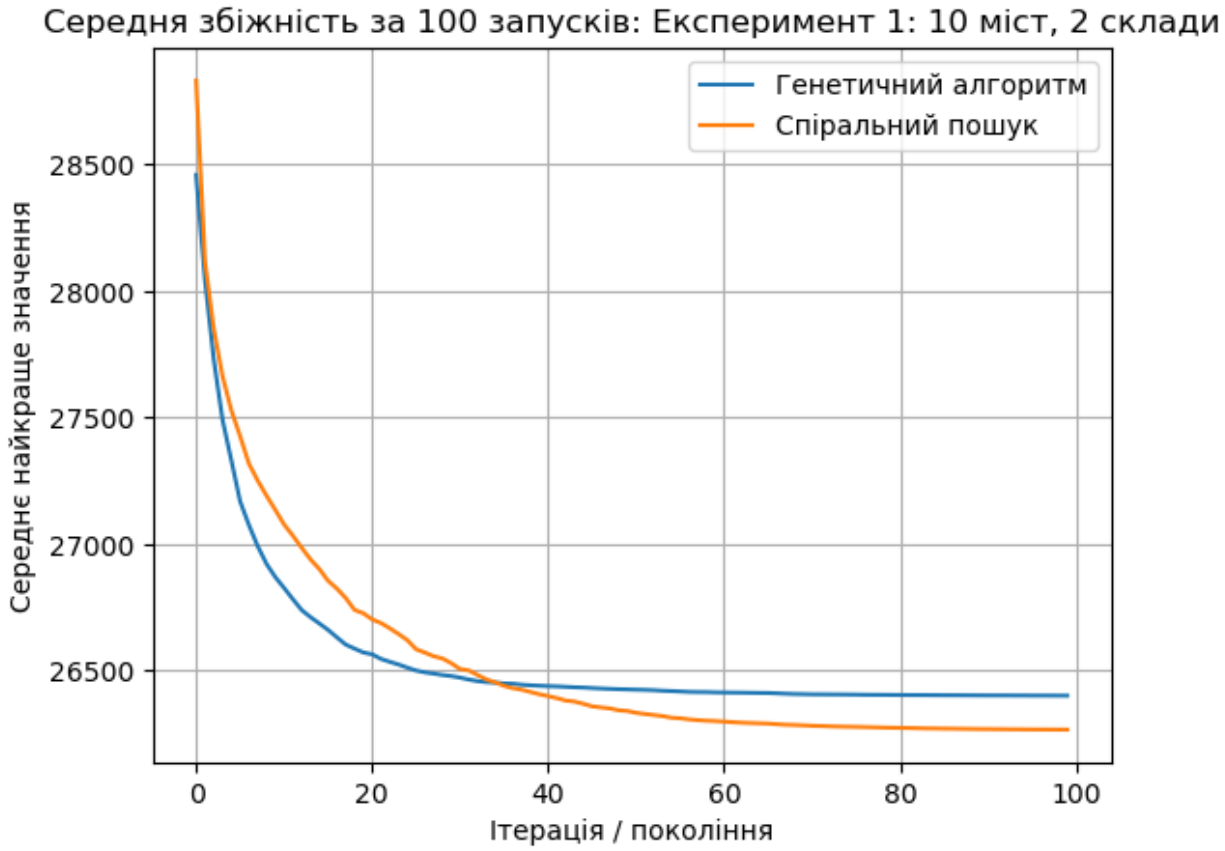


Рисунок 3.8 Приклад графіку середньої збіжності

Розглянемо графіки найкращого розміщення складів для кожного алгоритму (рис 3.9), (рис 3.10).

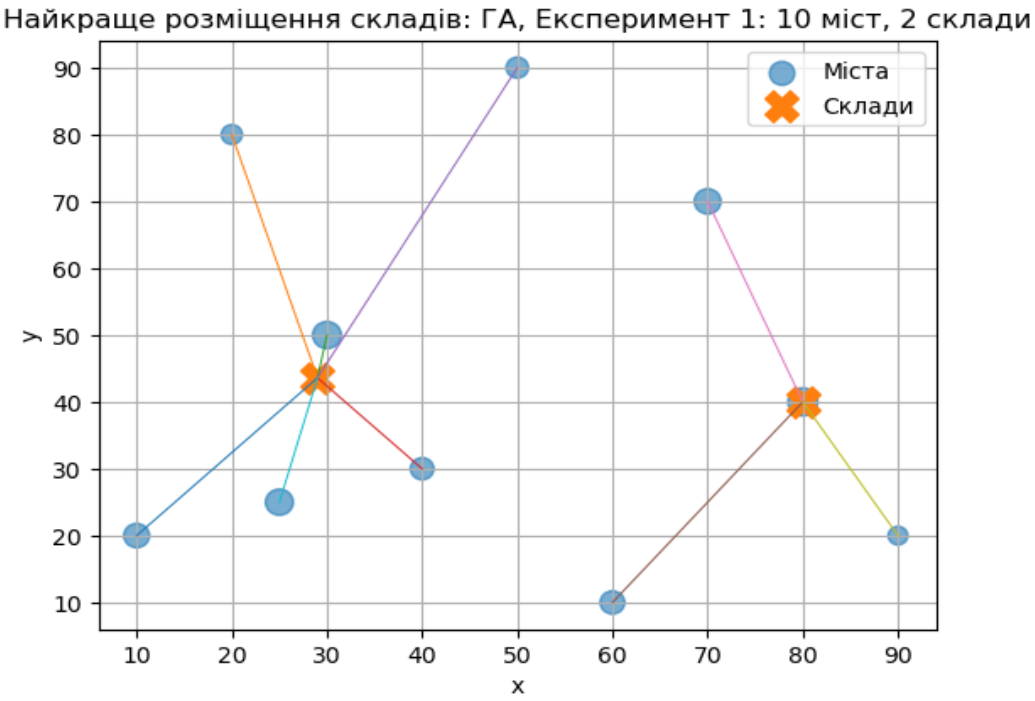


Рисунок 3.9 Найкраще розміщення для ГА

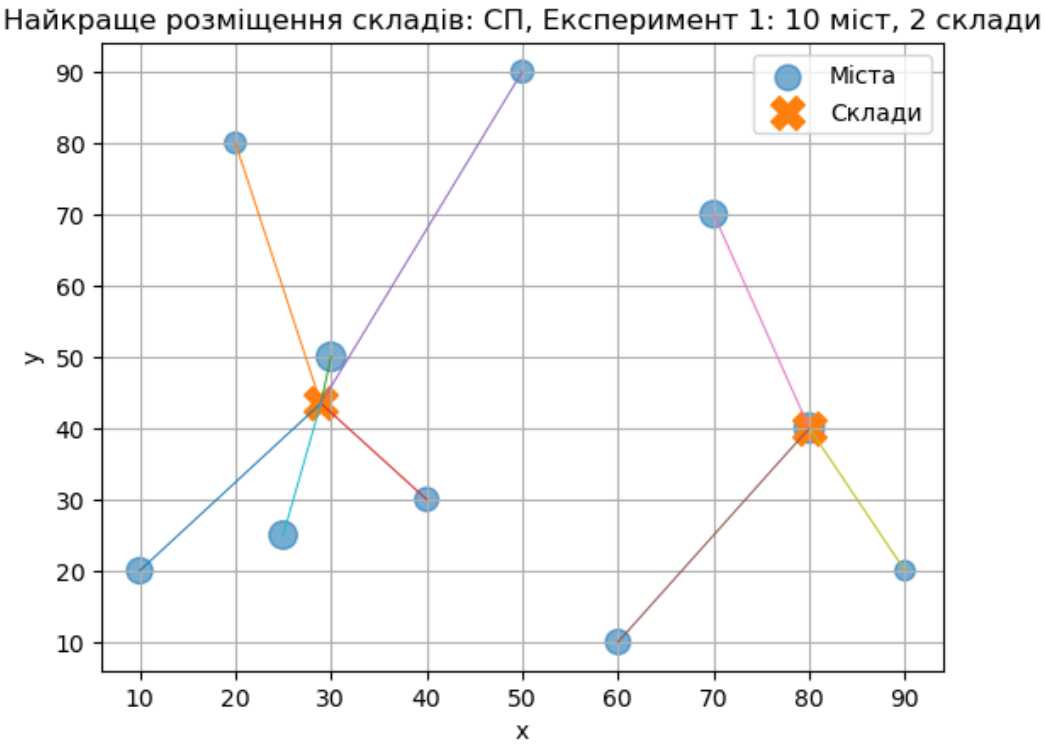


Рисунок 3.10 Найкраще розміщення для СП

Рисунки інших експериментів та координати складів додані в ДОДАТОК А.

Отже, розглянемо всі результати (рис. 3.11).

Кількість популяцій/точок	Алгоритм	Кількість міст	Кількість складів	Середній результат	Найкращий результат	Найгірший результат	Середній час, с	Стандартне відхилення
100	Генетичний алгоритм	10	2	26400.4589	26246.0042	26775.1862	1.2146	189.9310
		10	3	19048.6864	18947.2327	19412.5356	1.4789	125.4877
		20	2	60340.1663	60270.2368	62274.7835	1.9605	209.2814
		20	3	47681.1269	47239.7435	48722.0979	2.5493	326.3064
		50	3	121460.5344	120843.0679	125212.7868	5.7978	892.0543
		50	5	91670.5325	90344.7404	94819.3081	6.6720	1094.5269
	Спіральний пошук	10	2	26266.0830	26245.9759	26335.7956	0.6448	32.8684
		10	3	18963.1045	18947.0184	19667.7965	0.7144	100.3481
		20	2	60270.3422	60270.2241	60270.6175	1.3181	0.0712
		20	3	47385.0955	47239.7160	48163.1166	1.5652	323.2317
		50	3	120900.1877	120842.4356	121080.8390	4.4001	64.8905
		50	5	90798.3141	90273.8551	93429.8498	4.6894	765.2920
200	Генетичний алгоритм	10	2	26301.1330	26246.1629	26760.0835	2.9434	121.2303
		10	3	18985.0622	18946.6891	19203.9998	3.8327	60.7424
		20	2	60275.0170	60270.2159	60446.1909	5.4503	22.2380
		20	3	47551.1650	47239.8000	48280.5360	6.5408	308.3659
		50	3	121077.3704	120843.6980	124189.0043	9.2035	430.1535
		50	5	91264.9009	90289.2492	94417.4837	11.5321	911.9609
	Спіральний пошук	10	2	26257.3625	26246.4028	26332.1371	1.2909	25.8911
		10	3	18947.9393	18946.7903	18949.7167	1.5910	0.6608
		20	2	60270.2954	60270.2175	60270.4388	3.1676	0.0449
		20	3	47396.4682	47239.7043	48162.3611	3.5693	336.5206
		50	3	120894.2064	120843.6477	121018.1391	6.5143	66.4399
		50	5	90729.1056	90269.3624	93430.6666	7.7570	758.4792

Рисунок 3.11 Результати всіх есперименті

Аналіз результатів показує, що зі збільшенням кількості міст значення цільової функції зазвичай зростає. Це є очікуваним, оскільки більша кількість міст створює більший сумарний попит і потребує обслуговування більшої кількості точок.

Збільшення кількості складів має подвійний вплив. З одного боку, більша кількість складів дозволяє зменшити відстані від міст до найближчого складу, тому значення цільової функції може зменшуватися. З іншого боку, зі збільшенням кількості складів зростає розмірність задачі:

$$m = 2k.$$

Отже, при збільшенні кількості складів алгоритм повинен оптимізувати більшу кількість координат, що ускладнює процес пошуку.

3.6. Узагальнення результатів експериментів

У третьому розділі було проведено експериментальне дослідження традиційних методів оптимізації, генетичного алгоритму та методу спірального пошуку. Дослідження виконувалося на двох класичних тестових багатоекстремальних функціях — функції Швефеля та функції Міхалевича, а також на прикладній задачі розміщення складів.

Результати експериментів із традиційними методами оптимізації показали, що градієнтний спуск і метод Ньютона суттєво залежать від вибору початкової точки. Якщо початкова точка розташована поблизу глобального мінімуму, ці методи можуть швидко знайти якісний розв’язок. Однак у випадку старту з інших областей простору пошуку вони часто збігаються до локальних мінімумів або не досягають глобального оптимуму.

Для багатоекстремальних функцій це є суттєвим обмеженням, оскільки локальний мінімум не обов’язково є найкращим розв’язком на всій області пошуку. Тому традиційні методи доцільно використовувати переважно для локального уточнення вже знайденого розв’язку, але не як основний інструмент глобальної оптимізації.

Експерименти з генетичним алгоритмом показали, що ефективність методу суттєво залежить від способу селекції, коефіцієнта кросоверу та коефіцієнта мутації. Найкращі результати на тестових функціях були отримані при використанні рангового відбору та помірного значення коефіцієнта мутації.

Для функції Швефеля найкращу ефективність продемонстрував ранговий відбір при параметрах:

$$r_{cross} = 0.5,$$

$$r_{mut} = 0.03125.$$

У цьому випадку середній результат становив:

$$\bar{f} = 0.014852,$$

а найкращий результат:

$$f_{best} = 0.000034.$$

Успішність алгоритму становила:

$$SuccessRate = 85\%.$$

Для функції Міхалевича генетичний алгоритм показав стабільніші результати. У більшості експериментів успішність становила:

$$SuccessRate = 100\%.$$

Експерименти з методом спірального пошуку показали, що на його ефективність суттєво впливають кількість точок, коефіцієнт наближення та кут обертання.

Для функції Швефеля найкращі результати було отримано при параметрах:

$$n_{points} = 200,$$

$$r = 0.95,$$

$$\theta = \frac{\pi}{3}.$$

Або:

$$\theta = \frac{\pi}{4}.$$

У цих випадках успішність алгоритму досягала:

$$SuccessRate = 90\%.$$

Це свідчить про те, що збільшення кількості точок з 100 до 200 покращує покриття області пошуку та підвищує ймовірність знаходження глобального мінімуму.

Для функції Міхалевича метод спірального пошуку також показав стабільні результати.

На основі результатів тестових функцій для прикладної задачі розміщення складів було обрано параметри, які забезпечили найкращу стабільність і якість пошуку.

Для генетичного алгоритму було використано:

$$r_{cross} = 0.5,$$

$$selection\_method = 3,$$

де $selection\_method = 3$ відповідає ранговому відбору.

Коефіцієнт мутації визначався за формулою:

$$r_{mut} = \frac{1}{n_{bits} \cdot 2k}.$$

Для методу спірального пошуку було використано:

$$\theta = \frac{\pi}{3},$$

$$r = 0.95.$$

Аналіз результатів експериментів показує, що обидва досліджувані алгоритми можуть бути використані для розв’язання задачі розміщення складів. У всіх експериментах генетичний алгоритм і метод спірального пошуку знаходили близькі за якістю значення цільової функції, що підтверджує їх придатність для прикладної задачі логістичного типу.

У першому експерименті, де розглядалося:

$$N = 10,$$

$$k = 2,$$

розмірність задачі становила:

$$m = 2k = 2 \cdot 2 = 4.$$

Для цього випадку найкращі результати алгоритмів були майже однаковими:

$$F_{best}^{GA} = 26246.0042,$$

$$F_{best}^{SP} = 26245.9759.$$

Різниця між ними становила лише:

$$\Delta F = 26246.0042 - 26245.9759 = 0.0283.$$

Це свідчить про те, що обидва алгоритми змогли знайти практично однакове розміщення складів. Водночас метод спірального пошуку мав менше стандартне відхилення:

$$\sigma^{SP} = 32.8684,$$

порівняно з генетичним алгоритмом:

$$\sigma^{GA} = 189.9310.$$

Тому для цього експерименту СП показав більш стабільну роботу.

Зі збільшенням кількості складів значення цільової функції зменшується. Наприклад, при переході від:

$$k = 2$$

до:

$$k = 3$$

для $N = 10$ середній результат обох алгоритмів зменшується приблизно з:

$$F(X) \approx 26000$$

до:

$$F(X) \approx 19000.$$

Це пояснюється тим, що більша кількість складів дозволяє розмістити їх ближче до міст і зменшити сумарні транспортні витрати.

При збільшенні кількості міст до $N = 20$ та $N = 50$, значення цільової функції закономірно зростає, оскільки збільшується кількість точок попиту, які необхідно обслуговувати. Цільова функція містить суму за всіма містами:

$$F(X) = \sum_{i=1}^{N} D_i \cdot \min_{j=1,\dots,k} d(c_i, w_j),$$

тому зі збільшенням N зростає і загальна величина транспортних витрат.

Найскладнішим є експеримент з параметрами:

$$N = 50,$$

$$k = 5.$$

У цьому випадку розмірність простору пошуку становить:

$$m = 2k = 2 \cdot 5 = 10.$$

Тобто алгоритми повинні оптимізувати десять координат:

$$X = (x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, x_5, y_5).$$

Збільшення кількості складів зменшує транспортні витрати, але одночасно ускладнює процес оптимізації через зростання розмірності задачі.

Порівняння експериментів із кількістю популяцій/точок 100 та 200 показує, що збільшення цього параметра може покращувати стабільність результатів, але також збільшує час виконання. Наприклад, для генетичного алгоритму при:

$$N = 50,$$

$$k = 5$$

середній час роботи зріс з:

$$6.6720 \text{ с}$$

до:

$$11.5321 \text{ с}.$$

Це пов’язано з тим, що більша популяція потребує більшої кількості обчислень відстаней у цільовій функції на кожному поколінні.

Загалом за результатами таблиці можна зробити висновок, що метод спірального пошуку у більшості експериментів демонструє менший час роботи та менше стандартне відхилення, тобто працює швидше і стабільніше. Генетичний алгоритм також знаходить якісні розв’язки, однак його

результати мають дещо більший розкид, що пояснюється стохастичним характером селекції, кросоверу та мутації.

Таким чином, для задачі розміщення складів обидва алгоритми є придатними. Якщо основним критерієм є швидкість і стабільність, перевагу можна надати методу спірального пошуку. Якщо ж важливим є ширше дослідження простору можливих розв’язків, доцільним є використання генетичного алгоритму. У межах проведених експериментів метод спірального пошуку показав дещо кращі практичні результати за часом роботи та стабільністю.

3.7. Висновки до розділу 3

У третьому розділі було розглянуто програмну реалізацію та проведено експериментальне дослідження методів оптимізації багатоекстремальних функцій. Для цього було реалізовано традиційні методи оптимізації, генетичний алгоритм та метод спірального пошуку мовою програмування Python.

Експерименти з градієнтним спуском і методом Ньютона показали, що традиційні методи суттєво залежать від вибору початкової точки. Якщо початкова точка розташована поблизу глобального мінімуму, такі методи можуть швидко знайти якісний розв’язок. Проте для складних багатоекстремальних функцій вони часто збігаються до локальних мінімумів, тому не є достатньо надійними для глобального пошуку.

Дослідження генетичного алгоритму показало, що його ефективність залежить від способу селекції, коефіцієнта кросоверу та коефіцієнта мутації.

Найкращі результати в межах проведених експериментів були отримані при використанні рангового відбору та параметрів:

$$r_{cross} = 0.5,$$
$$r_{mut} = 0.03125.$$

Для функції Швефеля генетичний алгоритм продемонстрував здатність знаходити значення, близькі до глобального мінімуму, а для функції Міхалевича результати були стабільними майже для всіх конфігурацій параметрів.

Метод спірального пошуку також показав високу ефективність при оптимізації тестових багатоекстремальних функцій. Найкращі результати для функції Швефеля були отримані при параметрах:

$$n_{points} = 200,$$
$$r = 0.95,$$
$$\theta = \frac{\pi}{3}.$$

Такі параметри забезпечили достатнє покриття області пошуку та стабільну збіжність алгоритму.

Окремо було досліджено прикладну задачу розміщення складів, у якій необхідно мінімізувати сумарні транспортні витрати:

$$F(X) = \sum_{i=1}^{N} D_i \cdot \min_{j=1,\dots,k} \sqrt{(a_i - x_j)^2 + (b_i - y_j)^2}.$$

Для цієї задачі було проведено серію експериментів із різною кількістю міст і складів. Результати показали, що збільшення кількості складів зменшує значення цільової функції, але водночас збільшує розмірність простору пошуку:

$$m = 2k.$$

Порівняння алгоритмів на задачі розміщення складів показало, що обидва методи здатні знаходити якісні розв’язки. Метод спірального пошуку у більшості експериментів демонстрував менший час роботи та менше стандартне відхилення, тобто працював швидше і стабільніше. Генетичний алгоритм також показав хороші результати завдяки популяційному характеру пошуку, кросоверу та мутації.

Таким чином, проведені експерименти підтвердили доцільність використання метаевристичних методів для оптимізації багатоекстремальних функцій. Генетичний алгоритм і метод спірального пошуку можуть застосовуватися як для тестових математичних функцій, так і для прикладних задач логістичного типу.

РОЗДІЛ 4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі проводиться оцінка основних характеристик програмного продукту, призначеного для порівняння метаевристичних методів оптимізації багатоекстремальних функцій. Розроблений програмний продукт реалізує генетичний алгоритм та метод спірального пошуку і дає змогу проводити серії обчислювальних експериментів для аналізу ефективності цих методів.

Основною метою програмного продукту є не лише знаходження мінімуму окремої функції, а й забезпечення можливості порівняння двох різних підходів до глобальної оптимізації. Генетичний алгоритм базується на принципах еволюційного пошуку, використовує популяцію розв’язків, оператори селекції, кросоверу та мутації. Метод спірального пошуку, у свою чергу, ґрунтується на геометричному переміщенні множини точок навколо поточного найкращого розв’язку з поступовим звуженням області пошуку.

У розділі аналізуються різні варіанти реалізації програмного продукту з метою вибору оптимального варіанта архітектури. При цьому враховуються як економічні фактори, зокрема витрати на розробку та використання ресурсів, так і технічні характеристики продукту, що безпосередньо впливають на швидкість виконання експериментів, зручність аналізу результатів, функціональність і можливість подальшого розширення системи.

Для вирішення цього завдання було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) — це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії чи конкретної команди розробників. Як прямі, так і побічні

витрати розподіляються за окремими функціями продукту залежно від потрібних на кожному етапі проектування та розробки обсягів ресурсів.

Мета ФВА полягає у забезпеченні раціонального розподілу ресурсів, виділених на проектування, програмування, тестування та оформлення програмного продукту. У даному випадку аналіз спрямований на дослідження ключових функцій програмного продукту та виявлення витрат на їх безпосередню алгоритмічну і програмну реалізацію.

Фактично цей метод працює за таким алгоритмом:

1) визначається послідовність функцій, необхідних для створення програмного продукту;
2) для кожної функції визначаються можливі варіанти реалізації;
3) для кожного варіанта визначаються повні витрати та кількість робочих годин;
4) визначаються технічні параметри, які впливають на якість програмного продукту;
5) проводиться кінцевий розрахунок витрат на розробку;
6) обирається варіант із найкращим співвідношенням між технічним рівнем і вартістю.

4.1. Постановка завдання проектування

Метою проектування є розробка програмного продукту для експериментального порівняння метаевристичних методів оптимізації багатоекстремальних функцій. Програмний продукт має забезпечувати виконання повного циклу дослідження: від задання цільової функції та параметрів алгоритмів до проведення багаторазових запусків, фіксації

результатів, побудови графіків збіжності та порівняльного аналізу ефективності методів.

Основними користувачами програмного продукту можуть бути студенти, дослідники, аналітики та розробники, яким необхідно порівнювати ефективність алгоритмів глобальної оптимізації для складних багатоекстремальних задач. Програмний продукт може застосовуватися як навчальний та дослідницький інструмент для аналізу поведінки алгоритмів на різних типах цільових функцій.

Для досягнення поставленої мети програмний продукт має вирішувати такі ключові інженерні та дослідницькі завдання.

1. Реалізація алгоритмічної бази: програмна реалізація генетичного алгоритму та методу спірального пошуку.
2. Підтримка різних цільових функцій: можливість використання як тестових функцій, так і прикладних задач оптимізації.
3. Проведення серій експериментів: багаторазовий запуск алгоритмів для отримання статистично обґрунтованих результатів.
4. Розрахунок показників ефективності: визначення середнього, найкращого та найгіршого значення цільової функції, стандартного відхилення, часу роботи та кількості обчислень цільової функції.
5. Візуалізація результатів: побудова графіків збіжності алгоритмів і графічне відображення результатів.
6. Гнучкість та масштабованість: можливість зміни параметрів алгоритмів і додавання нових цільових функцій без значної зміни структури програми.

З огляду на це, завданням функціонально-вартісного аналізу у даному розділі є математично та економічно обґрунтувати вибір найбільш раціонального технологічного варіанта реалізації програмного продукту. Необхідно спроектувати систему так, щоб вона забезпечувала достатню швидкість виконання експериментів, зручність аналізу результатів і

можливість подальшого розвитку, але водночас мінімізувала витрати на розробку та підтримку програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка ПП, який дозволяє порівнювати метаевристичні методи оптимізації багатоекстремальних функцій за точністю, стабільністю, швидкістю роботи та характером збіжності. Виходячи з конкретної мети, можна виділити такі основні функції ПП:

F_1 – вибір мови програмування;

F_2 – вибір середовища проведення експериментів та подання результатів;

F_3 – математична реалізація алгоритмів оптимізації.

Кожна з основних функцій може мати декілька варіантів реалізації:

Функція F_1:

а) мова програмування Python;

б) мова програмування C++;

Функція F_2:

а) інтерактивне середовище Jupyter Notebook з використанням Matplotlib;

б) консольний застосунок або окрема desktop-реалізація.

Функція F_3:

а) власна реалізація генетичного алгоритму та методу спірального пошуку з використанням NumPy;

б) використання готових бібліотек оптимізації.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

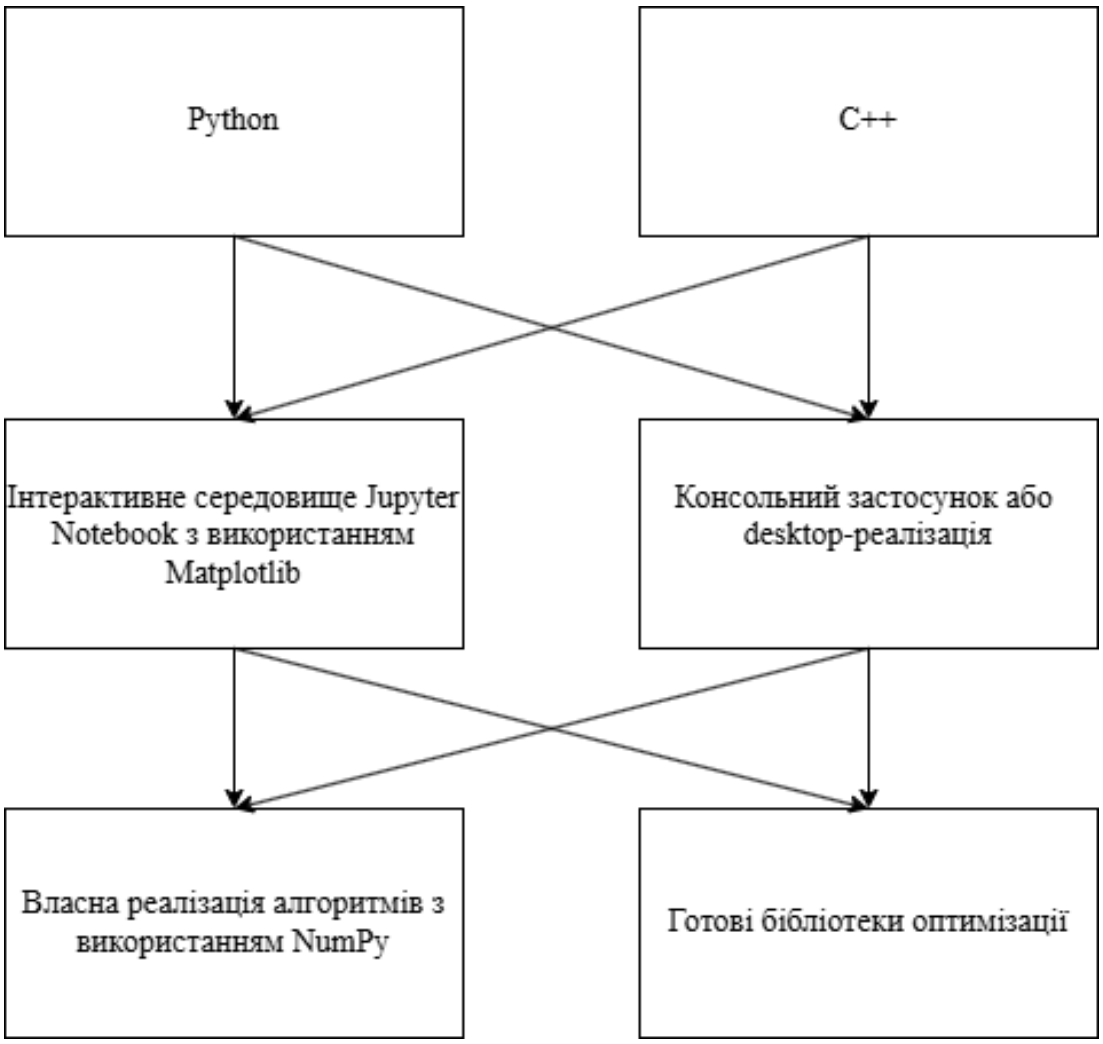


Рисунок 4.1 Морфологічна карта

На основі морфологічної карти будуємо позитивно-негативну матрицю варіантів основних функцій.

Таблиця 4.1 – Позитивно-негативна матриця

Функція	**Варіанти реалізації**	**Позитивні сторони**	**Негативні сторони**
F_1	a	Наявність великої кількості бібліотек, простота написання коду, швидке тестування алгоритмів, зручність роботи з масивами даних.	Менша швидкодія порівняно з C++.
	b	Висока швидкість виконання, ефективне управління пам’яттю, оптимізації обчислень управління пам'яттю комп'ютера.	Складність розробки, більші витрати часу на написання та налагодження коду даних, значні витрати часу на розробку.
F_2	a	Зручність проведення експериментів, швидка побудова графіків, простота зміни параметрів, наочне подання результатів.	Менш зручний вигляд для кінцевого користувача порівняно з повноцінним застосунком.
	b	Можливість створення окремого програмного продукту, звичний формат запуску	Більша трудомісткість розробки, складніша реалізація графіків та інтерфейсу
F_3	a	Повний контроль над логікою алгоритмів, можливість змінювати параметри.	Необхідність самостійної розробки та тестування алгоритмів
	b	Економія часу на розробку, використання вже протестованих алгоритмів	Обмежена гнучкість, складність детального пояснення внутрішньої роботи алгоритмів, залежність від сторонніх бібліотек

На основі проведеного аналізу позитивних та негативних сторін для кожної підфункції було зроблено вибір на користь конкретних технологічних рішень.

Для F_1 перевагу отримала мова програмування Python. Вона має велику кількість спеціалізованих бібліотек для чисельних обчислень, роботи з масивами даних і побудови графіків. Це дозволяє скоротити час розробки та зробити програмний продукт більш зручним для дослідницьких експериментів.

Для F_2 перевагу надано інтерактивному середовищу Jupyter Notebook з використанням бібліотеки Matplotlib. Такий варіант дозволяє швидко змінювати параметри алгоритмів, будувати графіки збіжності та аналізувати результати експериментів без створення складного інтерфейсу користувача.

Для F_3 обрано власну реалізацію генетичного алгоритму та методу спірального пошуку з використанням бібліотеки NumPy. Це забезпечує повний контроль над логікою алгоритмів, дозволяє змінювати параметри та проводити експерименти відповідно до мети дипломної роботи.

Таким чином, будемо розглядати такі варіанти реалізації програмного продукту:

Варіант 1:

$F_1 a - F_2 a - F_3 a$

Варіант 2:

$F_1 b - F_2 b - F_3 a$

Перший варіант передбачає використання мови Python, середовища Jupyter Notebook та власної реалізації генетичного алгоритму і методу спірального пошуку з використанням NumPy.

Другий варіант передбачає використання мови C++, створення окремого консольного або desktop-застосунку та власну реалізацію алгоритмів оптимізації.

Для оцінювання якості розглянутих функцій обрано систему параметрів, описану нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначимо основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня. Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

1) X_1 – швидкість виконання серії експериментів;
2) X_2 – об’єм оперативної пам’яті, необхідний для роботи системи;
3) X_3 – час на розробку та налаштування програмного продукту;
4) X_4 – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів обираються згідно з вимогами до програмного продукту й умовами його експлуатації, як показано у таблиці 4.2.

Таблиця 4.2 Основні параметри програмного продукту

Параметри	Умовне позначення	Значення параметру		
		Гірші	Середні	Кращі
Швидкість виконання серії експериментів, с	X_1	900	600	300
Об’єм оперативної пам’яті, Мб	X_2	2048	1024	512
Час на розробку та налаштування програмного продукту, год	X_3	160	110	80
Потенційний об’єм програмного коду, кількість рядків коду	X_4	1800	1200	800

Далі за даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – 4.5.

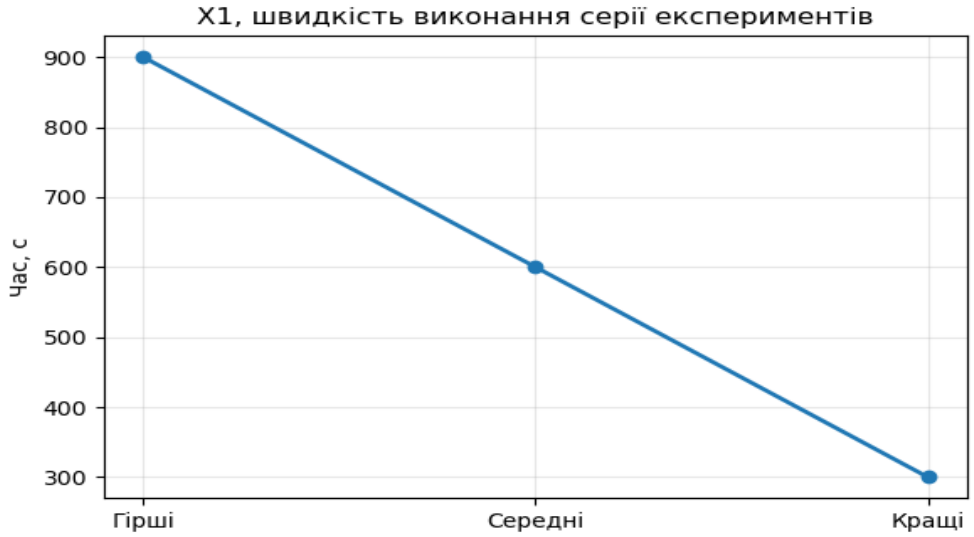


Рисунок 4.2 – X1, швидкість виконання серії експериментів

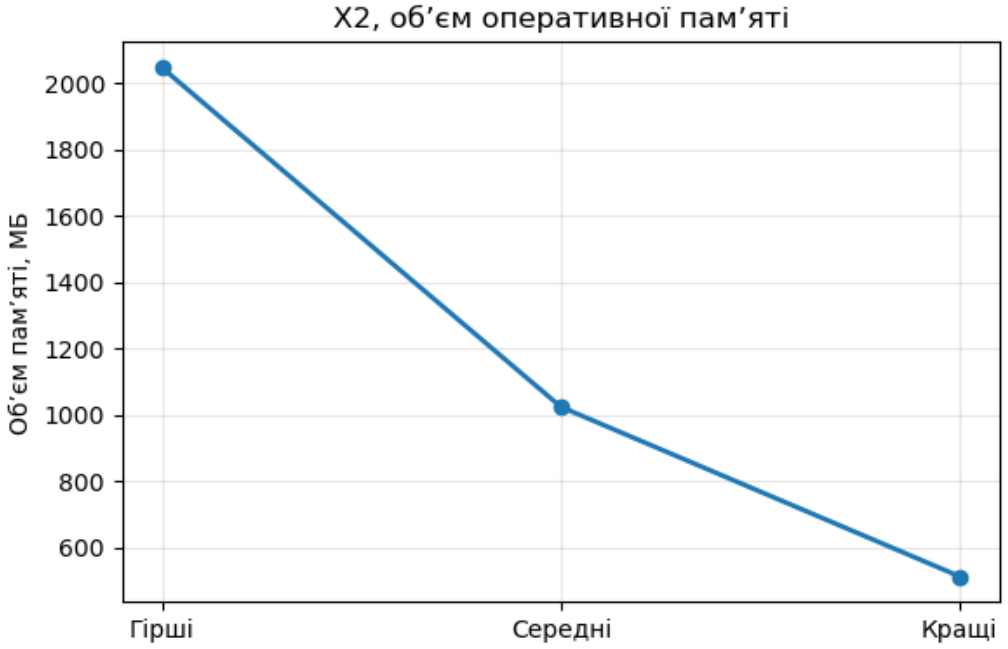


Рисунок 4.3 – X2, об'єм оперативної пам'яті

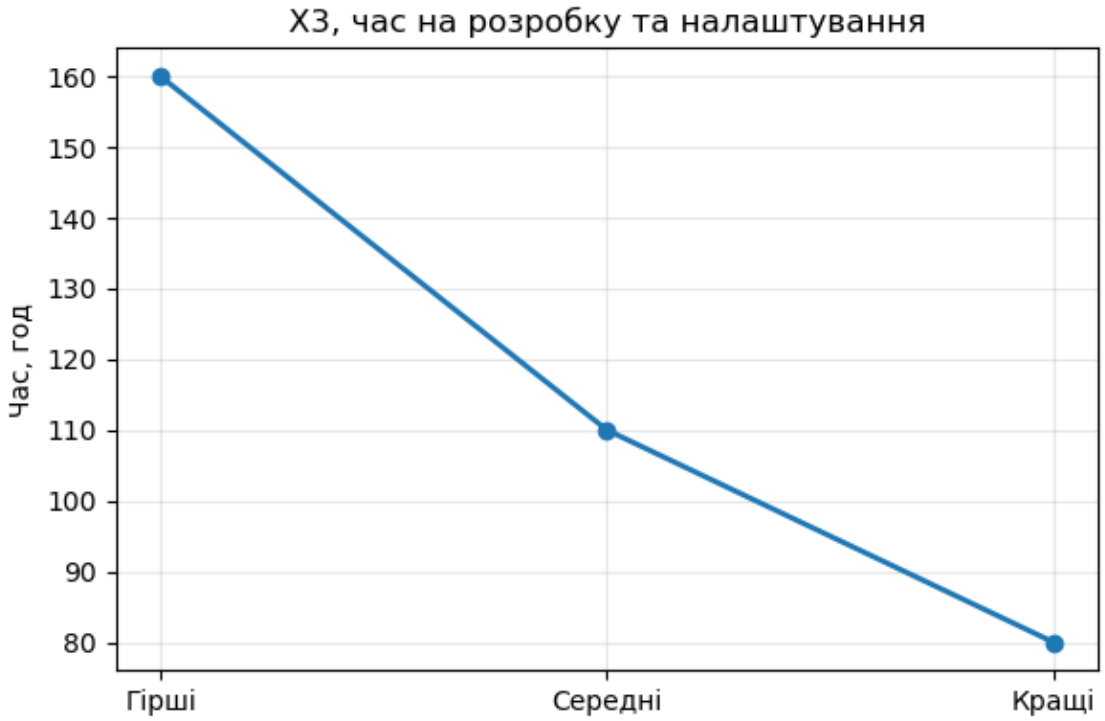


Рисунок 4.4 – X3, час на розробку та налаштування

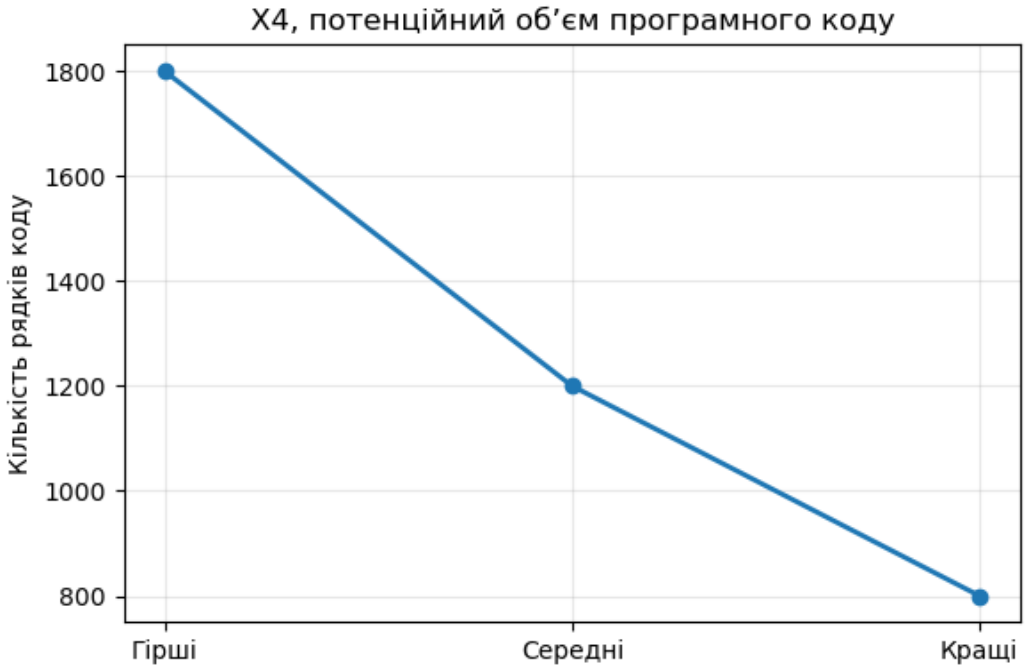


Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.4. Аналіз експертного оцінювання параметрів

Важливість кожного параметра в загальній кількості розглянутих під час оцінювання параметрів знаходять за методом попарного порівняння. Оцінювання проводить експертна комісія із семи осіб.

Щоб знайти коефіцієнт важливості, передбачається:

1) визначити рівень важливості параметра через присвоєння різних рангів;
2) перевірити придатність експертних оцінок у подальшому використанні;
3) визначити оцінки попарного пріоритету параметрів;
4) обробити результати і знайти коефіцієнт важливості.

Після детального обговорення й аналізу кожний експерт оцінює рівень важливості параметрів, присвоюючи їм ранги. Результати експертного ранжування наведено у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	**Назва параметра**	**Одиниці вимірювання**	**Ранг параметра за оцінкою експерта**							**Сума рангів, R_i**	**Відхилення, Δ_i,**	**Квадр. відхилення, $(\Delta)^2$**
			1	**2**	**3**	**4**	**5**	**6**	**7**			
Xl	Швидкість виконання серії експериментів	с	1	1	1	2	1	1	2	9	-8,5	72,25
X2	Об’єм оперативної пам’яті	МБ	2	2	3	1	2	3	1	14	-3,5	12,25
X3	Час на розробку та налаштування	год	3	4	2	3	3	2	4	21	3,5	12,25
X4	Потенційний об’єм коду	кількість рядків коду	4	3	4	4	4	4	3	26	8,5	72,25
Всього	$n \cdot (n + 1) / 2$	–	10	10	10	10	10	10	10	70	0	169

Для перевірки степені достовірності експертних оцінок визначимо такі параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \Sigma_j^N r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – кількість експертів; n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17.5$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень за всіма параметрами має дорівнювати 0;

г) загальна сума квадратів відхилення

$$S = \sum_{i=1}^{n} \Delta^2{}_i = 169$$

Обчислимо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)},$$

$$W = \frac{12 * 169}{\left(7^2 * (4^3 - 4)\right)} = 0.69 > W_k = 0.67$$

Ранжирування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	**Експерти**							**Кінцева оцінка**	**Числове значення**
	1	**2**	**3**	**4**	**5**	**6**	**7**		
X1 та X2	>	>	>	<	>	>	<	>	1,5
X1 та X3	>	>	>	>	>	>	>	>	1,5
X1 та X4	>	>	>	>	>	>	>	>	1,5
X2 та X3	>	>	<	>	>	<	>	>	1.5
X2 та X4	>	>	>	>	>	>	>	>	1,5
X3 та X4	>	<	>	>	>	>	<	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначають за формулою:

$$a_{ij} = 1{,}5 \text{ при } \mathrm{X}_i > \mathrm{X}_j, 1{,}0 \text{ при } \mathrm{X}_i = \mathrm{X}_j, 0{,}5 \text{ при } \mathrm{X}_i < \mathrm{X}_j.$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості K_{B_i} за такою формулою:

$$K_{Bi} = \frac{b_i}{\sum_{i=1}^{n} b_i}$$

$$b_i = \sum_{i=1}^{N} a_{ij}$$

Відносні оцінки розраховують декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2 %). На другому і наступних кроках відносні оцінки розраховують за такою формулою:

$$K_{Bi} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри	**Параметри**				**Перша ітерація**		**Друга ітерація**		**Третя ітерація**	
	X1	X2	X3	*X4*	b_i	$К_{вi}$	b_i^1	$К_{вi}^1$	b_i^2	$К_{вi}^2$
X1	1,0	1,5	1,5	1,5	5,5	0,3438	21,25	0,3602	77,875	0,3606
X2	0,5	1,0	1,5	1,5	4,5	0,2813	16,25	0,2754	59,125	0,2737
X3	0,5	0,5	1,0	1,5	3,5	0,2188	12,25	0,2076	44,875	0,2078
X4	0,5	0,5	0,5	1,0	2,5	0,1563	9,25	0,1568	34,125	0,1580
Всього:					16	1	59	1	216	1

4.5 Аналіз варіантів реалізації функцій

На основі порівняльного аналізу варіантів реалізації функцій за їх перевагами та недоліками і коефіцієнтами вагомості параметрів (табл. 4.1 та 4.5) можна виключити з функції F2 – варіант (b).

Варіанти, які залишилися:

1) F1a + F2a + F3a;
2) F1b + F2a + F3a;
3) F1a + F2a + F3b;
4) F1b + F2a + F3b.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується за формулою:

$$K_k(j) = \sum_{i=1}^{n} K_{Bi,j} B_{ij},$$

де n − кількість параметрів; K_{Bi} − коефіцієнт вагомості i–го параметра; B_i − оцінка i–го параметра в балах.

Таблиця 4.6 Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основна функція	Варіант реалізації	Абсолютне значення параметру	Бальна оцінка параметру	Коефіцієнт вагомості параметру	Коефіцієнт якості
F1	a	90	9	0,2078	1,87
	b	140	3,5	0,2078	0,73
F2	a	500	7	0,3606	2,52
F3	a	800	10	0,1580	1,58
	b	1200	6,5	0,1580	1,03

За даними табл. 4.6, та формулою

$$K_k = K_{TY}[F_{1k}] + K_{TY}[F_{2k}] + \cdots + K_{TY}[F_{zk}],$$

знайдемо, показники рівня якості кожного із варіантів ПП.

K_{k1} = 1,87 + 2,52+ 1,58 = 5,97;

K_{k2} = 0,73 + 2,52+ 1,58 = 4,83;

K_{k3} = 1,87 + 2,52 + 1,03 = 5,42;

K_{k4} = 0,73 + 2,52 + 1,03 = 4,28.

Отже, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки програмного продукту

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання.

1. розробка проекту програмного продукту;
2. розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується нормативно-довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_0 = T_\text{Р} \cdot K_\text{П} \cdot K_\text{СК} \cdot K_\text{М} \cdot K_\text{СТ} \cdot K_\text{СТМ},$$

де $T_\text{р}$ – трудомісткість розробки ПП; $K_\text{п}$ – поправочний коефіцієнт; $K_\text{СК}$ – коефіцієнт на складність вхідної інформації; $K_\text{М}$ – коефіцієнт рівня мови програмування; $K_\text{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм; $K_\text{СТМ}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру ступеня новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_\text{р} = 90$ людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_\text{п} = 1{,}7$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівні 1: $K_\text{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, враховуємо це за допомогою коефіцієнта $K_\text{СТ} = 0{,}8$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 \cdot 1{,}7 \cdot 0{,}8 = 122{,}4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для другого завдання (використовується алгоритм третьої групи складності, ступінь новизни Б), тобто $T_\text{р} = 19$, людино-днів, $K_\text{п} = 1{,}5$, $K_\text{СК} = 1$.

Для варіанту I (мова програмування Python) коефіцієнт рівня мови програмування $K_{\text{М}} = 0{,}8$, оскільки Python є мовою програмування високого рівня з розвиненою екосистемою бібліотек для аналізу даних (NumPy, Pandas), що суттєво скорочує обсяг реалізації:

$$T_2 = 19 \cdot 1{,}5 \cdot 0{,}8 = 22{,}8 \text{ людино-днів.}$$

Для варіанту II (мова програмування C++) коефіцієнт рівня мови програмування $K_{\text{М}} = 1{,}15$, оскільки C++ є мовою нижчого рівня, що потребує ручного управління пам'яттю та значно більших зусиль на реалізацію аналогічного функціоналу:

$$T_2 = 19 \cdot 1{,}5 \cdot 1{,}15 = 32{,}775 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_{\text{I}} = (122{,}4 + 22{,}8) \cdot 8 = 1161{,}1 \text{ людино-годин}$$

$$T_{\text{II}} = (122{,}4 + 32{,}775) \cdot 8 = 1241{,}1 \text{ людино-годин}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь один програміст з окладом 28 000 грн., один аналітик в області даних з окладом 32 000 грн. Визначимо середню зарплату за годину за формулою:

$$\text{С}_{\text{ч}} = \frac{M}{T_m \cdot t} \text{ грн.,}$$

де М – місячний оклад працівників; T_m– кількість робочих днів на місяць; t– кількість робочих годин в день.

$$\text{С}_{\text{ч}} = \frac{28000 + 32000}{2 \cdot 21 \cdot 8} = 178{,}57 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою:

$$\text{С}_{\text{ЗП}} = \text{С}_{\text{ч}} \cdot T_i \cdot K_{\text{д}},$$

де $\text{С}_{\text{ч}}$ – величина погодинної оплати праці програміста; T_i– трудомісткість відповідного завдання; $K_{\text{д}}$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 178{,}57 \cdot 1161{,}1 \cdot 1{,}2 = 248\,805{,}15$ грн.

II. $С_{ЗП} = 178{,}57 \cdot 1241{,}1 \cdot 1{,}2 = 265\,947{,}87$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{від} = С_{ЗП} \cdot 0{,}22 = 54\,737{,}13$ грн.

II. $С_{від} = С_{ЗП} \cdot 0{,}22 = 58\,508{,}34$ грн.

Тепер визначимо витрати на оплату однієї машино-години ($С_М$).

Так як одна ЕОМ обслуговує одного програміста з окладом 28 000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$C_r = 12 \cdot M \cdot K_з = 12 \cdot 28000 \cdot 0{,}2 = 67\,200 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = C_r \cdot (1 + K_з) = 67\,200 \cdot (1 + 0{,}2) = 80\,640 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{від} = С_{ЗП} \cdot 0{,}22 = 80\,640 \cdot 0{,}22 = 17\,740{,}80 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 20 000 грн.

$$С_А = K_{ТМ} \cdot K_А \cdot Ц_{пр} = 1{,}5 \cdot 0{,}25 \cdot 20\,000 = 7\,500 \text{ грн.},$$

де $K_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $K_А$– річна норма амортизації; $Ц_{пр}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = K_{ТМ} \cdot Ц_{пр} \cdot K_Р = 1{,}5 \cdot 20\,000 \cdot 0{,}06 = 1\,800 \text{ грн.},$$

де $K_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{ЕФ} = \left(Д_к - Д_в - Д_с - Д_р\right) \cdot t \cdot K_в = (365 - 104 - 12 - 14) \cdot 8 \cdot 0{,}9 =$$

$$= 1\,692 \text{ години},$$

де: $Д_к$– календарна кількість днів у році; $Д_в$, $Д_с$– відповідно кількість вихідних та святкових днів; $Д_р$– кількість днів планових ремонтів устаткування; t– кількість робочих годин в день; $K_в$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = T_{ЕФ} \cdot N_с \cdot K_з \cdot Ц_{ен} = 1692 \cdot 0{,}2 \cdot 0{,}2 \cdot 13{,}64 = 923{,}16 \text{ грн.},$$

де $N_с$ – середньо-споживча потужність приладу; $K_з$– коефіцієнт зайнятості приладу; $Ц_{ен}$– тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{пр} \cdot 0{,}67 = 20\,000 \cdot 0{,}67 = 13\,400 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{від} + С_А + С_Р + С_{ЕЛ} + С_Н,$$

$$С_{ЕКС} = 80\,640 + 17\,740{,}80 + 7\,500 + 1\,800 + 923{,}16 + 13\,400 =$$

$$= 122\,003{,}96 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М-Г} = \frac{С_{екс}}{T_{еф}} = \frac{122\,003{,}96}{1\,692} = 72{,}11 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов'язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot T,$$

$$\text{I. } С_М = 72{,}11 \cdot 1161{,}1 = 83\,726{,}92 \text{ грн.}$$

$$\text{II. } С_М = 72{,}11 \cdot 1241{,}1 = 89\,495{,}72 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0{,}67,$$

$$\text{I. } С_Н = 248\,805{,}15 \cdot 0{,}67 = 166\,699{,}45 \text{ грн.}$$

$$\text{II. } С_Н = 265\,947{,}87 \cdot 0{,}67 = 178\,185{,}07 \text{ грн.}$$

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{від} + С_{М} + С_{Н},$$

$$\text{I. } С_{ПП} = 248\,805{,}15 + 54\,737{,}13 + 83\,726{,}92 + 166\,699{,}45 =$$

$$= 553\,968{,}65 \text{ грн.}$$

$$\text{II. } С_{ПП} = 265\,947{,}87 + 58\,508{,}34 + 89\,495{,}72 + 178\,185{,}07 =$$

$$= 592\,137 \text{ грн.}$$

4.7. Вибір найкращого варіанта програмного продукту за техніко-економічним рівнем

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{\text{ТЕР}j} = K_{\text{К}j} / С_{\text{Ф}j},$$

$$K_{\text{ТЕР1}} = \frac{5{,}97}{553\,968{,}65} = 1{,}08 \cdot 10^{-5}$$

$$K_{\text{ТЕР2}} = \frac{4{,}83}{592\,137} = 8{,}16 \cdot 10^{-6}$$

Отже, найбільш ефективним є перший варіант реалізації програмного продукту, оскільки він має більше значення коефіцієнта техніко-економічного рівня: $K_{\text{ТЕР1}} = 1{,}08 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного продукту що розробляється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного продукту оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $K_{\text{ТЕР}} = 1{,}08 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

1) вибір мови програмування – Python;
2) інтерфейс користувача – Jupyter Notebook;

3) математична реалізація алгоритмів оптимізації – власна реалізація алгоритмів пошуку з використанням NumPy.

Цей варіант забезпечує найкраще співвідношення між технічними характеристиками, вартістю розробки та відповідністю меті роботи – порівнянню метаевристичних методів оптимізації багатоекстремальних функцій.

4.8. Висновки до розділу 4

У четвертому розділі проведено повний функціонально-вартісний аналіз розробленого програмного продукту для порівняння метаевристичних методів оптимізації багатоекстремальних функцій.

Визначено та обґрунтовано три основні функції програмного продукту: вибір мови програмування, вибір середовища проведення експериментів та математична реалізація алгоритмів оптимізації. Для кожної функції сформовано по два варіанти реалізації з подальшим їх порівнянням.

Визначено чотири параметри, що характеризують програмний продукт: швидкість виконання серії експериментів, об’єм оперативної пам’яті, час на розробку та налаштування, а також потенційний об’єм програмного коду. За результатами експертного оцінювання з коефіцієнтом конкордації $W = 0.69$ підтверджено узгодженість думок експертів та розраховано коефіцієнти вагомості параметрів.

На основі порівняльного аналізу варіантів реалізації функцій визначено коефіцієнти техніко-економічного рівня: $K_{\text{ТЕР1}} = 1{,}08 \cdot 10^{-5}$ та $K_{\text{ТЕР2}} = 8{,}16 \cdot 10^{-6}$. Проведено економічний аналіз обох варіантів розробки, що включав розрахунок трудомісткості, витрат на заробітну плату розробників, відрахувань на соціальний внесок, витрат на машинний час та накладних

витрат. Вартість розробки за першим варіантом становить $C_{ПП1} = 553\,968{,}65$, за другим – $C_{ПП2} = 592\,137$.

За результатами аналізу обрано перший варіант реалізації програмного продукту: мова програмування Python, середовище Jupyter Notebook та модульна власна реалізація алгоритмів. Обраний варіант забезпечує найвищий техніко-економічний рівень якості при менших витратах на розробку.

ВИСНОВКИ

У дипломній роботі було досліджено задачу оптимізації багатоекстремальних функцій та проведено порівняння двох метаевристичних методів глобальної оптимізації: генетичного алгоритму та методу спірального пошуку.

У першому розділі було розглянуто теоретичні основи глобальної оптимізації багатоекстремальних функцій. Показано, що такі задачі характеризуються наявністю багатьох локальних екстремумів, складною структурою простору пошуку та ризиком передчасної збіжності до локального мінімуму. Також було проаналізовано обмеження традиційних детермінованих і градієнтних методів, які суттєво залежать від початкової точки та не завжди забезпечують знаходження глобального оптимуму.

У другому розділі було розглянуто математичні моделі генетичного алгоритму та методу спірального пошуку. Описано основні етапи роботи генетичного алгоритму, зокрема формування популяції, селекцію, кросовер і мутацію. Також розглянуто математичну модель методу спірального пошуку, яка базується на спіральному русі точок-кандидатів навколо поточного найкращого розв’язку. Для експериментального дослідження було обрано функції Швефеля та Міхалевича, а також прикладну задачу розміщення складів.

У третьому розділі було виконано програмну реалізацію алгоритмів мовою Python та проведено серію обчислювальних експериментів. Результати показали, що традиційні методи оптимізації, зокрема градієнтний спуск і метод Ньютона, мають обмежену ефективність для багатоекстремальних функцій, оскільки можуть збігатися до локальних мінімумів. Це підтвердило доцільність використання метаевристичних підходів.

Під час дослідження генетичного алгоритму встановлено, що його ефективність залежить від способу селекції, коефіцієнта кросоверу та коефіцієнта мутації. Найкращі результати для складної функції Швефеля були отримані при використанні рангового відбору та помірного значення мутації. Для функції Міхалевича генетичний алгоритм продемонстрував стабільні результати та високу точність.

Метод спірального пошуку також показав високу ефективність при оптимізації багатоекстремальних функцій. Його результати суттєво залежали від кількості точок-кандидатів, кута обертання та коефіцієнта наближення. Збільшення кількості точок покращувало стабільність пошуку, а правильний вибір коефіцієнта наближення дозволяв досягти балансу між глобальним дослідженням простору та локальним уточненням розв’язку.

У задачі розміщення складів обидва алгоритми продемонстрували можливість застосування до прикладних задач логістичного типу. Метод спірального пошуку показав менший час роботи та стабільні результати, тоді як генетичний алгоритм забезпечив гнучкий механізм глобального пошуку завдяки роботі з популяцією розв’язків.

У четвертому розділі було проведено функціонально-вартісний аналіз програмного продукту. За результатами аналізу найбільш доцільним варіантом реалізації обрано програмний продукт мовою Python у середовищі Jupyter Notebook з модульною власною реалізацією алгоритмів. Такий варіант забезпечує зручність проведення експериментів, наочність результатів і прийнятну трудомісткість розробки.

Отже, поставлену мету дипломної роботи було досягнуто. Було досліджено особливості задач глобальної оптимізації багатоекстремальних функцій, реалізовано генетичний алгоритм і метод спірального пошуку, а також проведено їх експериментальне порівняння на тестових функціях і прикладній задачі розміщення складів. Отримані результати підтверджують

доцільність використання метаевристичних методів для розв’язання складних задач глобальної оптимізації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Савченко І. О. Еволюційні методи оптимізації. Комп’ютерний практикум : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2025. 47 с.
2. Nocedal J., Wright S. J. Numerical Optimization. 2nd ed. New York : Springer, 2006. 664 p.
3. Jamil M., Yang X.-S. A Literature Survey of Benchmark Functions for Global Optimization Problems. *International Journal of Mathematical Modelling and Numerical Optimisation*. 2013. Vol. 4, No. 2. P. 150–194.
4. Boyd S., Vandenberghe L. Convex Optimization. Cambridge : Cambridge University Press, 2004. 716 p.
5. Talbi E.-G. Metaheuristics: From Design to Implementation. Hoboken : John Wiley & Sons, 2009. 624 p.
6. Yang X.-S. Nature-Inspired Metaheuristic Algorithms. 2nd ed. Frome : Luniver Press, 2010. 160 p.
7. Drezner Z., Hamacher H. W. Facility Location: Applications and Theory. Berlin : Springer, 2002. 457 p.
8. Daskin M. S. Network and Discrete Location: Models, Algorithms, and Applications. 2nd ed. Hoboken : John Wiley & Sons, 2013. 544 p.
9. Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs. 3rd rev. and extended ed. Berlin : Springer, 1996. 387 p.
10. Eiben A. E., Smith J. E. Introduction to Evolutionary Computing. 2nd ed. Berlin : Springer, 2015. 287 p.
11. Haupt R. L., Haupt S. E. Practical Genetic Algorithms. 2nd ed. Hoboken : John Wiley & Sons, 2004. 288 p.

12. Tamura K., Yasuda K. Spiral Dynamics Inspired Optimization. *Journal of Advanced Computational Intelligence and Intelligent Informatics*. 2011. Vol. 15, No. 8. P. 1116–1122.
13. Vanaret C., Gotteland J.-B., Durand N., Alliot J.-M. Certified Global Minima for a Benchmark of Difficult Optimization Problems. 2020. URL: https://arxiv.org/abs/2003.09867 (date of access: 21.05.2026).
14. Harris C. R., Millman K. J., van der Walt S. J., Gommers R., Virtanen P., Cournapeau D. та ін. *Array programming with NumPy*. Nature. 2020. Vol. 585. P. 357–362.
15. Hunter J. D. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*. 2007. Vol. 9, No. 3. P. 90–95.

ДОДАТОК А. ДОДАТКОВІ ІЛЮСТРАТИВНІ МАТЕРІАЛИ

Таблиця А.1 – Усі вхідні дані задачі розміщення складів

№ міста	Координата a_i	Координата b_i	Попит D_i
1	10	20	120
2	20	80	80
3	30	50	150
4	40	30	100
5	50	90	90
6	60	10	110
7	70	70	130
8	80	40	160
9	90	20	70
10	25	25	140
11	35	75	95
12	55	55	125
13	65	85	115
14	75	15	105
15	85	60	135
16	15	60	85
17	45	15	145
18	95	85	155
19	12	35	75
20	38	92	165
21	22	45	90
22	48	68	118

Продовження таблиці А.1

23	58	32	132
24	68	12	101
25	88	48	147
26	18	88	112
27	28	12	98
28	42	42	126
29	52	78	139
30	62	58	108
31	72	92	122
32	82	28	152
33	92	10	73
34	8	75	84
35	32	18	116
36	46	85	129
37	56	22	111
38	66	66	143
39	76	52	134
40	86	88	157
41	14	14	93
42	24	64	121
43	34	34	109
44	44	74	136
45	54	44	114
46	64	94	149
47	74	34	127
48	84	74	138
49	94	54	119
50	50	50	160

Інші 5 експериментів для розміру популяції/кількісті точок – 100.

Експеримент 2. Розмір популяції/кількість точок – 100, кількість міст – 10, кількість складів – 3. Координати складів для найкращого запуску ГА:
[[51.90734863, 83.20465088]
[79.98352051, 39.85748291]
[25.02746582, 25.2532959]] (рис А.1).

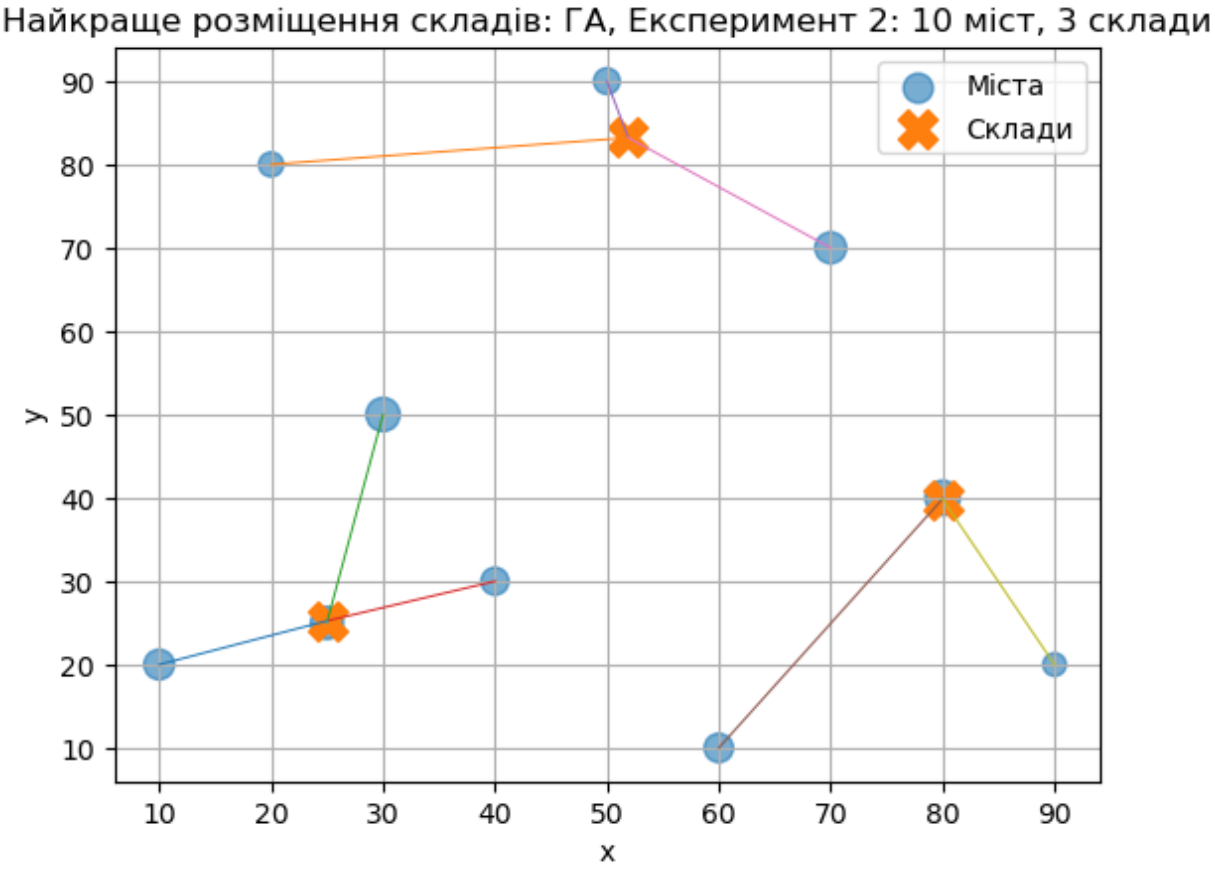


Рисунок А.1 ГА (Експеримент 2)

Координати складів для найкращого запуску СП:

[[52.02021381, 83.56505651]

[25.02165942, 25.33738459]

[79.99434319, 39.95621744]] (рис А.2).

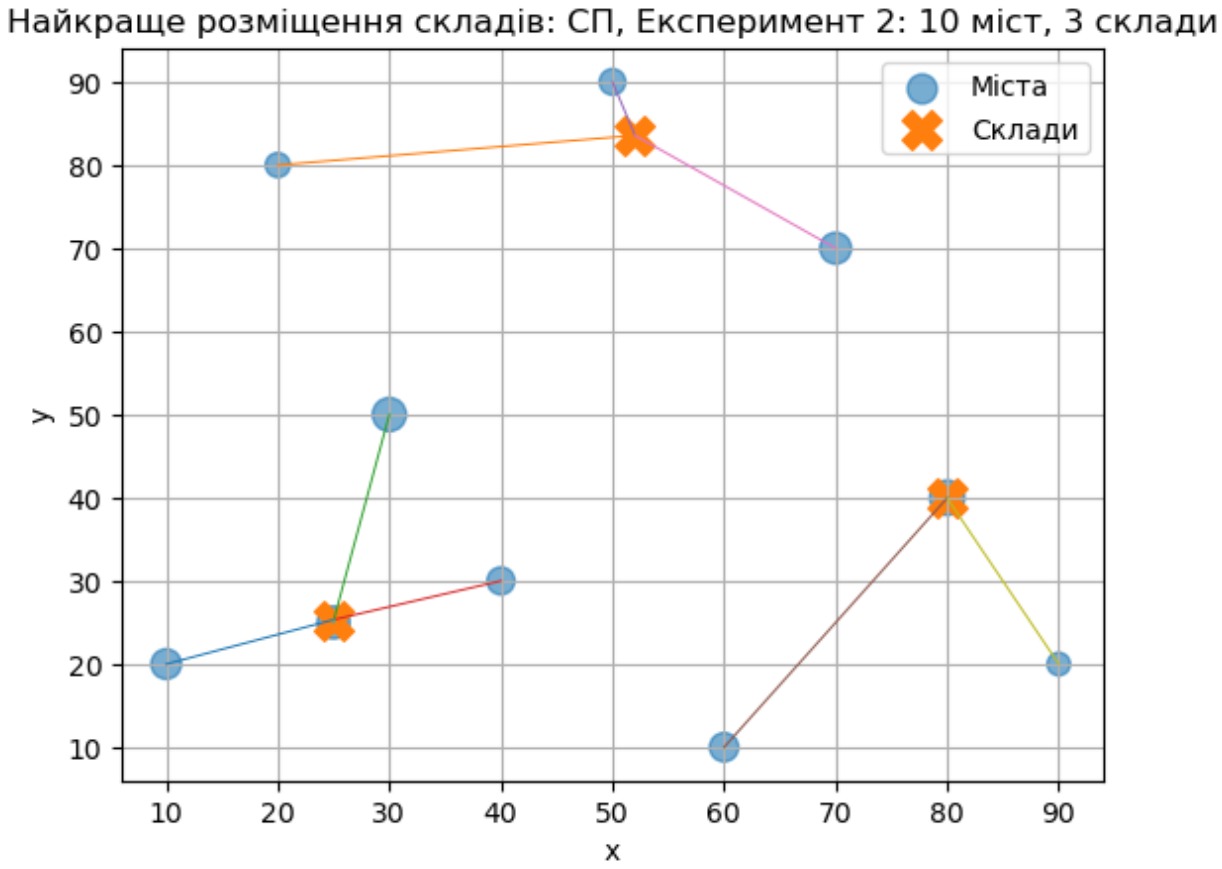


Рисунок А.2 СП (Експеримент 2)

Експеримент 3. Розмір популяції/кількість точок – 100, кількість міст – 20, кількість складів – 2. Координати складів для найкращого запуску ГА:
[[65.91491699, 71.62017822]
[36.80267334, 27.65197754]] (рис А.3).

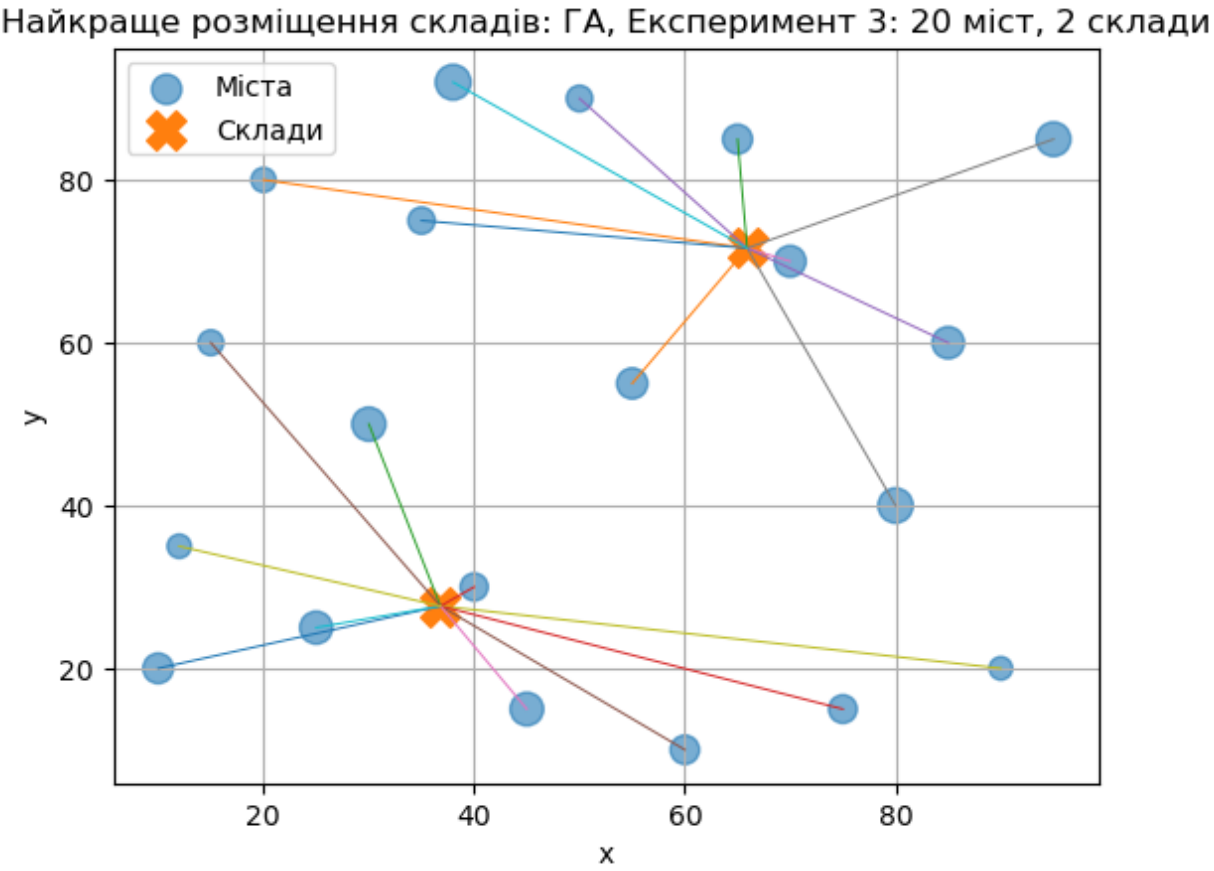


Рисунок А.3 ГА (Експеримент 3)

Координати складів для найкращого запуску СП:

[[36.85320403, 27.6406451]

[65.92119713, 71.64638851]] (рис А.4).

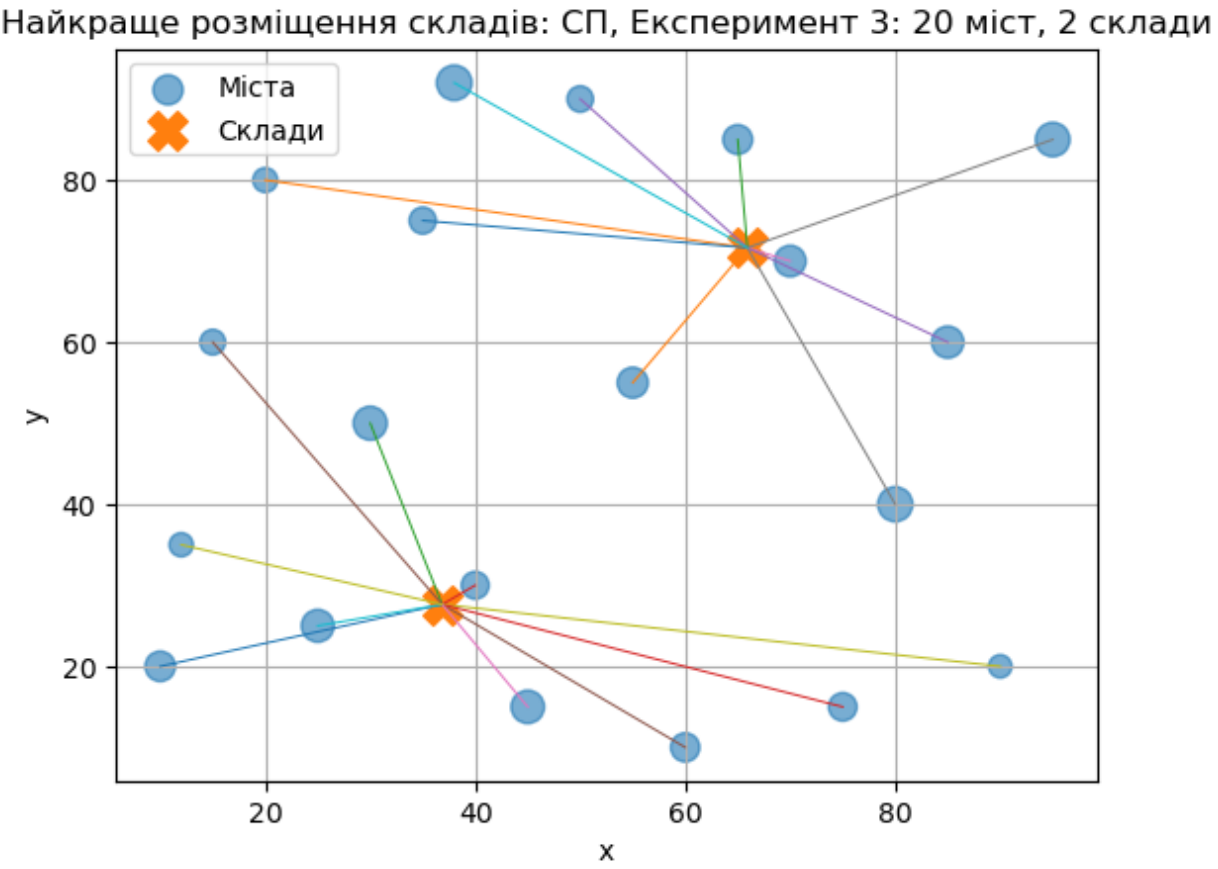


Рисунок А.4 СП (Експеримент 3)

Експеримент 4. Розмір популяції/кількість точок – 100, кількість міст – 20, кількість складів – 3. Координати складів для найкращого запуску ГА:
[[40.0970459, 89.10827637]
 [79.20837402, 55.3314209]
 [28.36303711, 27.7130127]] (рис А.5).

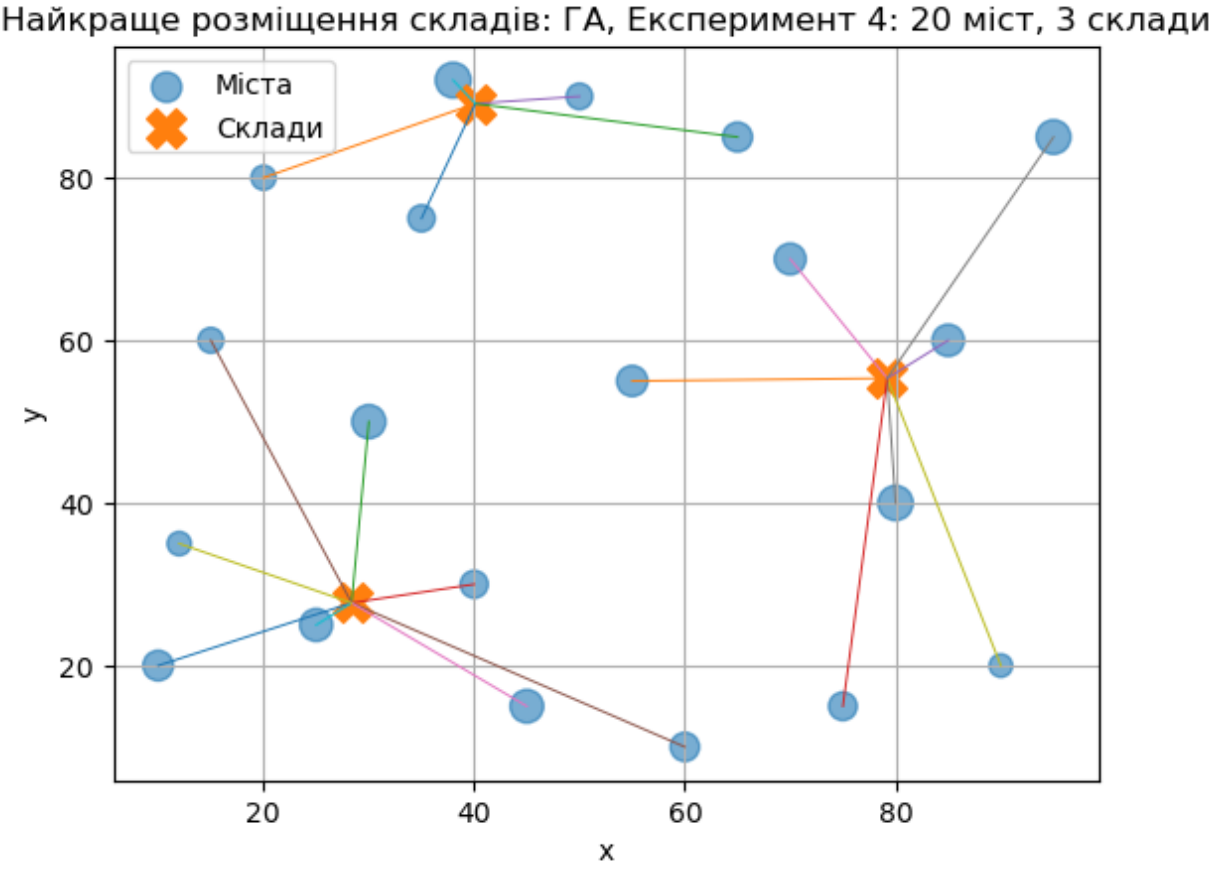


Рисунок А.5 ГА (Експеримент 4)

Координати складів для найкращого запуску СП:

[[40.12006153, 89.12449151]

[28.37715629, 27.79082633]

[79.18518348, 55.27939462]] (рис А.6).

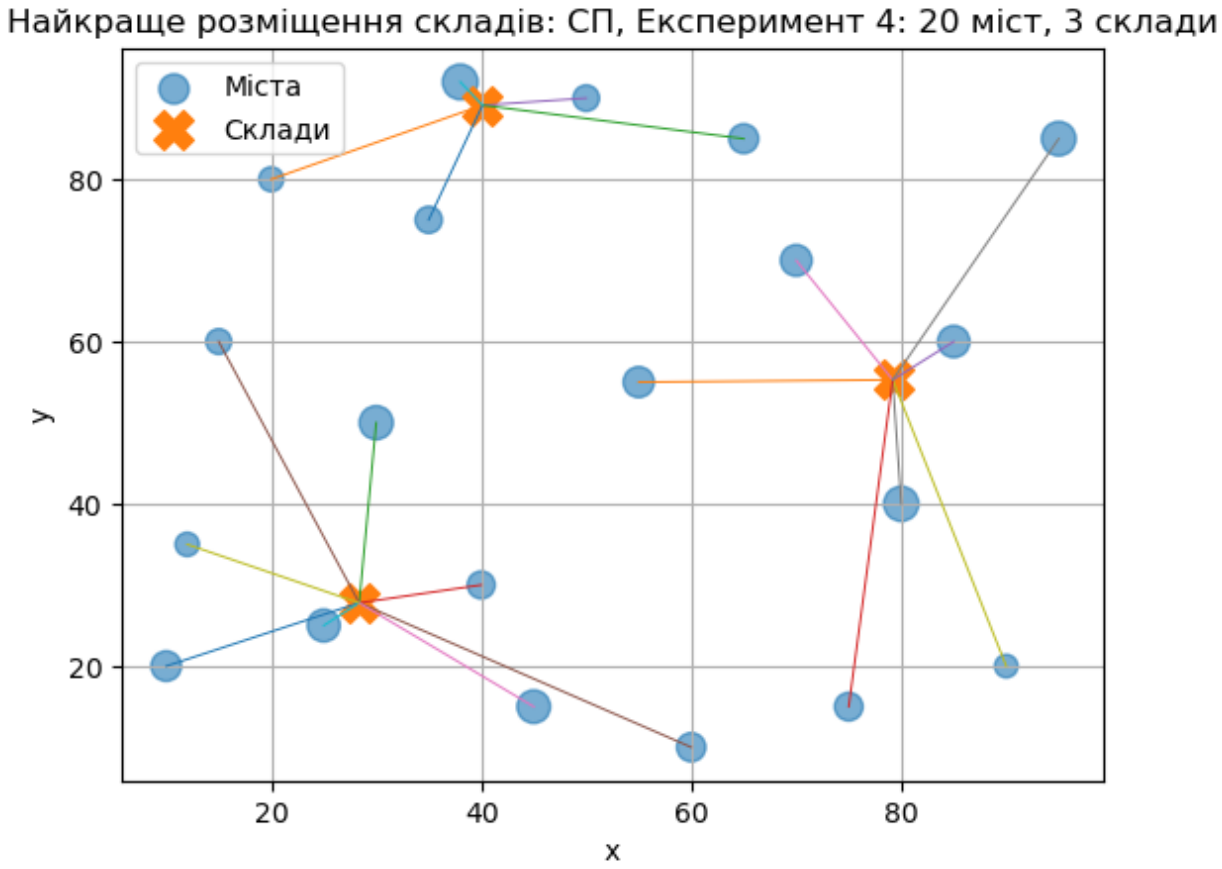


Рисунок А.6 СП (Експеримент 4)

Експеримент 5. Розмір популяції/кількість точок – 100, кількість міст – 50, кількість складів – 3. Координати складів для найкращого запуску ГА:
[[78.66363525, 35.97259521]
[34.00878906, 34.00726318]
[54.60357666, 78.56445312]] (рис А.7).

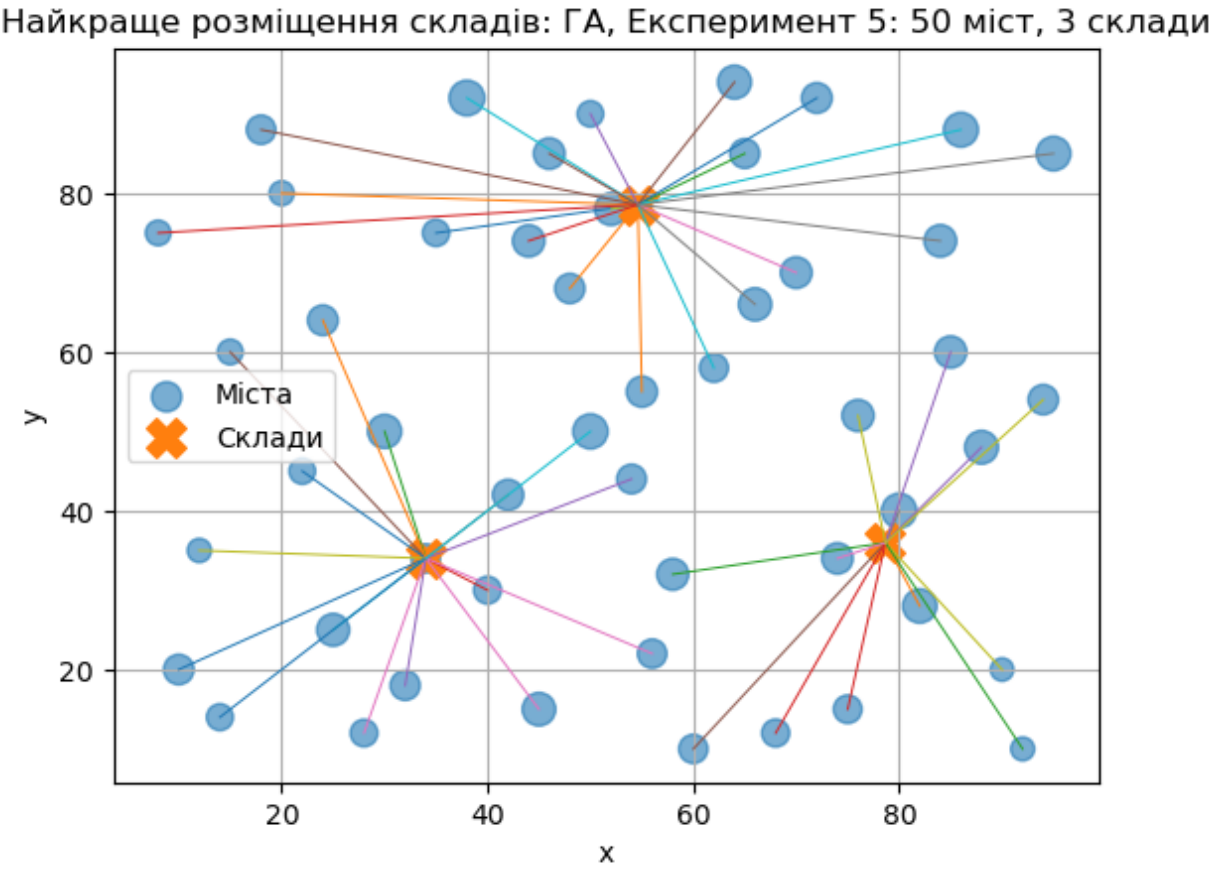


Рисунок А.7 ГА (Експеримент 5)

Координати складів для найкращого запуску СП:

[[34.00673456, 33.99865592]

[78.68125593, 35.93840001]

[54.84345682, 78.6266507]] (рис А.8).

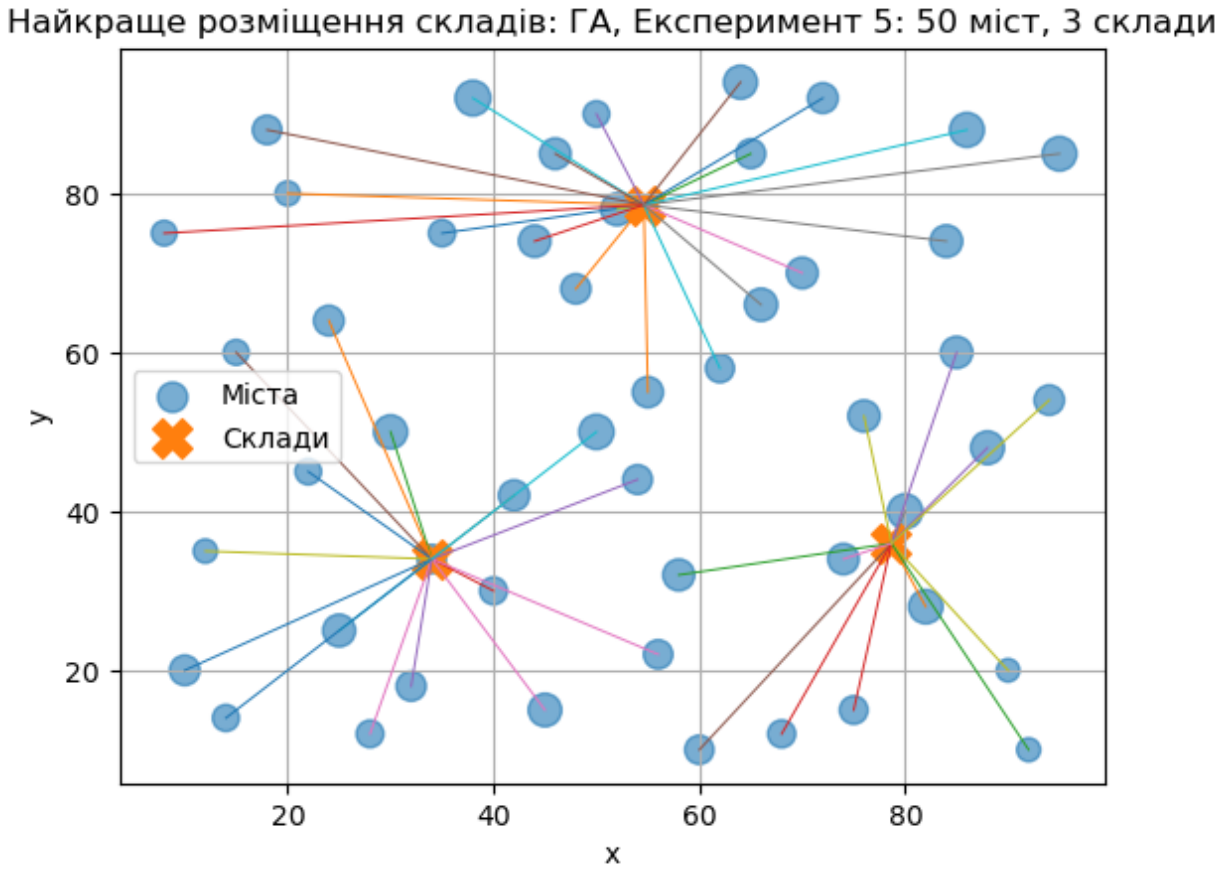


Рисунок А.8 СП (Експеримент 5)

Експеримент 6. Розмір популяції/кількість точок – 100, кількість міст – 50, кількість складів – 5. Координати складів для найкращого запуску ГА:
[[76.29699707, 80.02929688]
[25.70037842, 23.88305664]
[38.62762451, 78.23181152]
[76.47857666, 32.53173828]
[49.8916626, 49.79705811]] (рис А.9).

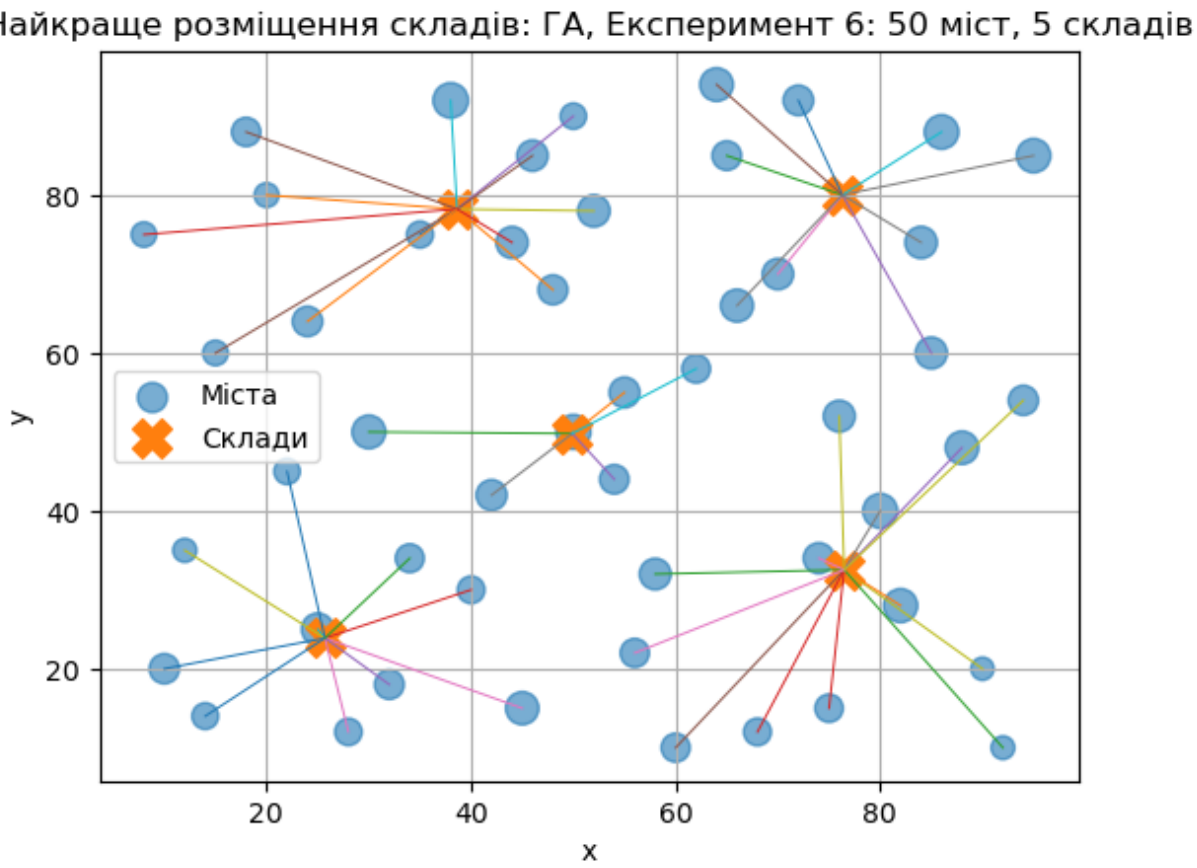


Рисунок А.9 ГА (Експеримент 6)

Координати складів для найкращого запуску СП:

[[38.55999307 78.19621449]

[76.06567151 32.57713753]

[50.01463848 50.01284494]

[26.46803317 23.97328804]

[77.06036346 79.4104191]] (рис А.10).

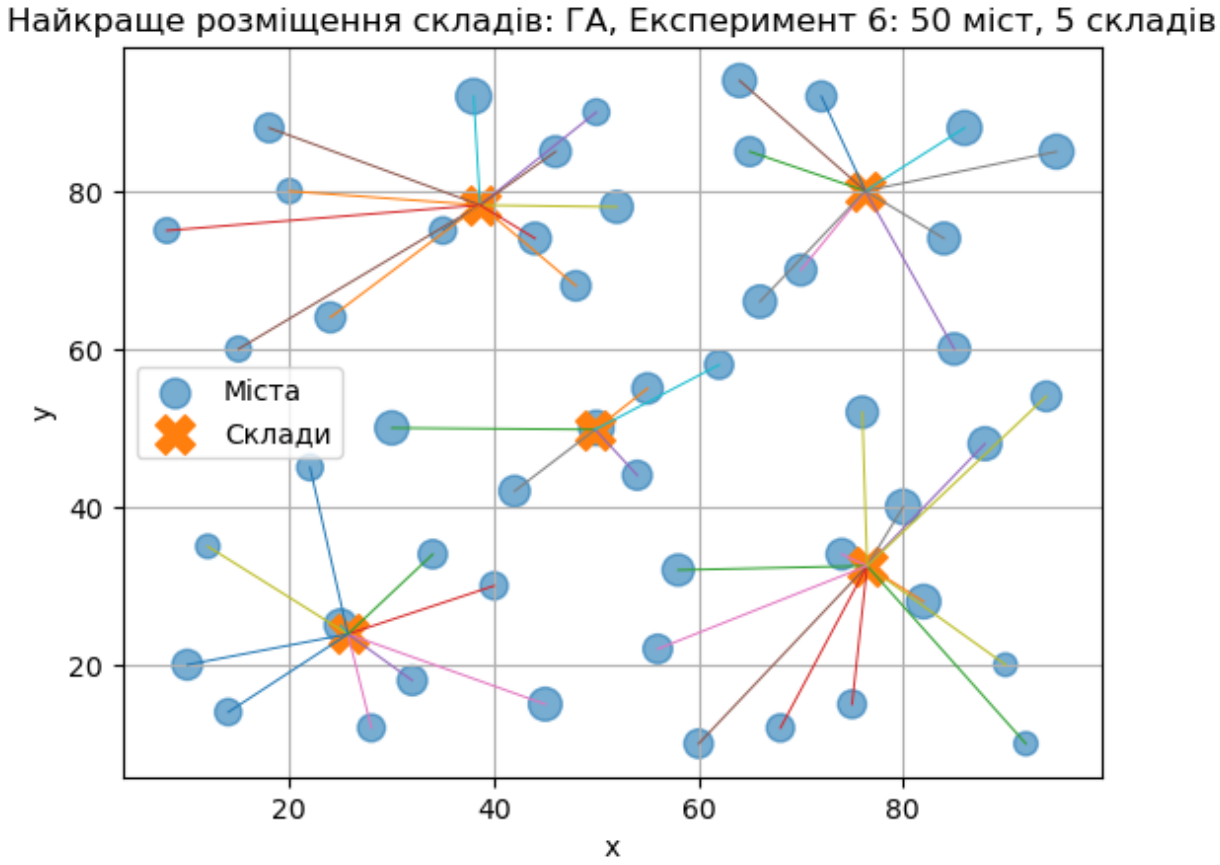


Рисунок А.10 СП (Експеримент 6)

Інші 6 експериментів для розміру популяції/кількісті точок – 200.

Експеримент 1. Розмір популяції/кількість точок – 100, кількість міст – 10, кількість складів – 2. Координати складів для найкращого запуску ГА:
[[28.90472412 43.66912842]
[80.00030518 39.99938965]] (рис А.11).

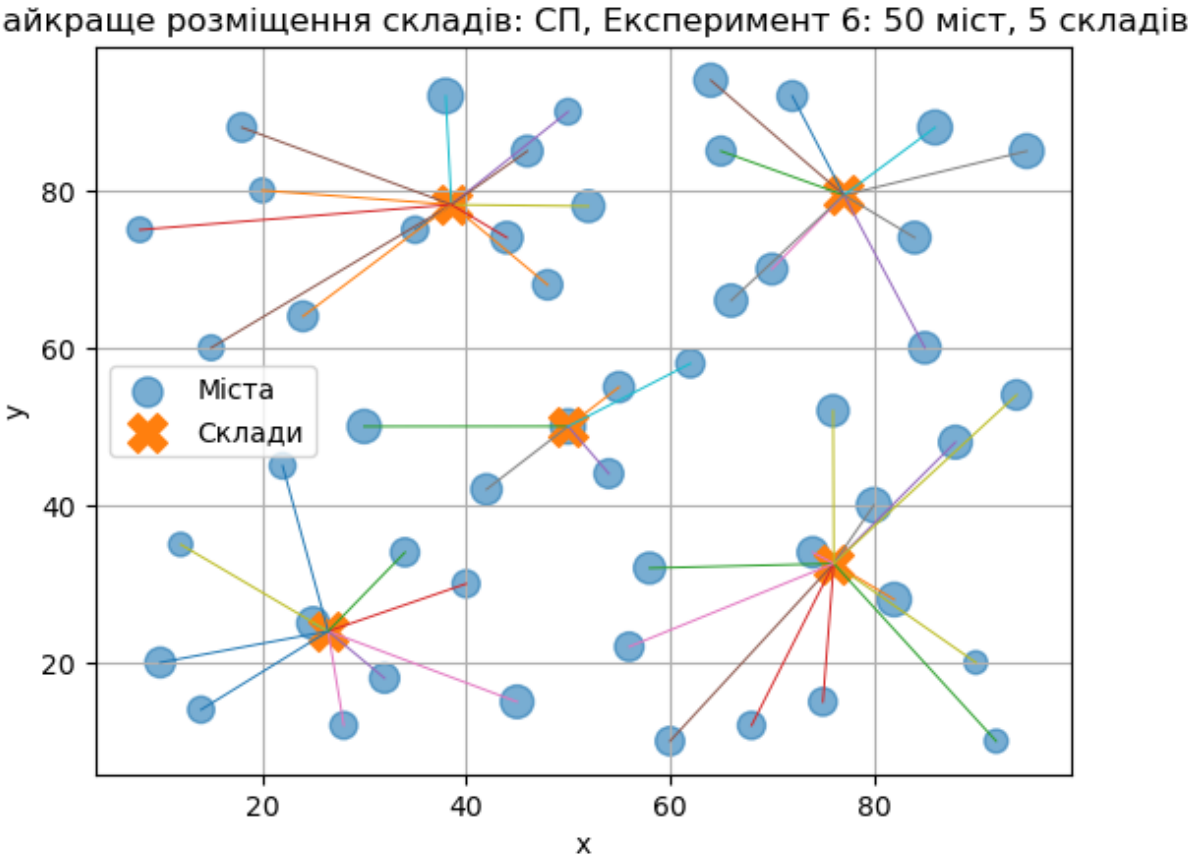


Рисунок А.11 ГА (Експеримент 1)

Координати складів для найкращого запуску СП:

[[80.00199497 40.00018435]

[28.91769704 43.55508094]] (рис А.12).

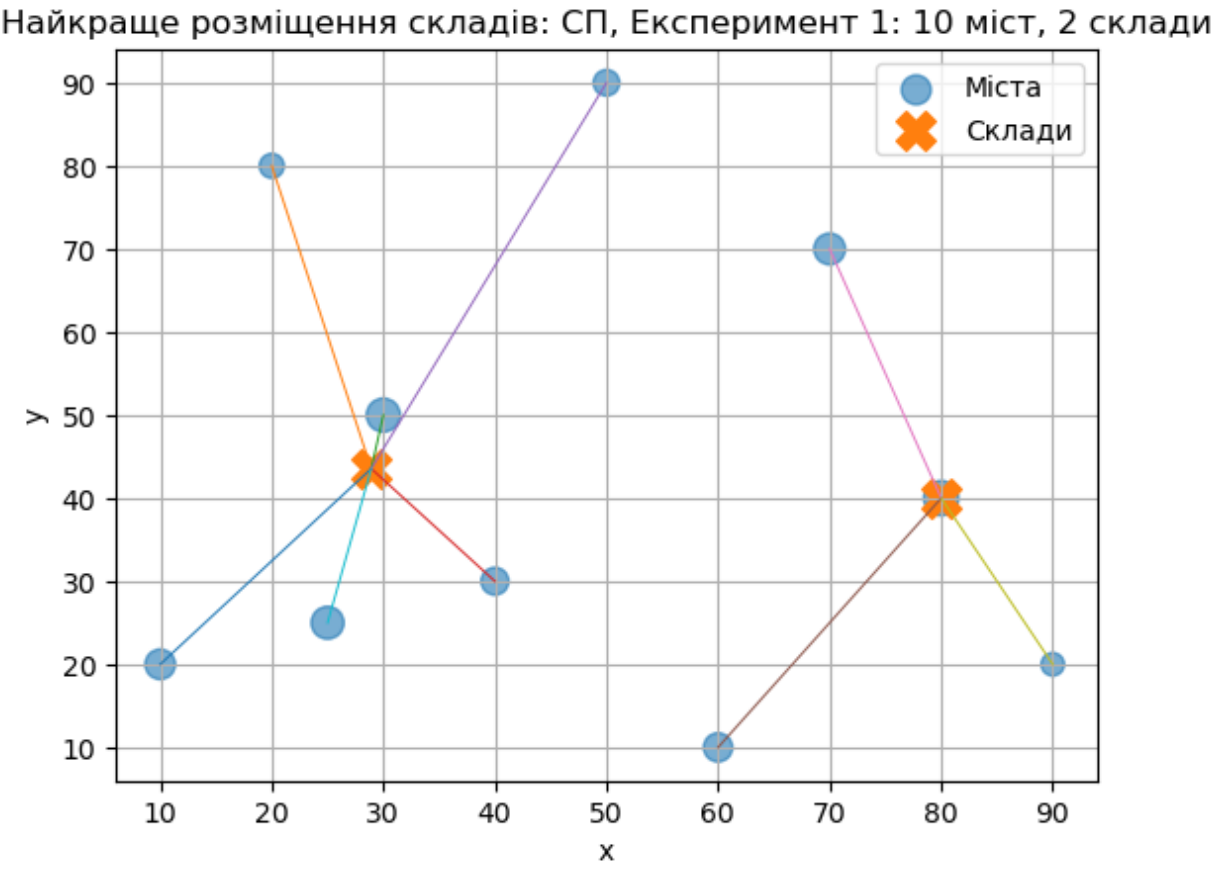


Рисунок А.12 СП (Експеримент 1)

Експеримент 2. Розмір популяції/кількість точок – 200, кількість міст – 10, кількість складів – 3. Координати складів для найкращого запуску ГА:
[[51.70593262 83.75396729]
[79.98199463 39.91851807]
[25.00305176 25.13427734]] (рис А.13).

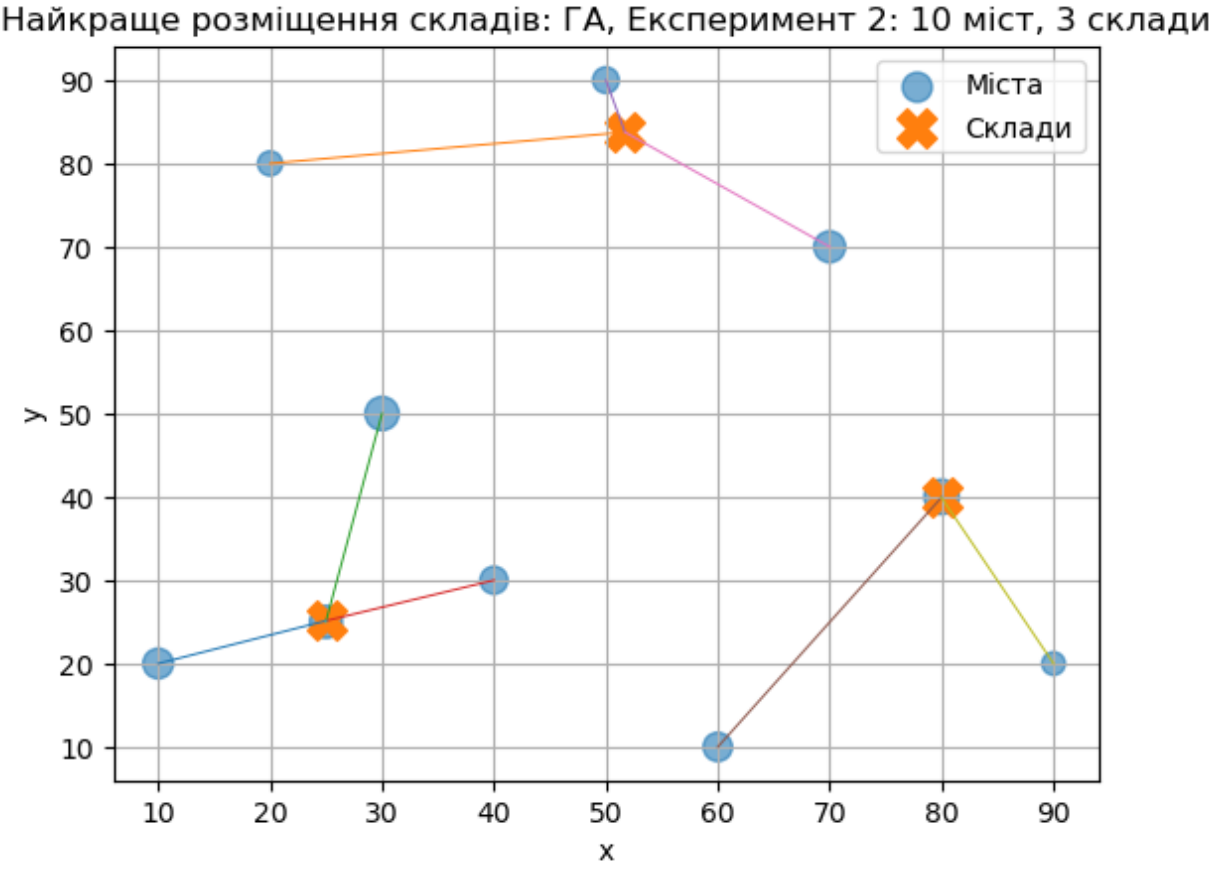


Рисунок А.3 ГА (Експеримент 2)

Координати складів для найкращого запуску СП:

[[25.02233709 25.14898824]

[52.01377707 83.53888725]

[79.98139345 39.9425234]] (рис А.14).

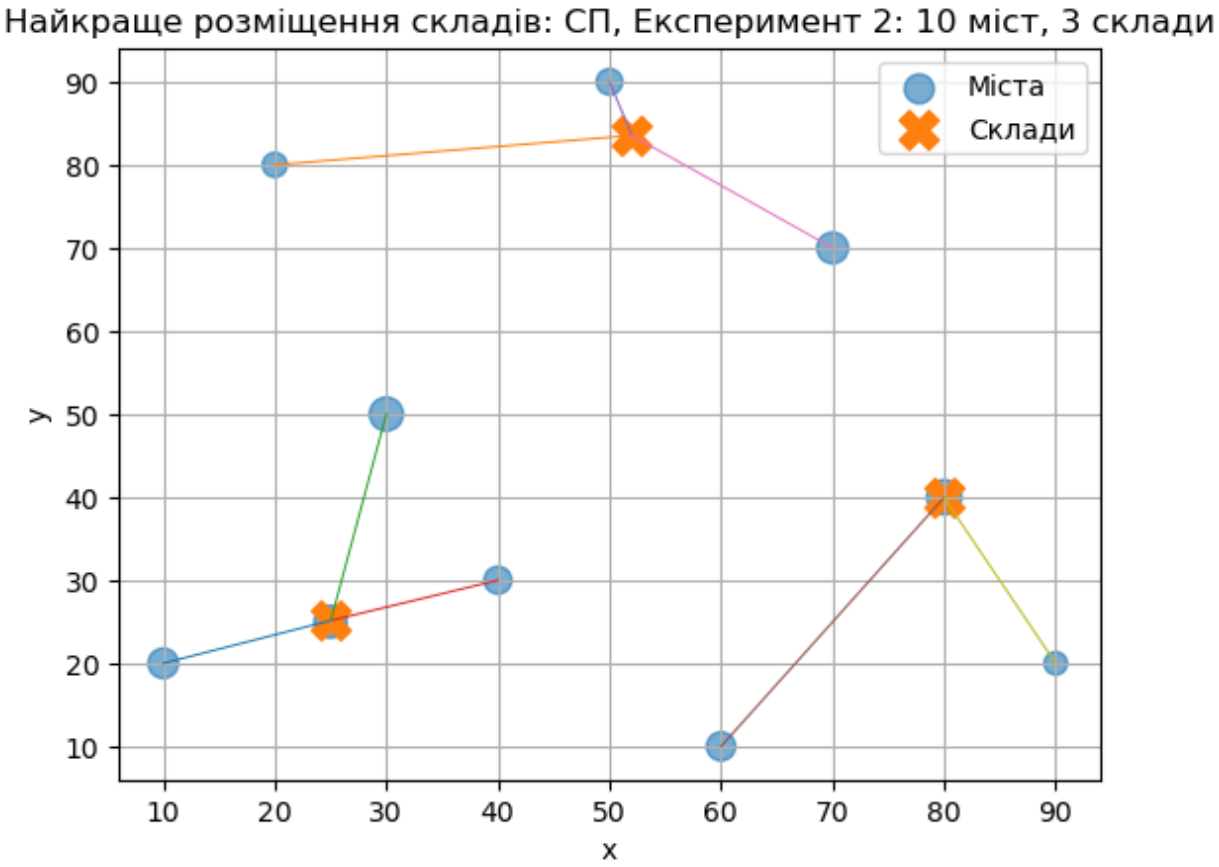


Рисунок А.14 СП (Експеримент 2)

Експеримент 3. Розмір популяції/кількість точок – 200, кількість міст – 20, кількість складів – 2. Координати складів для найкращого запуску ГА:
[[65.91491699 71.64154053]
[36.8347168 27.64587402]] (рис А.15).

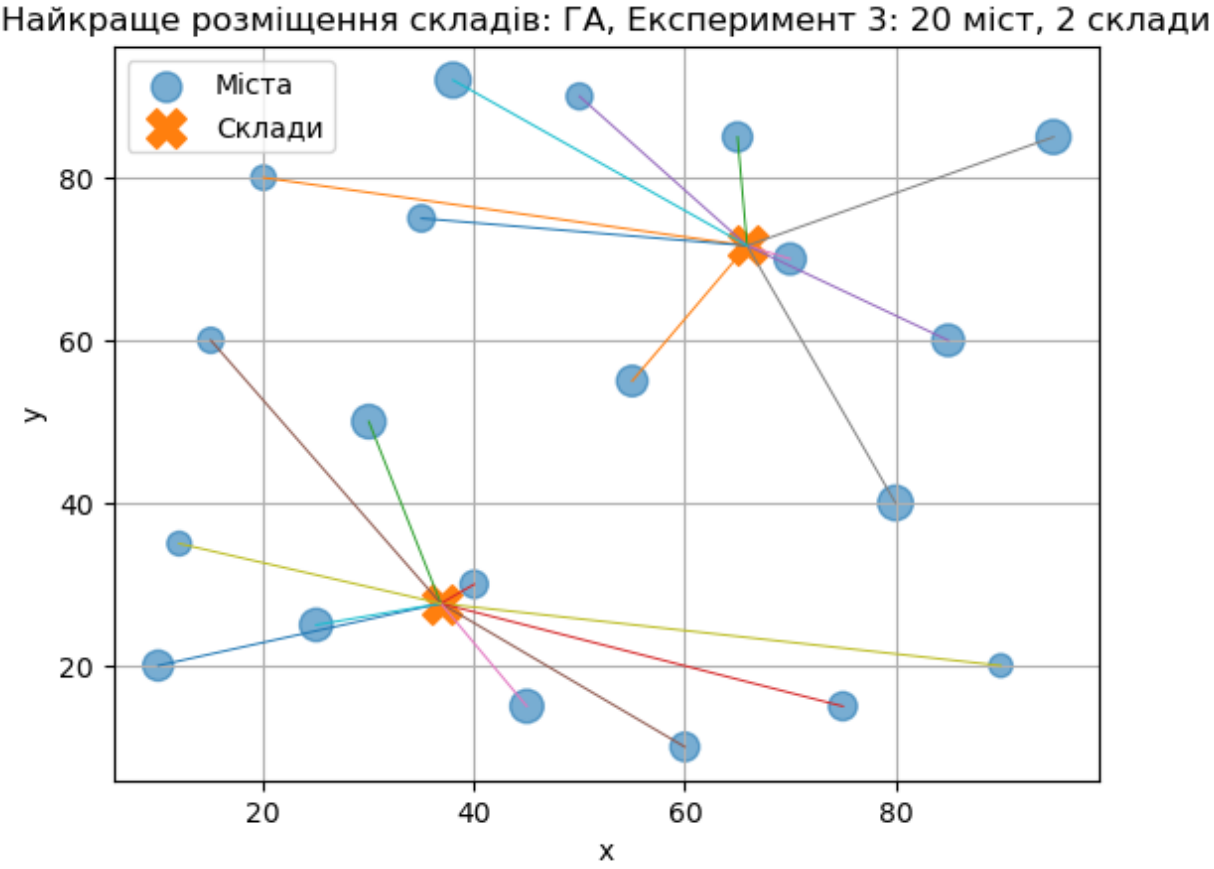


Рисунок А.15 ГА (Експеримент 3)

Координати складів для найкращого запуску СП:

[[65.91325013 71.64423632]

[36.82855923 27.64060532]] (рис А.16).

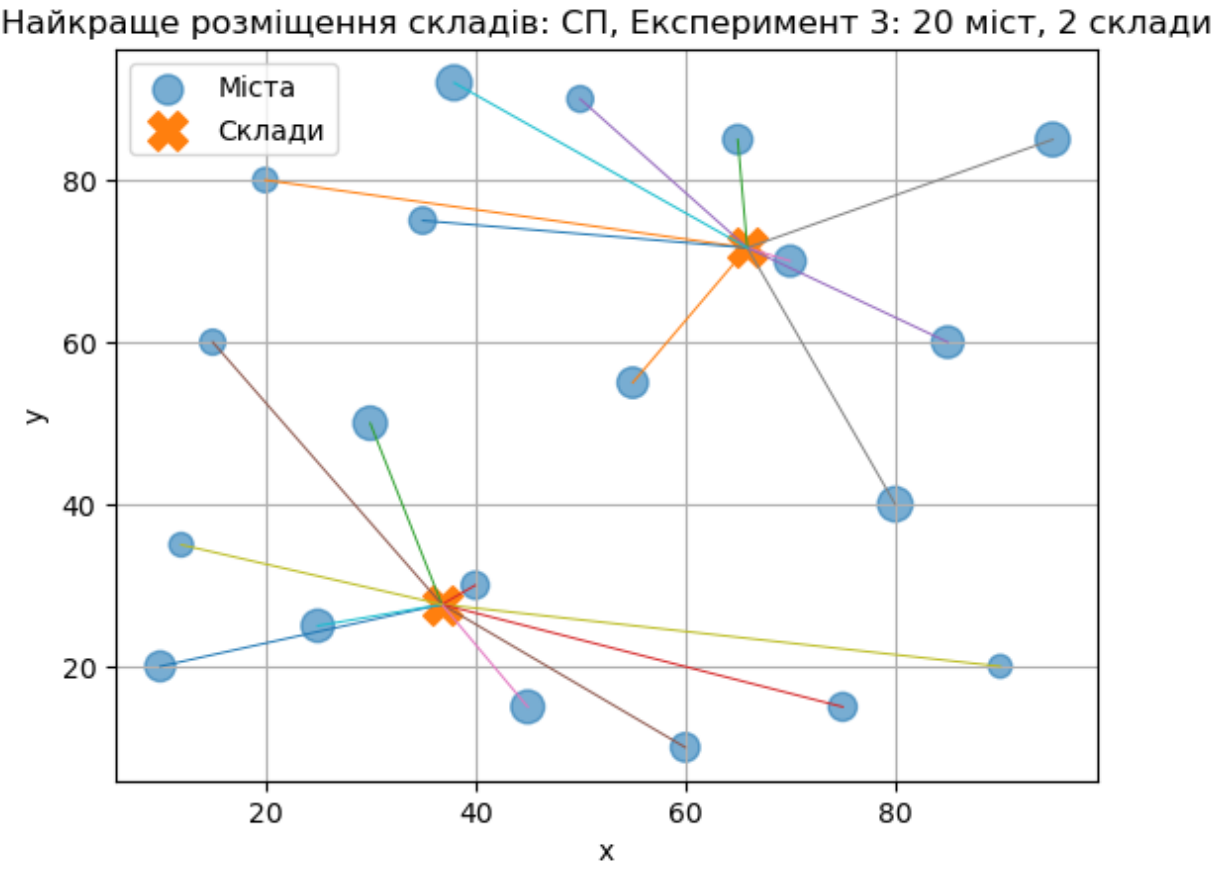


Рисунок А.16 СП (Експеримент 3)

Експеримент 4. Розмір популяції/кількість точок – 200, кількість міст – 20, кількість складів – 3. Координати складів для найкращого запуску ГА:
[[79.24957275 55.34973145]
[28.39660645 27.734375]
[40.19317627 89.01824951]] (рис А.17).

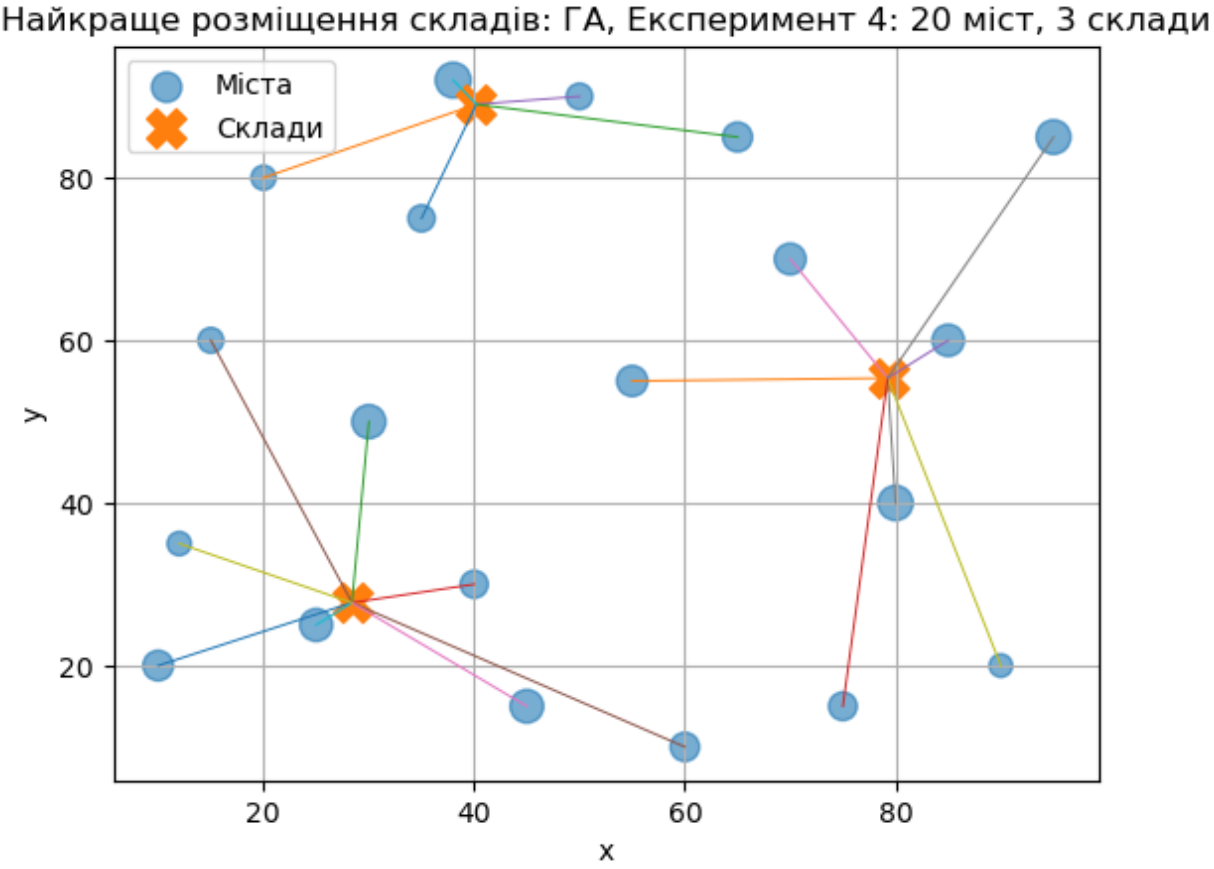


Рисунок А.17 ГА (Експеримент 4)

Координати складів для найкращого запуску СП:

[[40.16855289 89.09489543]

[79.24100734 55.30501186]

[28.32045701 27.76861562]] (рис А.18).

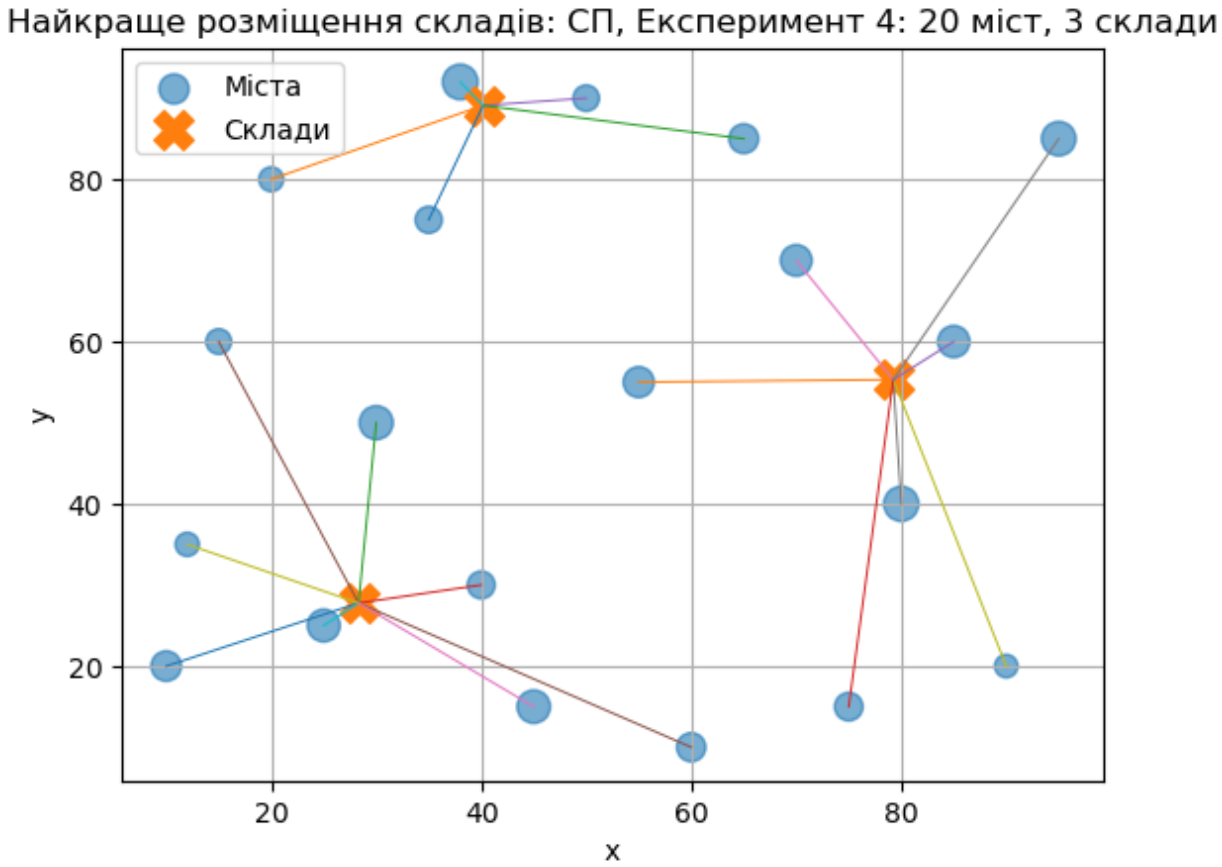


Рисунок А.18 СП (Експеримент 4)

Експеримент 5. Розмір популяції/кількість точок – 200, кількість міст – 50, кількість складів – 3. Координати складів для найкращого запуску ГА:
[[78.54003906 35.94055176]
[54.63256836 78.55987549]
[34.00268555 33.97064209]] (рис А.19).

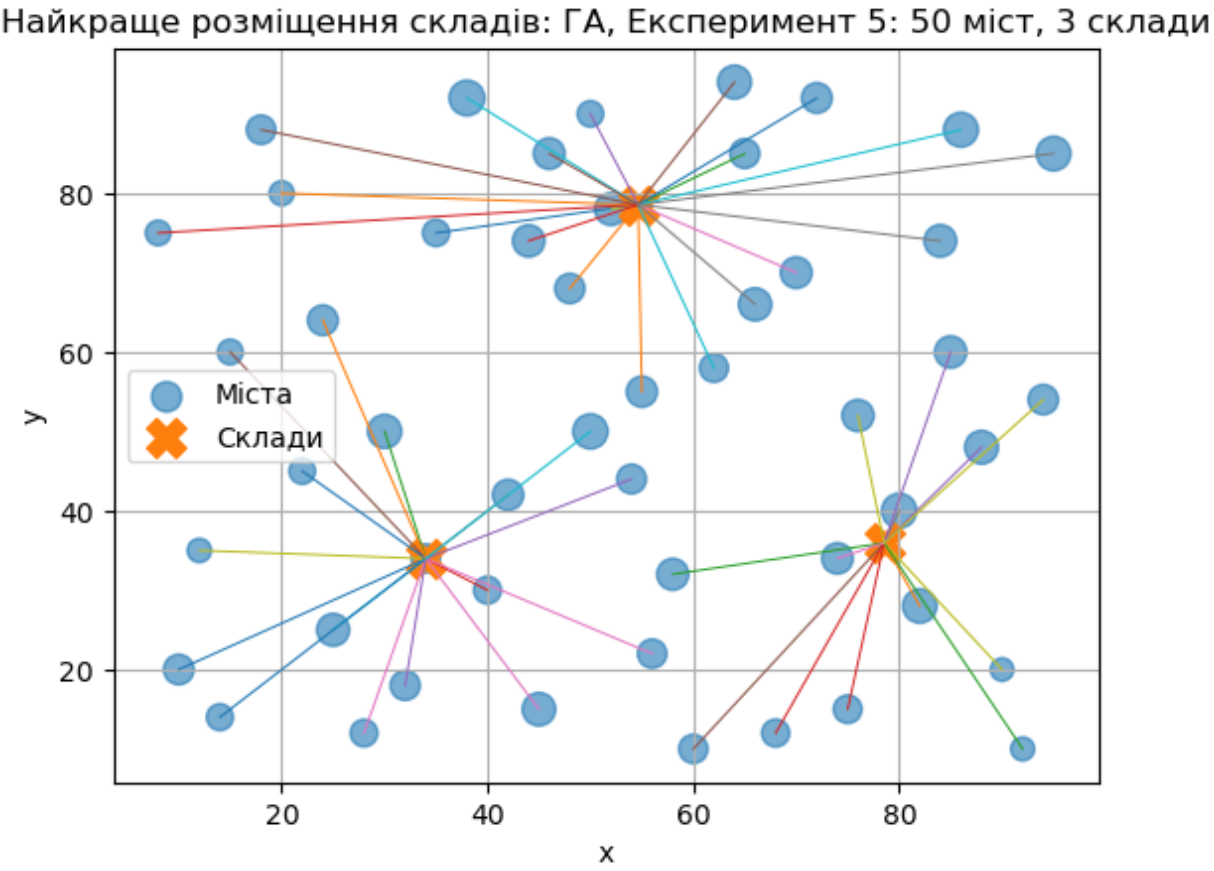


Рисунок А.19 ГА (Експеримент 5)

Координати складів для найкращого запуску СП:

[[34.00312942 33.99265807]

[78.54646806 35.63746284]

[54.61056981 78.53432771]] (рис А.20).

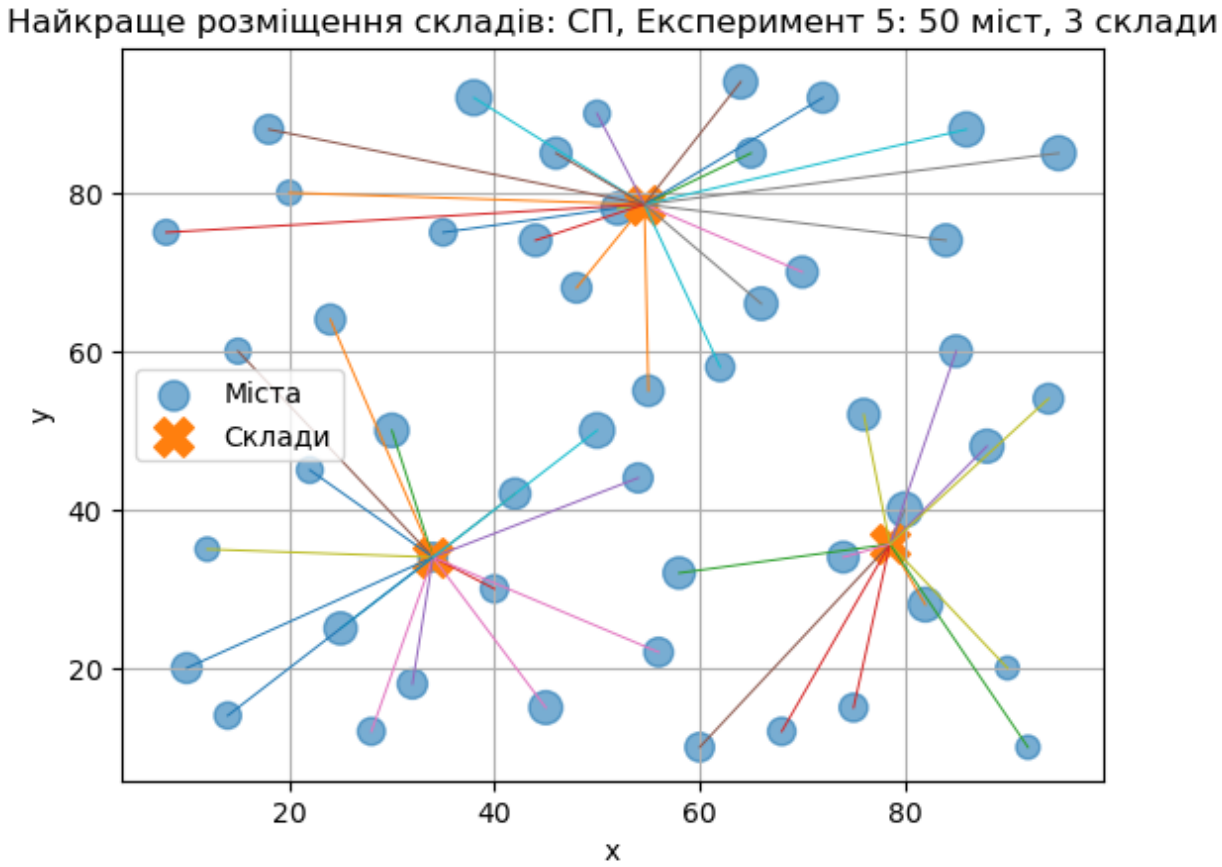


Рисунок А.20 СП (Експеримент 5)

Експеримент 6. Розмір популяції/кількість точок – 200, кількість міст – 50, кількість складів – 5. Координати складів для найкращого запуску ГА:

[[38.0355835 78.03039551]

[77.13012695 79.74700928]

[49.92828369 49.89776611]

[26.62811279 24.0234375]

[76.27716064 32.91015625]] (рис А.21).

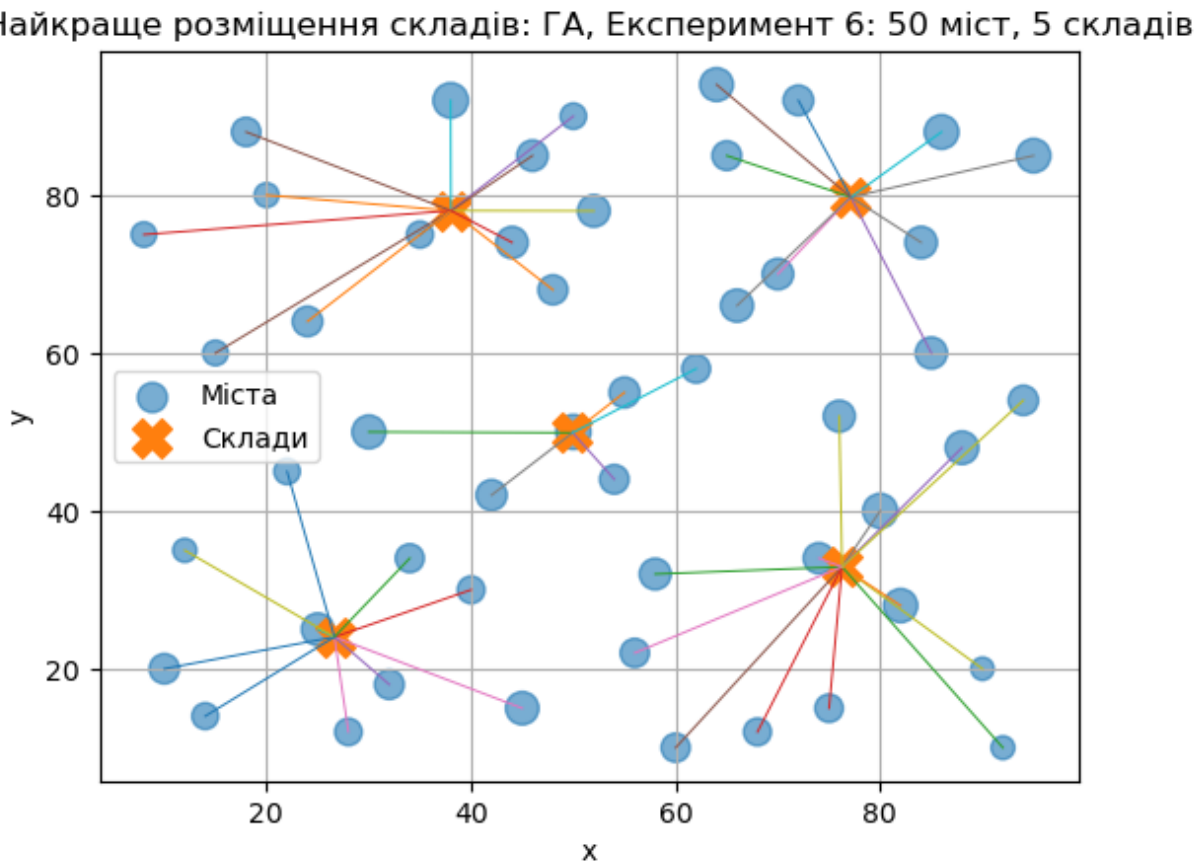


Рисунок А.21 ГА (Експеримент 6)

Координати складів для найкращого запуску СП:

[[26.31741206 24.20157955]

[49.99503156 49.98436483]

[76.32438217 32.61112055]

[76.9719899 79.50805668]

[38.09755133 77.91177604]] (рис А.22).

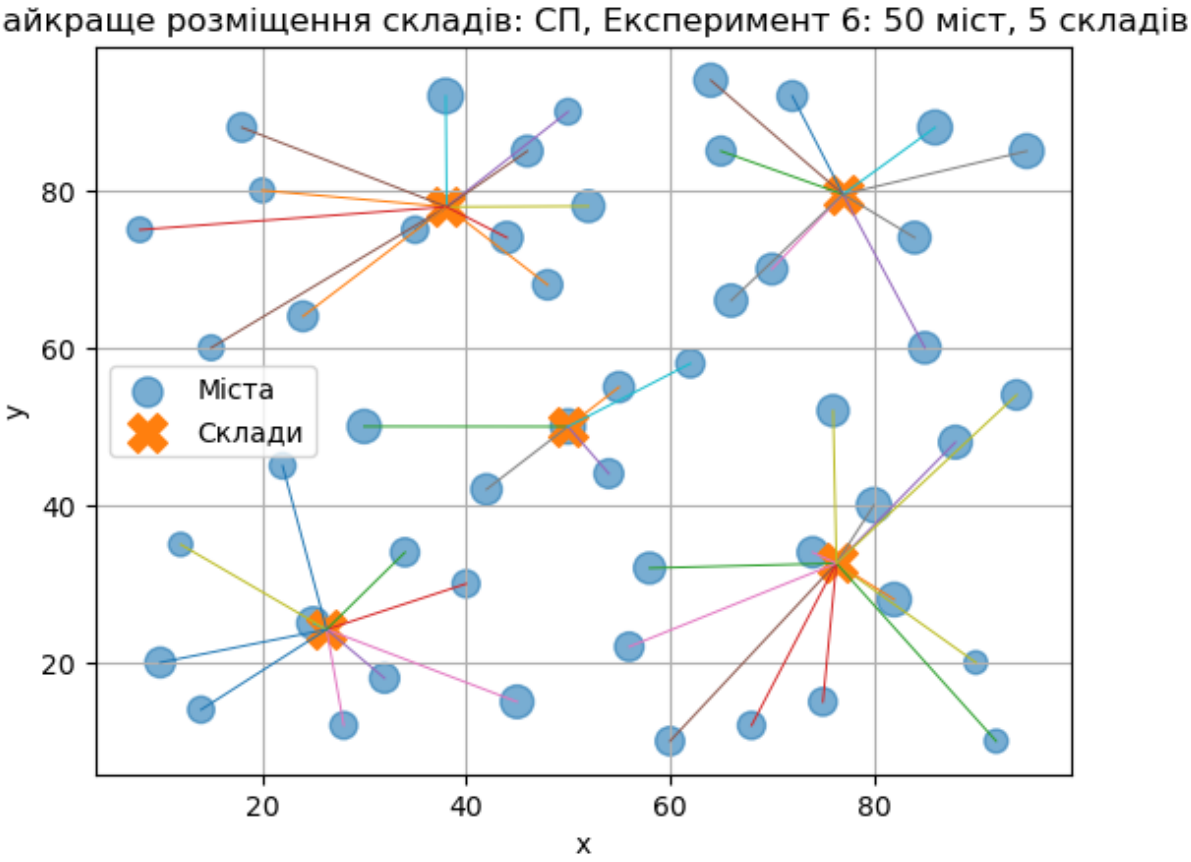


Рисунок А.22 СП (Експеримент 6)

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМИ

```
import numpy as np
import matplotlib.pyplot as plt
import math
import time

# ЦІЛЬОВІ ФУНКЦІЇ

def schwefel(x):
    return 837.9658 - (
        x[0] * math.sin(math.sqrt(abs(x[0]))) +
        x[1] * math.sin(math.sqrt(abs(x[1])))
    )

def michalewicz(x):
    return -math.sin(x[0]) * (math.sin(x[0]**2 / math.pi) **
20) - \
           math.sin(x[1]) * (math.sin(2 * x[1]**2 / math.pi)
** 20)

# ГРАДІЄНТНИЙ СПУСК

# Чисельний градієнт

def numerical_gradient(f, x, h=1e-5):
    x = np.array(x, dtype=float)
    grad = np.zeros_like(x)

    for i in range(len(x)):
        x_forward = x.copy()
        x_backward = x.copy()
```

```
            x_forward[i] += h
            x_backward[i] -= h

            grad[i] = (f(x_forward) - f(x_backward)) / (2 * h)

        return grad

    # Градієнтний спуск

    def gradient_descent(f, start_point, bounds, lr=0.01,
n_iter=1000):
        x = np.array(start_point, dtype=float)

        start_time = time.time()

        for iteration in range(n_iter):

            grad = numerical_gradient(f, x)

            x = x - lr * grad

            for j in range(len(x)):
                x[j] = np.clip(x[j], bounds[j][0], bounds[j][1])

        end_time = time.time()
        elapsed_time = end_time - start_time

        return x, f(x), elapsed_time

    # МЕТОД НЬЮТОНА

    # Чисельний градієнт
```

```
def numerical_gradient(f, x, h=1e-5):
    x = np.array(x, dtype=float)
    grad = np.zeros_like(x)

    for i in range(len(x)):
        x1 = x.copy()
        x2 = x.copy()

        x1[i] += h
        x2[i] -= h

        grad[i] = (f(x1) - f(x2)) / (2 * h)

    return grad

# Чисельна матриця Гессе

def numerical_hessian(f, x, h=1e-4):
    x = np.array(x, dtype=float)

    n = len(x)

    hessian = np.zeros((n, n))

    for i in range(n):
        for j in range(n):

            x1 = x.copy()
            x2 = x.copy()
            x3 = x.copy()
            x4 = x.copy()

            x1[i] += h
            x1[j] += h
```

```
            x2[i] += h
            x2[j] -= h

            x3[i] -= h
            x3[j] += h

            x4[i] -= h
            x4[j] -= h

            hessian[i, j] = (
                f(x1)
                - f(x2)
                - f(x3)
                + f(x4)
            ) / (4 * h * h)

    return hessian

# Метод Ньютона

def newton_method(f, start_points, bounds=None, title=""):

    for start in start_points:
        result = minimize(
            f,
            x0=np.array(start, dtype=float),
            method="trust-ncg",
            jac=lambda x: numerical_gradient(f, x),
            hess=lambda x: numerical_hessian(f, x),
            options={"maxiter": 100}
        )

# ГЕНЕТИЧНИЙ АЛГОРИТМ
```

```
    # Цільова функція
    def objective(x):
        return 837.9658 - (
            x[0] * math.sin(math.sqrt(abs(x[0])))
            + x[1] * math.sin(math.sqrt(abs(x[1])))
        )
    # Параметри
    bounds = [[-500.0, 500.0], [-500.0, 500.0]] # Межі пошуку
    n_iter = 100 # кількість поколінь
    n_bits = 16 # кількість бітів на змінну
    n_pop = 100 # розмір популяції
    r_cross = 0.9 # коефіцієнт кросоверу
    r_mut = 1.0 / (float(n_bits) * len(bounds)) # коефіцієнт
мутації

    # Метод селекції
    # 1 — турнірний
    # 2 — рулетковий
    # 3 — ранговий
    selection_method = 1

    # Декодування бітового рядка у числО
    def decode(bounds, n_bits, bitstring):
        decoded = list()
        largest = 2**n_bits
        for i in range(len(bounds)):
            start, end = i * n_bits, (i * n_bits)+n_bits
            substring = bitstring[start:end]
            chars = ''.join([str(s) for s in substring])
            integer = int(chars, 2)
                value = bounds[i][0] + (integer/largest) *
(bounds[i][1] - bounds[i][0])
            decoded.append(value)
```

```
        return decoded

    # Турнірний відбір
    def tournament_selection(pop, scores, k=3):
        selection_ix = randint(len(pop))
        for ix in randint(0, len(pop), k-1):
            if scores[ix] < scores[selection_ix]:
                selection_ix = ix
        return pop[selection_ix]

    # Рулетковий відбір
    def roulette_selection(pop, scores):
        min_score = min(scores)
        max_score = max(scores)
        if max_score == min_score:
            return pop[randint(0, len(pop))]

        adj_scores = [(max_score - s) / (max_score - min_score)
for s in scores]
        total = sum(adj_scores)
        pick = rand() * total
        current = 0
        for i in range(len(pop)):
            current += adj_scores[i]
            if current > pick:
                return pop[i]

    # Ранговий відбір
    def rank_selection(pop, scores):
        sorted_indices = sorted(range(len(scores)), key=lambda
i: scores[i])
        ranks = [0] * len(scores)
        for rank, idx in enumerate(sorted_indices):
            ranks[idx] = len(scores) - rank
```

```
        total = sum(ranks)
        pick = rand() * total
        current = 0
        for i in range(len(pop)):
            current += ranks[i]
            if current > pick:
                return pop[i]

    # Кросовер двох батьківських особин
    def crossover(p1, p2, r_cross):
        c1, c2 = p1.copy(), p2.copy()
        if rand() < r_cross:
            pt = randint(1, len(p1)-2)
            c1 = p1[:pt] + p2[pt:]
            c2 = p2[:pt] + p1[pt:]
        return [c1, c2]

    # Оператор мутації
    def mutation(bitstring, r_mut):
        for i in range(len(bitstring)):
            if rand() < r_mut:
                bitstring[i] = 1 - bitstring[i]

    # Генетичний алгоритм
    def genetic_algorithm(objective, bounds, n_bits, n_iter,
n_pop, r_cross, r_mut, selection_method):
        pop = [randint(0, 2, n_bits*len(bounds)).tolist() for _
in range(n_pop)]
        best, best_eval = 0, objective(decode(bounds, n_bits,
pop[0]))
        for gen in range(n_iter):
            decoded = [decode(bounds, n_bits, p) for p in pop]
            scores = [objective(d) for d in decoded]
            for i in range(n_pop):
```

```
                if scores[i] < best_eval:
                    best, best_eval = pop[i], scores[i]
                        print(">%d, new best f(%s) = %f" %
(gen, decoded[i], scores[i]))
            if selection_method == 1:
                 selected = [tournament_selection(pop, scores)
for _ in range(n_pop)]
            elif selection_method == 2:
                selected = [roulette_selection(pop, scores) for
_ in range(n_pop)]
            else:
                selected = [rank_selection(pop, scores) for _ in
range(n_pop)]
            children = list()
            for i in range(0, n_pop, 2):
                p1, p2 = selected[i], selected[i+1]
                for c in crossover(p1, p2, r_cross):
                    mutation(c, r_mut)
                    children.append(c)
            pop = children
        return [best, best_eval]

    # СПІРАЛЬНИЙ ПОШУК

    # Цільова функція
    def objective(x):
        return 837.9658 - (
            x[0] * math.sin(math.sqrt(abs(x[0])))
            + x[1] * math.sin(math.sqrt(abs(x[1])))
        )
    # Параметри
    bounds = [[-500.0, 500.0], [-500.0, 500.0]] # Межі пошуку
    n_points = 100          # кількість точок
    n_iter = 100             # кількість ітерацій
```

```
    theta = np.pi / 3        # кут обертання
    r = 0.95                 # коефіцієнт наближення
    rotation_matrix      =      np.array([[np.cos(theta),      -
np.sin(theta)],
                                [np.sin(theta),  np.cos(theta)]])
#матриця повороту

    # Ініціалізація точок
    points = np.random.uniform(low=[b[0] for b in bounds],
                               high=[b[1] for b in bounds],
                               size=(n_points, 2))
    # Основний цикл
    for iter in range(n_iter):
        scores = []
        for x in points:
            if (
                x[0] < bounds[0][0] or x[0] > bounds[0][1] or
                x[1] < bounds[1][0] or x[1] > bounds[1][1]
            ):
                scores.append(np.inf)
            else:
                scores.append(objective(x))

        scores = np.array(scores)
        best_idx = np.argmin(scores)
        x_best = points[best_idx]

        for i in range(n_points):
            diff = points[i] - x_best
            points[i] = x_best + r * rotation_matrix @ diff

    final_scores = []

    for x in points:
```

```
    if (
        x[0] < bounds[0][0] or x[0] > bounds[0][1] or
        x[1] < bounds[1][0] or x[1] > bounds[1][1]
    ):
        final_scores.append(np.inf)
    else:
        final_scores.append(objective(x))

final_scores = np.array(final_scores)



# ЗАДАЧА РОЗМІЩЕННЯ СКЛАДІВ

# Набір міст
all_cities = np.array([
    [10, 20],
    [20, 80],
    [30, 50],
    [40, 30],
    [50, 90],
    [60, 10],
    [70, 70],
    [80, 40],
    [90, 20],
    [25, 25],

    [35, 75],
    [55, 55],
    [65, 85],
    [75, 15],
    [85, 60],
    [15, 60],
    [45, 15],
```

```
[95, 85],
[12, 35],
[38, 92],

[22, 45],
[48, 68],
[58, 32],
[68, 12],
[88, 48],
[18, 88],
[28, 12],
[42, 42],
[52, 78],
[62, 58],

[72, 92],
[82, 28],
[92, 10],
[8, 75],
[32, 18],
[46, 85],
[56, 22],
[66, 66],
[76, 52],
[86, 88],

[14, 14],
[24, 64],
[34, 34],
[44, 74],
[54, 44],
[64, 94],
[74, 34],
[84, 74],
```

```
    [94, 54],
    [50, 50]
])

# Набір попиту міст
all_demand = np.array([
    120, 80, 150, 100, 90,
    110, 130, 160, 70, 140,

    95, 125, 115, 105, 135,
    85, 145, 155, 75, 165,

    90, 118, 132, 101, 147,
    112, 98, 126, 139, 108,

    122, 152, 73, 84, 116,
    129, 111, 143, 134, 157,

    93, 121, 109, 136, 114,
    149, 127, 138, 119, 160
])

# Цільова функція

def create_objective(cities, demand, n_warehouses):
    def objective(x):
        warehouses = np.array(x).reshape(n_warehouses, 2)

        total_cost = 0

        for city, d in zip(cities, demand):
            distances = []

            for warehouse in warehouses:
```

```
                distance = math.sqrt(
                    (city[0] - warehouse[0]) ** 2 +
                    (city[1] - warehouse[1]) ** 2
                )
                distances.append(distance)

            nearest_distance = min(distances)
            total_cost += d * nearest_distance

        return total_cost

    return objective
```

ДОДАТОК В. ПРЕЗЕНТАЦІЯ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут імені Ігоря Сікорського»
ННІ прикладного системного аналізу · Кафедра математичних методів системного аналізу

Дипломна робота на здобуття ступеня бакалавра на тему

«Порівняння метаевристичних методів оптимізації багатоекстремальних функцій»

Виконав
Студент IV курсу, групи КА-21
Ніколаєв Ігор Дмитрович

Науковий керівник
Доцент кафедри ММСА, д.т.н.
Савченко Ілля Олександрович

Київ — 2026

Об'єкт, предмет і мета дослідження

1 Об'єкт дослідження

Процеси глобальної оптимізації складних багатоекстремальних функцій.

2 Предмет дослідження

Метаевристичні методи глобальної оптимізації – генетичний алгоритм та метод спірального пошуку, а також програмні засоби їх реалізації для дослідження тестових і прикладних цільових функцій.

3 Мета роботи

Дослідити особливості задач глобальної оптимізації багатоекстремальних функцій, обґрунтувати доцільність метаевристичних методів, реалізувати генетичний алгоритм і метод спірального пошуку та провести їх експериментальне порівняння на тестових функціях Швефеля і Міхалевича та прикладній задачі розміщення складів.

2

Актуальність теми

Чому задача важлива

1 Зростання складних задач оптимізації

У системному аналізі, штучному інтелекті, логістиці та інженерному проєктуванні постійно зростає кількість задач, де потрібно знайти найкращий набір параметрів за складним критерієм ефективності.

2 Багатоекстремальні функції

Такі функції мають безліч локальних екстремумів і складний рельєф простору пошуку, через що алгоритм може передчасно збігатися до локального мінімуму замість глобального.

3 Обмеженість класичних методів

Детерміновані та градієнтні методи спираються на похідні й початкову точку, тому для багатоекстремальних задач часто застрягають у локальних мінімумах.

4 Перевага метаевристик

Стохастичний пошук без обчислення похідних дозволяє досліджувати простір глобально. Порівняння ГА та СП визначає їхні переваги, недоліки й доцільність застосування.

3

Постановка задачі

Математична постановка

Глобальна оптимізація — пошук вектора параметрів, що мінімізує цільову функцію:

$$\min f(x),\ x = (x_1, \ldots, x_n) \in D$$

Умова глобального мінімуму:

$$f(x^*) \le f(x),\quad \forall\, x \in D$$

f(x) — цільова функція; **D** — область допустимих розв'язків. Складність: безліч локальних мінімумів і ризик передчасної збіжності.

Завдання дослідження

- проаналізувати особливості задач глобальної оптимізації багатоекстремальних функцій;
- розглянути обмеження традиційних детермінованих і градієнтних методів;
- дослідити принципи генетичного алгоритму та математичну модель методу спірального пошуку;
- розглянути тестові функції Швефеля та Міхалевича; сформулювати задачу розміщення складів;
- реалізувати програмні засоби для дослідження алгоритмів;
- провести експериментальне порівняння генетичного алгоритму та методу спірального пошуку;
- оцінити ефективність за якістю розв'язку, стабільністю та кількістю обчислень цільової функції.

4

Багатоекстремальні функції та обмеження класичних методів

Властивості багатоекстремальних функцій

- наявність кількох локальних мінімумів / максимумів;
- складна форма поверхні цільової функції;
- висока залежність результату від початкової точки;
- ризик передчасної збіжності алгоритму;
- складність побудови універсального методу пошуку глобального оптимуму.

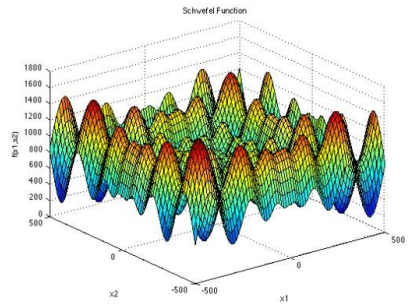
Обмеження градієнтних методів і методу Ньютона

- потребують обчислення похідних (градієнт, матриця Гессе);
- сильна залежність від вибору початкової точки;
- орієнтація лише на локальну інформацію → застрягання в локальному мінімумі;
- низька ефективність у високорозмірних задачах;
- непридатність до недиференційованих або неявно заданих функцій.

5

Тестові багатоекстремальні функції

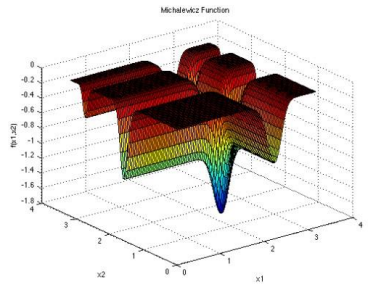
Функція Швефеля



$$f(x,y) = 837.9658 - x\sin\left(\sqrt{|x|}\right) - \sin\left(\sqrt{|y|}\right)$$

Область: [−500; 500]; мінімум = 0 при (420.9687; 420.9687).
Багато локальних мінімумів; глобальний – біля меж області.

Функція Міхалевича



$$f(x,y) = -sin(x)\,sin^{20}\left(\frac{x^2}{\pi}\right) - sin(y)\,sin^{20}\left(\frac{2y^2}{\pi}\right)$$

Область: [0; π]; мінімум ≈ −1.8013 при (2.20; 1.57).
Вузькі локальні мінімуми; висока чутливість до координат.

6

Результати класичних методів

Градієнтний спуск і метод Ньютона досліджено на функціях Швефеля та Міхалевича з 10 різних стартових точок. Успіх: абсолютна похибка ≤ 1 (Швефеля), відносна похибка ≤ 1 % (Міхалевича).

Зведене порівняння (10 стартових точок)

Метод · функція	Успішних стартів	Найкраща похибка	Час, с
Град. спуск · Швефеля	3 / 10	0.0015	≈ 0.021
Град. спуск · Міхалевича	5 / 10	0.0002 %	≈ 0.024
Метод Ньютона · Швефеля	5 / 10	$2.5 \cdot 10^{-5}$	≈ 0.001
Метод Ньютона · Міхалевича	6 / 10	0.0002 %	≈ 0.001

Метод Ньютона збігається швидше (≈ 20× менший час) і точніше, але обидва методи — локальні.

Приклад передчасної збіжності — градієнтний спуск, функція Швефеля

Старт	Знайдено f(x)	Успіх
[0, 0]	830.09	ні
[380, 380]	3.41	ні
[400, 400]	0.76	так
[420, 420]	0.0015	так
[450, 450]	1.43	ні

Успіх лише зі стартів поблизу глобального мінімуму (x ≈ 420.97); з віддалених точок метод застрягає в локальних мінімумах.

Висновок

Класичні методи знаходять глобальний мінімум лише при вдалому виборі початкової точки й залежать від похідних. Для надійного глобального пошуку багатоекстремальних функцій потрібні метаевристичні методи — генетичний алгоритм і метод спірального пошуку.

7

Метод 1. Генетичний алгоритм

Популяційний еволюційний метод: кожен розв'язок — особина (бітова хромосома $n_{bits} = 16$); популяція поступово вдосконалюється через селекцію, кросовер і мутацію.

Початкова популяція → Обчислення функції пристосованості → Селекція → Кросовер → Мутація → Нове покоління

↺ повтор до досягнення фіксованої кількості поколінь

Параметри

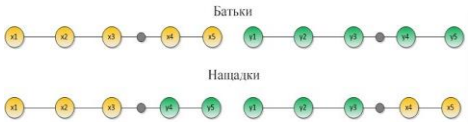
Розмір популяції $n_{pop} = 100$.
Кількість поколінь $n_{iter} = 100$.
Коеф. кросоверу $r_{cross} \in \{0.5;\ 0.9\}$.
Коеф. мутації $r_{mut} \in \{0.03125;\ 0.1\}$.

Способи селекції

Турнірний відбір – найкраща з випадкової групи особин.
Рулетковий відбір – імовірність пропорційна цільовій функції.
Ранговий відбір – імовірність залежить від рангу особини.

Одноточковий кроссовер

Для кожної пари батьківських хромосом випадково перевіряється умова кросоверу. Якщо випадково згенероване число менше за задану ймовірність кросоверу, випадково обирається точка розриву хромосоми. Після цього батьківські хромосоми обмінюються відповідними частинами.



Мутація у двійковому кодуванні

Для кожного біта хромосоми випадково перевіряється умова мутації. Якщо випадково згенероване число менше за задану ймовірність мутації, відповідний біт інвертується. Тобто значення біта змінюється на протилежне.

Кількість обчислень цільової функції

$$N^{GA} = 1 + n_{pop} \cdot n_{iter}$$

8

Метод 2. Метод спірального пошуку

Геометрична концепція: множина точок-кандидатів обертається навколо поточного найкращого розв'язку та поступово наближається до нього за спіральною траєкторією. Оператори селекції, кросоверу й мутації не використовуються.

$$x_i(k+1) = x^*(k) + rR(\theta)(x_i(k) - x^*(k))$$

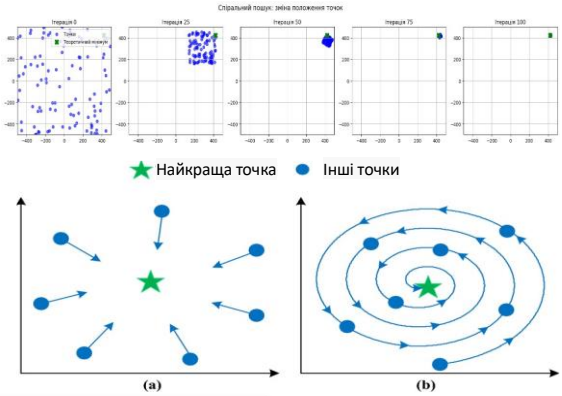
$x_i(k)$ — положення i-ї точки на ітерації k; $x^*(k)$ — поточний найкращий розв'язок; r — коефіцієнт наближення; $R(\theta)$ — матриця повороту; θ — кут повороту.

Обертання навколо центру + наближення до найкращої точки

Параметри

Кількість точок $n_{\text{points}} \in \{100;\ 200\}$.
Кількість ітерацій $n_{iter} = 100$.
Кут обертання $\theta \in \{\pi/3;\ \pi/4;\ \pi/6\}$.
Коефіцієнт наближення $r \in \{0.98;\ 0.95;\ 0.9\}$.

Зміна положень точок під час ітерацій (функція Швефеля)



Кількість обчислень цільової функції

$$N^{\text{SP}} = n_{\text{points}} \cdot (n_{iter} + 1)$$

Прикладна задача: розміщення складів

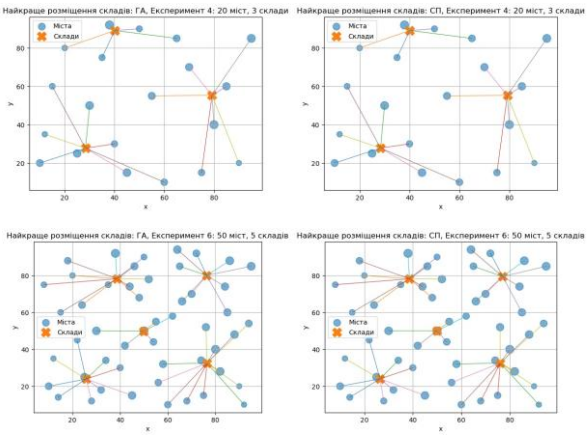
Прикладна багатоекстремальна задача логістичного типу: знайти координати фіксованої кількості складів так, щоб мінімізувати сумарні транспортні витрати. Кожне місто обслуговується найближчим складом.

$$F(X) = \sum_{i=1}^{N} D_i \cdot \min_{j=1,\dots,k} \sqrt{(a_i - x_j)^2 + (b_i - y_j)^2} \to \min,\quad X \in \Omega = [0, 100]^{2k}$$

(a_i, b_i) — координати міста, D_i — попит (вага), (x_j, y_j) — координати складу, k — кількість складів

Чому задача багатоекстремальна

- оператор «найближчий склад» поділяє простір на області закріплення міст;
- різні конфігурації складів дають різні локально-оптимальні розв'язки;
- розмірність задачі m = 2k зростає зі збільшенням кількості складів.



Програмна реалізація

Технологічний стек

Python — мова реалізації
NumPy — масиви та числові обчислення
Matplotlib — графіки та візуалізація
Jupyter Notebook — середовище експериментів

Протокол експерименту

- 100 незалежних запусків на кожен набір параметрів;
- критерій завершення – 100 поколінь / ітерацій;
- успіх: абсолютна похибка ≤ 0.01 (Швефеля), відносна похибка ≤ 1 % (Міхалевича);
- метрики: середнє / найкраще / найгірше, σ, час, успішність.

Що реалізовано

Традиційні методи
градієнтний спуск і метод Ньютона – для демонстрації обмежень локального пошуку.

Генетичний алгоритм
бітове кодування й декодування, три способи селекції, одноточковий кросовер, мутація.

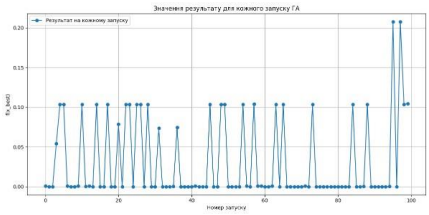
Метод спірального пошуку
генерація точок, пошук найкращої точки, оновлення через матрицю повороту й коефіцієнт наближення.

Цільові функції
Швефеля, Міхалевича та задача розміщення складів; побудова графіків збіжності й результатів.

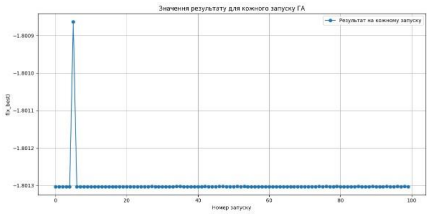
11

Результати: генетичний алгоритм (тестові функції)

Приклади графіків значень за 100 запусків



функція Швефеля



функція Міхалевича

Всі результати

Функція	Спосіб відбору	r_cross	r_mut	Середній результат	Найкращий результат	Найгірший результат	Середній час, с	Успішність %
Функція Швефеля (мінімум: 0)	Турнірний	0.9	0.03125	0.029300	0.000034	0.207357	0.284381	72
		0.9	0.1	0.034601	0.000465	0.258639	0.296516	32
		0.5	0.03125	0.030463	0.000034	0.207357	0.293661	51
		0.5	0.1	0.049103	0.000301	0.316145	0.286745	24
	Рулеточний	0.9	0.03125	0.050733	0.000251	0.364414	0.293269	23
		0.9	0.1	0.515204	0.000093	2.693009	0.293939	4
		0.5	0.03125	0.076348	0.000282	1.551655	0.285810	27
		0.5	0.1	0.637349	0.008382	3.047488	0.287237	3
	Ранговий	0.9	0.03125	0.013841	0.000034	0.161535	0.385892	78
		0.9	0.1	0.190213	0.000112	1.521968	0.388109	8
		0.5	0.03125	0.014852	0.000034	0.240102	0.379276	85
		0.5	0.1	0.253836	0.002259	0.966439	0.382039	6
Функція Міхалевича (мінімум: -1.8013)	Турнірний	0.9	0.03125	-1.801299	-1.801303	-1.800863	0.300782	100
		0.9	0.1	-1.801202	-1.801303	-1.800748	0.296084	100
		0.5	0.03125	-1.801268	-1.801303	-1.800712	0.286368	100
		0.5	0.1	-1.801225	-1.801303	-1.800917	0.287100	100
	Рулеточний	0.9	0.03125	-1.801255	-1.801303	-1.800841	0.295330	100
		0.9	0.1	-1.801072	-1.801302	-1.799406	0.296638	100
		0.5	0.03125	-1.801230	-1.801302	-1.800541	0.285227	100
		0.5	0.1	-1.800978	-1.801303	-1.799020	0.286605	100
	Ранговий	0.9	0.03125	-1.801296	-1.801303	-1.800857	0.388671	100
		0.9	0.1	-1.800913	-1.801301	-1.798754	0.381535	100
		0.5	0.03125	-1.801302	-1.801303	-1.801289	0.360242	100
		0.5	0.1	-1.800814	-1.801303	-1.797043	0.365067	100

Ключові результати

Швефеля: найкращий – ранговий відбір ($r_{cross} = 0.5, r_{mut} = 0.03125$): успішність 85 %, середнє 0.0149.

Міхалевича: стабільні 100 % майже для всіх конфігурацій.

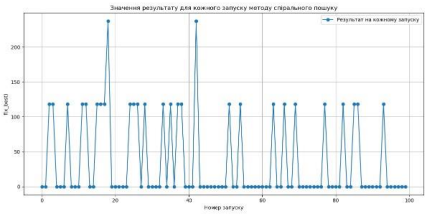
Рулетковий відбір – найгірший; велика мутація (0.1) знижує успіх до 3–4 %.

Час: 0.28–0.39 с на запуск.

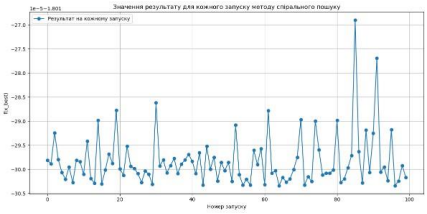
12

Результати: метод спірального пошуку (тестові функції)

Приклади графіків значень за 100 запусків



Функція Швефеля



Функція Міхалевича

Всі результати

Функція	Кількість точок	theta	r	Середній результат	Найкращий результат	Найгірший результат	Середній час, с	Успішність %
Функція Швефеля (мінімум: 0)	100	π/3	0.98	13.387233	0.000119	119.932817	0.056777	1
		π/3	0.95	35.533839	0.000067	236.880996	0.059161с	70
		π/3	0.90	46.190976	0.000025	236.876695	0.061111	62
		π/4	0.98	11.944156	0.019953	120.127952	0.055276	0
		π/4	0.95	37.902771	0.000102	236.877622	0.057585	66
		π/4	0.90	58.034810	0.000025	236.876695	0.058778	54
		π/6	0.98	18.021620	0.018597	237.736173	0.053301	0
		π/6	0.95	52.119625	0.000867	236.886625	0.058142	49
		π/6	0.90	53.297395	0.000026	236.876713	0.060789	59
	200	π/3	0.98	6.585722	0.006336	120.417721	0.112336	2
		π/3	0.95	13.029322	0.000047	236.876802	0.113946	90
		π/3	0.90	27.240842	0.000025	236.876695	0.117892	78
		π/4	0.98	11.320786	0.006472	120.259012	0.110684	1
		π/4	0.95	11.845229	0.000092	118.443851	0.117005	90
		π/4	0.90	34.347143	0.000025	118.438360	0.122276	71
		π/6	0.98	7.948646	0.026362	121.838537	0.110079	0
		π/6	0.95	31.981215	0.000411	236.881606	0.123854	74
		π/6	0.90	22.503314	0.000025	118.438758	0.121418	81
Функція Міхалевича (мінімум: -1.8013)	100	π/3	0.98	-1.798976	-1.801155	-1.787591	0.065151	100
		π/3	0.95	-1.801298	-1.801303	-1.801269	0.065320	100
		π/3	0.90	-1.793290	-1.801303	-1.000000	0.069722	99
		π/4	0.98	-1.799257	-1.801269	-1.792883	0.066520	100
		π/4	0.95	-1.801298	-1.801303	-1.801269	0.068943	100
		π/4	0.90	-1.793290	-1.801303	-1.000000	0.066826	99
		π/6	0.98	-1.799244	-1.801243	-1.791019	0.066331	100
		π/6	0.95	-1.801296	-1.801303	-1.801260	0.065568	100
		π/6	0.90	-1.801303	-1.801303	-1.801297	0.065859	100
	200	π/3	0.98	-1.800074	-1.801302	-1.794582	0.133938	100
		π/3	0.95	-1.801302	-1.801303	-1.801290	0.133200	100
		π/3	0.90	-1.801303	-1.801303	-1.801303	0.133358	100
		π/4	0.98	-1.800227	-1.801298	-1.797045	0.133911	100
		π/4	0.95	-1.801302	-1.801303	-1.801292	0.133177	100
		π/4	0.90	-1.801303	-1.801303	-1.801303	0.134175	100
		π/6	0.98	-1.800197	-1.801267	-1.792557	0.133230	100
		π/6	0.95	-1.801300	-1.801303	-1.801287	0.133280	100
		π/6	0.90	-1.801303	-1.801303	-1.801303	0.137381	100

Ключові результати

Швефеля: найкраще – 200 точок, r=0.95, θ=π/3 або π/4 → успішність до 90 %.

r=0.98 — занадто повільне звуження; r=0.90 — передчасна збіжність.

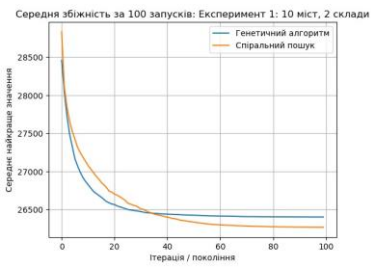
Міхалевича: практично 100 % для всіх конфігурацій.

Час: 0.05–0.13 с — швидше за ГА.

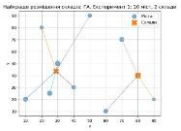
13

Результати: задача розміщення складів

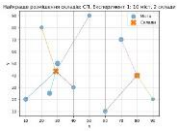
Середня збіжність (10 міст, 2 склади)



Збіжність ГА та СП



ГА



СП

Всі результати 12 експериментів

Кількість популяцій/точок	Алгоритм	Кількість міст	Кількість складів	Середній результат	Найкращий результат	Найгірший результат	Середній час, с	Стандартне відхилення
100	Генетичний алгоритм	10	2	26400.4589	26246.0042	26775.1862	1.2146	189.9310
		10	3	19048.6864	18947.2327	19412.5356	1.4789	125.4877
		20	2	60340.1663	60270.2368	62274.7835	1.9605	209.2814
		20	3	47681.1269	47239.7435	48722.0979	2.5493	326.3064
		50	3	121460.5344	120843.0679	125212.7868	5.7978	892.0543
		50	5	91670.5325	90344.7404	94819.3081	6.6720	1094.5269
	Спіральний пошук	10	2	26266.0830	26245.9759	26335.7956	0.6448	32.8684
		10	3	18963.1045	18947.0184	19667.7965	0.7144	100.3481
		20	2	60270.3422	60270.2241	60270.6175	1.3181	0.0712
		20	3	47385.0955	47239.7160	48163.1166	1.5652	323.2317
		50	3	120900.1877	120842.4356	121080.8390	4.4001	64.8905
		50	5	90798.3141	90273.8551	93429.8498	4.6894	765.2920
200	Генетичний алгоритм	10	2	26301.1330	26246.1629	26760.0835	2.9434	121.2303
		10	3	18985.0622	18946.6891	19203.9998	3.8327	60.7424
		20	2	60275.0170	60270.2159	60446.1909	5.4503	22.2380
		20	3	47551.1650	47239.8000	48280.5360	6.5408	308.3659
		50	3	121077.3704	120843.6980	124189.0043	9.2035	430.1535
		50	5	91264.9009	90289.2492	94417.4837	11.5321	911.9609
	Спіральний пошук	10	2	26257.3625	26246.4028	26332.1371	1.2909	25.8911
		10	3	18947.9393	18946.7903	18949.7167	1.5910	0.6608
		20	2	60270.2954	60270.2175	60270.4388	3.1676	0.0449
		20	3	47396.4682	47239.7043	48162.3611	3.5693	336.5206
		50	3	120894.2064	120843.6477	121018.1391	6.5143	66.4399
		50	5	90729.1056	90269.3624	93430.6666	7.7570	758.4792

Порівняння алгоритмів

- обидва алгоритми знаходять близькі за якістю розв'язки;
- для 10 міст / 2 склади різниця ΔF ≈ 0.03;
- Спіральний пошук має менше σ – стабільніший;
- Спіральний пошук зазвичай швидший за генетичний алгоритм;
- більше складів зменшує F, але збільшує розмірність m = 2k;
- більше міст – більший сумарний попит і витрати.

14

Висновки

1. **Мету досягнуто**
   Досліджено особливості глобальної оптимізації багатоекстремальних функцій, реалізовано генетичний алгоритм й метод спірального пошуку та проведено їх експериментальне порівняння.

2. **Класичні методи обмежені**
   Градієнтний спуск і метод Ньютона залежать від початкової точки й часто збігаються до локальних мінімумів – це обґрунтовує метаевристичний підхід.

3. **Генетичний алгоритм**
   Ефективність залежить від селекції, кросоверу й мутації; найкращі результати – ранговий відбір з помірною мутацією (Швефеля 85 %, Міхалевича 100 %).

4. **Метод спірального пошуку**
   Висока ефективність при 200 точках, $r = 0.95, \theta = \pi/3$; швидший і стабільніший, з меншим стандартним відхиленням.

5. **Прикладна задача**
   Обидва методи придатні для розміщення складів; Спіральний пошук – кращий за швидкістю та стабільністю, генетичний алгоритм – за широтою дослідження простору.

Подальші дослідження

→ **Гібридизація методів**
Поєднання глобального пошуку метаевристик із локальним уточненням градієнтними методами для підвищення точності.

→ **Високорозмірні задачі**
Дослідження поведінки алгоритмів при більшій кількості складів і змінних; адаптивне налаштування параметрів r, θ та мутації.

→ **Розширення прикладних задач**
Урахування вартості відкриття складів, обмежень місткості та реальних дорожніх відстаней замість евклідових.

→ **Інші метаевристики**
Порівняння з алгоритмом рою частинок, диференціальною еволюцією та методом імітації відпалу.

Дякую за увагу!

Ніколаєв Ігор Дмитрович · група КА-21 · Київ, 2026