МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра фундаментальної та прикладної математики

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«ГЕНЕТИЧНІ АЛГОРИТМИ РОЗВ'ЯЗАННЯ ДІОФАНТОВИХ РІВНЯНЬ»**

Виконав: студент 2 курсу, групи 8.1132
спеціальності 113 Прикладна математика
(шифр і назва спеціальності)
освітньої програми Прикладна математика
(назва освітньої програми)
М.С. Музиченко
(ініціали та прізвище)
Керівник завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри загальної математики, доцент, к.ф.-м.н. Спиця О.Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра фундаментальної та прикладної математики

Рівень вищої освіти магістр

Спеціальність 113 прикладна математика

(шифр і назва)

Освітня програма прикладна математика

ЗАТВЕРДЖУЮ

Завідувач кафедри
фундаментальної та прикладної
математики, професор, д.т.н.

_____ Гребенюк С.М.

(підпис)

« ____ » _____ 2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Музиченко Миколі Сергійовичу

(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Генетичні алгоритми розв'язання діофантових рівнянь

керівник роботи (проекту) Гребенюк Сергій Миколайович, д.т.н., професор

(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від «____» _____ 2023 року № _____

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд основних генетичних алгоритмів розв'язання математичних задач
2. Реалізація генетичного алгоритму розв'язання діафантових рівнянь

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

Презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	17.05.2023	
2.	Збір вихідних даних.	9.06.2023	
3.	Обробка методичних та теоретичних	29.06.2023	
	джерел.		
4.	Розробка першого розділу.	21.09.2023	
5.	Розробка другого розділу.	27.10.2023	
6.	Оформлення та нормоконтроль	25.11.2023	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	12.12.2023	

Студент _____ М.С. Музиченко
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.М. Гребенюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця

РЕФЕРАТ

Кваліфікаційна робота магістра «Генетичні алгоритми розв'язання діофантових рівнянь»: 62 с., 8 рис., 7 табл., 16 джерел.

ГЕНЕТИЧНИЙ АЛГОРИТМ, ДІОФАНТОВІ РІВНЯННЯ, ЛОКАЛЬНИЙ МАКСИМУМ, МУТАЦІЯ, ПРОБЛЕМА ЗБІЖНОСТІ, СЕЛЕКЦІЯ, СХРЕЩУВАННЯ, ФУНКЦІЯ ФІТНЕСУ.

Об’єкт дослідження – розв’язання діофантовихрівнянь генетичним алгоритмом.

Мета роботи: розробити програмну реалізацію розв’язання діофантових рівнянь генетичним алгоритмом.

Методи дослідження – моделювання, аналіз, аналогія, порівняння.

У роботі наведено історію та класифікацію діофантових рівнянь. Розглянуто проблему рішення діофантових рівнянь. Наведено приклади основних методів їх розв’язання. Детально розглянуто схему роботи генетичних алгоритмів. Описано принципи задання цільової функції, створення початкової популяції, селекції, схрещування та мутацій. Був запропонований метод рішення діофантових рівнянь за допомогою генетичного алгоритму і розроблено програмну реалізацію. Розглянуто проблему збіжності генетичних алгоритмів і методи її рішення. Вдосконалено базовий алгоритм додаванням випадкового шуму до цільової функції. Протестовано набір різних значень початкових параметрів програми і експериментально визначено оптимальні за швидкодією та збіжністю.

SUMMARY

Master's Qualification Theses "Genetic algorithms for solving Diophantine equations": 62 p., 8 figures, 7 tables, 16 sources.

GENETIC ALGORITHM, DIOPHANTINE EQUATIONS, LOCAL MAXIMUM, MUTATION, CONVERGENCE PROBLEM, SELECTION, CROSSING, FITNESS FUNCTION.

Object of study – solving Diophantine equations by genetic algorithm.

Purpose: to develop a software implementation of solving Diophantine equations by a genetic algorithm.

Research methods – simulation, analysis, analogy, comparison.

The work presents the history and classification of Diophantine equations. The problem of solving Diophantine equations is considered. Examples of the main methods of their solution are given. The working scheme of genetic algorithms is considered in detail. The principles of setting the objective function, creating the initial population, selection, crossing and mutations are described. A method for solving Diophantine equations using a genetic algorithm was proposed and a software implementation was developed.The problem of convergence of genetic algorithms and methods of its solution are considered. Improved the basic algorithm by adding random noise to the objective function. A set of different values of the program's initial parameters was tested and the optimal ones in terms of speed and convergence were determined experimentally.

ЗМІСТ

ВСТУП

Сучасний науковий світ невпинно розширює свої горизонти, висуваючи перед дослідниками та вченими нові завдання та виклики. Однією з ключових галузей математики, яка здобуває все більшу популярність і важливість, є теорія діофантових рівнянь. Ця галузь вивчає цілочисельні рішення алгебраїчних рівнянь та систем рівнянь, що відкриває безмежні можливості для розвитку математичної теорії та її застосувань у різних галузях.

Діофантові рівняння стали об'єктом інтенсивних досліджень завдяки своєму фундаментальному значенню та потужному впливу на розвиток математичної науки. Важливість цієї теорії проявляється у різноманітних аспектах, починаючи від алгебраїчних та арифметичних властивостей цілих чисел і закінчуючи застосуваннями у криптографії, теорії чисел, математичній логіці та інших галузях науки та техніки.

Діофантові рівняння відомі людству з давніх часів. За період, що пройшов з часів Діофанта, математики знайшли розв’язання у величезної кількості діофантових рівнянь, а для багатьох інших довели, що розв’язків у них немає. Для цього довелося розробити велику кількість різноманітних методів, кожен із яких застосовний до рівнянь дуже спеціального виду. Задача, що полягає у знаходженні універсального методу визначення розв'язності довільного алгебраїчного діофантового рівняння, вважалася однією з найважливіших нерозв’язаних задач на початку XX століття. Доказ алгоритмічної нерозв'язності цього завдання зайняв близько двадцяти років і був завершений Юрієм Матіясевичем у 1970 році.

Не слід думати, що із встановленням нерозв'язності даної задачі закрито всі алгоритмічні проблеми, пов'язані з діофантовими рівняннями. Навпаки, тут є багато відкритих проблем, які 21 століття успадкувало від 20-го. Цей напрямок продовжує розвиватися. Розуміння і вивчення

діофантовихрівнянь відкриває нові горизонти для математичних досліджень і може призвести до важливих відкриттів та застосувань у різних сферах науки та техніки.

У даній кваліфікаційній роботі були розглянуті різні методи рішення діофантових рівнянь: повного перебору, розкладання на множники, виділення повного квадрату, нескінченного спуску, алгоритм Евкліда, методи, що ґрунтується на теорії ланцюгових дробів та теорії порівнянь, та інші.

У роботі запропоновано метод рішення діофантових рівнянь за допомогою генетичного алгоритму. Генетичні алгоритми, засновані на імітації природного відбору та еволюційних процесів, в останні десятиліття здобули широке визнання в різних галузях науки і техніки. Актуальність застосування генетичних алгоритмів у розв'язанні діофантових рівнянь визначається кількома факторами.

- комплексність задачі діофантових рівнянь: Діофантові рівняння можуть мати велику кількість цілочисельних розв’язків, і у багатьох випадках вирішення їх аналітично складне або навіть неможливе. Генетичні алгоритми, здатні працювати в умовах великої кількості можливих рішень та нестрого визначених умов, стають ефективним інструментом для знаходження оптимальних або прийнятних рішень;
- гнучкість генетичних алгоритмів: Генетичні алгоритми дозволяють враховувати та оптимізувати різні критерії та обмеження, що може бути важливим у випадку діофантових рівнянь з різноманітними параметрами та умовами. Їх гнучкість робить їх ефективним інструментом для розв'язання складних задач;
- можливість роботи з невизначеністю: Діофантові рівняння часто пов'язані з задачами, де інформація або умови можуть бути невизначеними чи неповними. Генетичні алгоритми можуть

працювати в умовах невизначеності, допомагаючи знаходити рішення в умовах обмеженої чи неповної інформації.

Об’єктом дослідження є методи розв’язання діофантових рівнянь, а предметом дослідження – розв’язання діофантових рівнянь генетичним алгоритмом.

Метою даної роботи є розробити програмну реалізацію розв’язання діофантових рівнянь генетичним алгоритмом, що забезпечувала би високу збіжність при максимальній швидкодії.Виходячи з мети, визначені основні завдання дослідження, а саме:

- розгляд схеми роботи генетичних алгоритмів;
- застосування розглянутої схеми до розв'язання діофантових рівнянь.

Робота складається з двох основних розділів. Перший розділ містить теоретичні відомості про історію вивчення та класифікацію діофантових рівнянь, проблему знаходження універсального методу розв'язання діофантових рівнянь, а також приклади розв’язаня даних рівнянь різними математичними методами. Другий розділ присвячено базовим принципам роботи генетичного алгоритму та застосуванню генетичного алгоритму для знаходження розв’язків діофантових рівнянь.

1 МЕТОДИ РОЗВ’ЯЗАННЯ ДІОФАНТОВИХ РІВНЯНЬ

1.1 Етапи розвитку теорії діофантових рівнянь

Діофантовими рівняннями називають алгебраїчні рівняння чи системи алгебраїчних рівнянь з цілими коефіцієнтами, для яких треба знайти цілі або раціональні розв'язки. При цьому кількість невідомих у рівняннях повинна бути не менше двох (якщо не обмежуватися лише цілими числами) [1].

Діофантові рівняння можна записати у вигляді $P(x_1, x_2, ..., x_n) = 0$, $n \geq 2$, де $P(x_1, x_2, ..., x_n)$ – багаточлен із цілими коефіцієнтами, а всі змінні x_i приймають цілочислові значення [2].

Розв’язати лінійне діофантове рівняння означає встановити таке:

а) чи має воно хоча б одне ціле рішення;

б) звичайно чи нескінченно число його цілих рішень;

в) визначити всі його цілочисленні розв’язки.

Дуже часто діофантові рівняння називають невизначеними.

Діофантові рівняння, як правило, мають багато розв'язків, тому їх називають невизначеними рівняннями.

Діофантові рівняння пов'язані із іменем давньогрецького математика Діофанта Александрійського. Про деталі життя Діофанта Александрійського практично нічого не відомо. З одного боку, Діофант цитує Гіпсикла (II століття до н.е.); з іншого боку, про Діофанта пише Феон Александрійський (близько 350 року н.е.) – звідки можна зробити висновок, що його життя відбувалося в межах цього періоду. Можливе уточнення часу життя Діофанта базується на тому, що його робота "Арифметика" присвячена "достопочтеннішому Діонісію". Вважають, що цей Діонісій –нехто інший, як єпископ Діонісій Александрійський, який жив наприкінці III століття н.е. [3].

В Палатинській антології зберігається епіграма-задача, розв'язання якої наводить на те, що Діофант прожив 84 роки. До нас дійшли 7 книг Діофантаз,

можливо, 13, які були об'єднані в "Арифметику" – збірник задач (їх усього 189), кожна з яких подана розв'язанням та необхідним поясненням. У книзі II розв'язуються задачі, пов'язані з невизначеними рівняннями та системами рівнянь з 2, 3, 4, 5, 6 невідомими ступені не вище другої. Діофант використовував для розв'язання рівнянь різні прийоми. Методи, розроблені у книзі II, Діофант використовував і до більш складних завдань книги III, пов'язаних з системами трьох, чотирьох та більшої кількості рівнянь ступені не вище другої. У книзі IV зустрічаються визначені та невизначені рівняння третьої та вищих ступенів [4].

Індійські математики приблизно з п'ятого століття також розглядали невизначені рівняння першого ступеня. Деякі такі рівняння з двома та трьома невідомими виникли у зв'язку з проблемами, що виникли в астрономії, наприклад, при розгляді питань, пов'язаних з визначенням періодичного повторення небесних явищ. Перше загальне розв’язання рівняння

$$ax + by = c,$$

де *a*, *b*, *c*– цілі числа, зустрічається у індійського мудреця Брахмагупти (близько 625 року) [6].

Ще в Стародавньому Вавилоні займалися пошуками піфагорових трійок – цілих розв’язків рівняння

$$x^2 + y^2 = z^2.$$

Формули, що дозволяють знайти його рішення, були отриманіпіфагорійцями: $x = k^2 - 1, y = 2k, z = k^2 + 1$.

Зазначимо лише, що з $z \leq 100$ рівняння $x^2 + y^2 = z^2$ має 16 примітивних трійок:

(3; 4; 5) (20; 21; 29) (11; 60; 61) (13; 84; 85)

(5; 12; 13)	(12; 35; 37)	(16; 63; 65)	(36; 77; 85)
(8; 15; 17)	(9; 40; 41)	(33; 56; 65)	(39; 80; 89)
(7; 24; 25)	(28; 45; 53)	(48; 55; 73)	(65; 72; 97)

Піфагорова трійка називається примітивною, якщо вона не може бути отримана з якоїсь іншої піфагорової трійки множенням на одне і те ж натуральне число. Таким чином, примітивні піфагорові трійки є взаємно простими. Іншими словами, їх найбільший загальний дільник дорівнює 1.

У цьому ряду, наприклад, відсутні трійки (6; 8; 10), (12; 16; 20), (10; 24; 26). Їх не відносять до примітивних піфагорових трійок, тому що вони містять числа, кратні числам із примітивних трійок.

Одним із яскравих представників класу діофантових рівнянь другого ступеня є рівняння Пелля (його ще називають невизначеним рівнянням Ферма), тобто рівняння:

$$x^2 + ay^2 = 1,$$

де *a*– ціле додатне число, не є повним квадратом.

Перші згадки про рівняння Пелля знайдені в роботах математиків Давньої Греції та Давньої Індії [4]. У «Началах» Евкліда (III ст. е.) міститься рівняння $x^2 - ay^2 = 1$,яке відноситься до невизначених рівнянь. В цій праці для нього наводяться цілочисленні розв’язки. Розв’язання для випадку довільного неквадратного *a* знав давньогрецький математикАрхімед (287 р. до н.е. – 212 р. до н.е.), який поставив перед іншимвченим Еллади – Ератосфеном (276 р. до н.е. – 194 р. до н.е.) відому «Задачу про биків», написану віршами.

Архімед пропонує читачеві знайти кількість биків бога Сонця Геліоса за таких умов:

– у Геліоса було чотири стада, кожна з яких відрізнялася за кольором;

- кількість білих биків дорівнювала = (1/2 + 1/3) темних + рудих биків;
- темних биків = (1/4 + 1/5) рябих + рудих биків;
- рябих биків = (1/6 + 1/7) білих + рудих биків;
- білих корів= (1/3 + 1/4) темного стаду;
- темних корів = (1/4 + 1/5) рябого стаду;
- рябих корів = (1/5 + 1/6) рудого стаду;
- рудих корів = (1/6 + 1/7) білого стаду.

Після цього Архімед пропонує знайти кількість биків і корів різного кольору, вказуючи, що той у кого це вийде не є невігласом [7].

Розв’язання першої частини задачі зводиться до системи 8 алгебраїчних рівнянь. Якщо її вирішити, то виявиться, що мінімальна кількість голів скота у Геліоса становить 50 389 082.

Друга частина завдання включає додаткові умови:

- кількість білих і темних бугаїв – квадратне число;
- кількість строкатих і рудих бугаїв – трикутне число.

Той, хто зможе за цих умов визначити кількість голів худоби у стадах Геліоса, на думку Архімеда, є мудрець [7].

Друга частина завдання, тобто пошук рішення, яке б задовольняло умовам першої та другої частини, зводиться до рівняння Пелля. Її розв’язання було опубліковано в 1880 [8]. Загальна кількість бугаїв приблизно дорівнює 7.76×10^{206544}. Щоб записати всі 206545 цифр необхідно 660 сторінок з 2500 знаків на кожній. Вперше точне числове значення розв'язання задачі про биків було роздруковано в 1965 році з використанням комп'ютерної техніки [9].

Загальний спосіб розв'язання рівняння – так званий "циклічний метод" – присутній у працях індійського математика XII століття Брахмагупти, проте без доведення, що цей метод завжди призводить до розв'язку. Узагальнену задачу сформулював французький математик П'єр Ферма, тому в Франції це рівняння називається "рівнянням Ферма". Сучасне ж

найменування рівняння Пеллявиникло завдяки Л. Ейлеру, який помилково приписав їх авторство Джону Пеллю, англійському математику, який цим рівнянням ніколи не займався [4].

У 1624 році вийшла книга французького математика Баша де Мезір'яка "Problemes plaisans et delectables que se font par les nombres". Баша де Мезір'як для розв'язання рівняння $ax + by = c$ застосував процес, який зводиться до послідовного обчислення неповних часток та розгляду відповідних дробів. Після Баша де Мезір'яка в XVII та XVIII століттях різні правила для розв'язання невизначеного рівняння першого ступеня з двома невідомими давали Роль, Ейлер, Сондерсон та інші математики [4].

Ланцюгові дроби для розв'язання таких рівнянь були застосовані Лагранжем. Невизначені рівняння першого ступеня почали записувати та розв'язувати у формі порівняння значно пізніше, починаючи з Гаусса.

У 1630 р. французький математик П'єр Ферма (1601 – 1665) сформулював гіпотезу, яку називають великою теоремою Ферма: "Рівняння

$$x^n + y^n = z^n$$

для натурального $n \geq 3$ не має розв'язків у натуральних числах". Ферма не довів свою теорему в загальному випадку, але відомий його запис на полях "Арифметики" Діофанта:"...неможливо куб записати у вигляді суми двох кубів, або парну степінь у вигляді суми таких самих степенів, або взагалі будь-яке число, яке є степенем більше, ніж другий, неможливо записати у вигляді суми двох таких же степенів. У мене є справжній дивовижний доказ цього твердження, але поля ці занадто вузькі, щоб його умістити".

Пізніше в паперах Ферма було знайдено доказ його теореми для $n = 4$. З того часу більше 300 років математики намагалися довести велику теорему Ферма. У 1770 р. Л.Ейлер довів теорему Ферма для $n = 3$; у 1825 р. Лежандр (1752 – 1833) і Діріхле (1805 – 1859) – для $n = 5$. Доведення великої теореми Ферма в загальному випадку не вдавалося довгі роки. І тільки в 1995 р.

Ендрю Уайлз довів цю теорему. Протягом віків математики сподівалися знайти загальний спосіб розв'язання будь-якого діофантового рівняння. Однак у 1970 р. ленінградський математик Ю.В. Матіясевич довів, що такого загального способу не існує.

1.2 Проблема розв’язання діофантових рівнянь

У 1900 році представники багатьох країн зібралися у Парижі для участі у Другому Міжнародному Конгресі математиків. Видатний німецький математик Давид Гільберт виступив там із доповіддю, яка називалася "Математичні проблеми". У вступній частині доповіді Гільберт сказав, що, на його думку, математика – це наука, яку рухають уперед невирішені завдання. Потім він перерахував проблеми, вирішення яких, на його думку, важливе для подальшого прогресу математики [10]. Станом на 2023 рік вирішено 16 із 23 проблем Гільберта.

Проблема під номером 10 присвячена діофантовим рівнянням. За період, що пройшов з часів Діофанта, математики знайшли розв’язання у величезної кількості діофантових рівнянь, а для багатьох інших довели, що розв’язків (у цілих або натуральних числах) у них немає. Для цього довелося розробити велику кількість різноманітних методів, кожен із яких застосовний до рівнянь дуже спеціального виду.

Яскравим прикладом тут може бутиВелика теорема Ферма, яка чекала свого доказу понад три сторіччя. Гільберт вважав за потрібне покінчити з різнобоєм методів розв'язання діофантових рівнянь, знайшовши єдиний, універсальний метод, який можна було б застосувати добудь-якого діофантового рівняння та дізнатися, чи є у нього розв’язок.

У сучасних термінах можна сказати, що Гільберт просив знайти алгоритм вирішення діофантових рівнянь. У своєму формулюванні проблеми він, однак, не використав саме це слово. Справа в тому, що в той час у

математиці ще не було суворого визначення алгоритму. Таке визначення було вироблено набагато пізніше, у 30-ті роки XX століття. У наші дні десяту проблему Гільберта можнарозуміти як таке завдання: написати якоюсь мовою програмування програму, яка, отримавши на вхід довільне діофантове рівняння, через кінцевий час видала б відповідь, скажімо, друкуючи число 0, якщо рівняння розв’язків не має, і що-завгодно, відмінне від 0, у протилежному випадку.

Сьогодні відомо, що написати таку програму не можна. Десята проблема Гільберта не має рішення, яке вимагав Гільберт, а доказ цього факту вважається негативним рішенням цієї проблеми.

Нерозв'язність 10-ї проблеми Гільберта означає, що якщо зафіксуємо деяку мову програмування Я, то для будь-якої програми Р цією мовою, яка гіпотетично вирішує цю проблему, знайдеться конкретне діофантове рівняння, на якому програма помилиться. А саме,

- або програма надрукує 0, а рівняння має розв’язок,
- або рівняння розв’язків не має, а програма друкує щось, відмінне від нуля, або ж не друкує нічого (просто зупинившись чи працюючи необмежено довго).

Яка може бути користь від цього доказу неможливості? Академік Матіясевич порівняв ситуацію із законом збереження енергії, який унеможливлює побудову "вічного двигуна", позбавляючи цим винахідників від свідомо марної трати часу на його побудови, а патентні бюро – від розгляду заявок горе-винахідників. Аналогічно доведена нерозв'язність 10-ої проблеми Гільберта дає математикам "моральне право" більше не витрачати час на спроби знайти універсальний метод розв’язання діофантових рівнянь [11].

Однак у доказу нерозв'язності проблеми є й інша користь, важливіша, ніж саме її вирішення – це нові ідеї та методи, розроблені для отримання її вирішення. На початку 50-х років американський математик Мартін Дейвіс висунув гіпотезу, яка стверджувала, що одне з основних понять інформатики

– поняття перелічуваної множини – збігається з теоретико-числовим поняттям діофантової множини, що виникає при вивченні параметричних діофантових рівнянь. Гіпотезу Дейвіса було доведено через два десятиліття після її висування, останній крок у 1970 році зробив радянський математик Юрій Матіясевич.

Гіпотеза Дейвіса, ставши теоремою, дозволила встановити нерозв'язність багатьох інших проблем та отримати інші цікаві наслідки. Наприклад, на основі діофантових рівнянь було запропоновано нову криптографічну систему.

Академік Юрій Матіясевич наголосив, що не слід думати, що із встановленням нерозв'язності 10-ої проблеми Гільберта закрито всі алгоритмічні проблеми, пов'язані з діофантовими рівняннями [12]. Навпаки, тут є багато відкритих проблем, які 21 століття успадкувало від 20-го. Цей напрямок продовжує розвиватися.

1.3 Методи розв’язання діофантових рівнянь

Можна умовно виділити такі способи розв'язання діофантових рівнянь:

- метод повного перебору всіх можливих значень змінних, що входять до рівняння;
- метод розкладання на множники;
- метод, заснований на вираженні однієї змінної через іншу та виділенні цілої частини дробу;
- метод, що ґрунтуються на виділенні повного квадрата;
- метод розв'язання рівняння з двома змінними як квадратного щодо однієї зі змінних;
- метод, заснований на оцінці виразів, що входять до рівняння;
- метод нескінченного спуску;
- метод, що ґрунтується на алгоритмі Евкліда;

– метод, що ґрунтується на теорії ланцюгових дробів;

– метод, що ґрунтується на теорії порівнянь та ін.

Розглянемо докладніше застосування вказаних вище методів на конкретних прикладах.

1.3.1 Метод повного перебору всіх можливих значень змінних, що входять до рівняння

Приклад 1 Знайти множину всіх пар натуральних чисел, якіє розв'язками рівняння:

$$49x + 51y = 602,$$

Виразимо із рівняння змінну x через y,$x = (602 - 51y)/49$, оскільки x та y – натуральні числа, то $x \geq 1$, $602 - 51y \geq 49$, $y \leq 553/51$. Повний перебір варіантів показує, що натуральними розв'язками рівняння є x=5, y=7.

Відповідь: (5; 7).

1.3.2 Метод розкладання на множники

Розглянемо випадок, як у рівняннях можна застосувати формулу різниці квадратів чи інший спосіб розкладання на множники.

Приклад 2 Знайти всі цілі розв'язки рівняння:

$$x^2 - 3xy + 2y^2 = 3.$$

Розкладемо ліву частину цього рівняння на множники:

$$x^2 - 3xy + 2y^2 = (x - y)(x - 2y).$$

Маємо: $(x-y)(x-2y)=3$. Оскільки число 3 можна представити у вигляді добутку цілих чисел з урахуванням порядку чотирма способами:

$$3 = 1 \cdot 3 = 3 \cdot 1 = (-1) \cdot (-3) = (-3) \cdot (-1),$$

то отримуємо сукупність чотирьох систем для знаходження значень змінних:

$$\begin{cases} x-y=1, \\ x-2y=3; \end{cases}$$

$$\begin{cases} x-y=3, \\ x-2y=1; \end{cases}$$

$$\begin{cases} x-y=-1, \\ x-2y=-3; \end{cases}$$

$$\begin{cases} x-y=-3, \\ x-2y=-1. \end{cases}$$

Цілими розв’язками даних рівнянь є відповідно пари:(−1;−2), (5;2), (1;2), (−5; −2).

Відповідь: (−1; −2), (5; 2), (1; 2), (−5; −2).

1.3.3 Метод, заснований на вираженні однієї змінної через іншу та виділенні цілої частини дробу

Приклад 3 Розв'язати у цілих числах рівняння:

$$3x+2y=7.$$

Переписавши рівняння у вигляді $2(x+y)=7-x$, робимо висновок, що $7-x$ кратно 2, тобто $7-x=2k, k\in Z$. Отже, $x=7-2k$, та звихідного рівняння знаходимо $y=3k-7$. Отже, всі пари виду $(7-2k;3k-7), k\in Z$є розв’язками вихідного рівняння.

Відповідь: $(7-2k; 3k-7), k \in Z$.

1.3.4 Метод, що ґрунтуються на виділенні повного квадрата

Приклад 4 Знайдіть усі цілі розв'язки рівняння:

$$x^2 - 6xy + 13y^2 = 29.$$

Перетворимо ліву частину рівняння, виділивши повні квадрати:

$$x^2 - 6xy + 13y^2 = (x^2 - 6xy + 9y^2) + 4y^2 = (x-3y)^2 + (2y)^2 = 29,$$

отже, $(2y)^2 \leq 29$.

Отримуємо, що *y* може дорівнювати 0; ±1; ±2:

– *y* = 0:

$(x-0)^2 = 29$. Немає розв'язків у цілих числах;

– *y* = −1:

$(x+3)^2 + 4 = 29$,

$(x+3)^2 = 25$,

$x + 3 = 5$ або $x + 3 = -5$

$x = 2$ або $x = -8$:

– *y* = 1:

$(x+3)^2 + 4 = 29$,

$(x-3)^2 = 25$,

$x - 3 = 5$ або $x - 3 = -5$

$x = 8$ або $x = -2$;

– *y* = −2:

$(x+6)^2 + 16 = 29$, $(x+6)^2 = 13$. Немає розв'язків у цілих числах;

– *y* = 2:

$(x-6)^2 + 16 = 29$, $(x-6)^2 = 13$. Немає розв'язків у цілих числах.

Відповідь: (2; −1); (−8; −1); (8; 1); (−2; 1).

1.3.5 Метод розв'язання рівняння з двома змінними як квадратного щодо однієї зі змінних

Приклад 5 Розв'язати рівняння у цілих числах:

$$x^2 - xy + y^2 = x + y.$$

Перетворимо рівняння

$$x^2 - x(y+1) + y^2 - y = 0.$$

Розглянемо його як квадратне щодо *x*:

$$D = (y+1)^2 - 4(y^2 - y) = -3y^2 + 6y + 1 \geq 0,$$
$$3(y-1)^2 \leq 4,$$
$$(y-1)^2 \leq 2.$$

Перевірка для *y* = 0; 1; 2 дає шукані розв'язки.

Відповідь: (0; 0), (0; 1), (1; 0), (1; 2), (2; 1), (2; 2).

1.3.6 Метод, заснований на оцінці виразів, що входять до рівняння

Приклад 6. Знайдіть усі пари (a, b) натуральних чисел a та b, що задовольняють рівності

$$45^a - b^b = 1998.$$

Числа 1998 і 45^a кратні 3, тому число b^bмає бути кратним 3, тобто $b = 3k$ для деякого натурального k. Тоді $45^a = 1998 + 3^3k \cdot k^{3k}$. Обидва доданки у правій частині отриманої рівності кратні $3^3 = 27$, тому число 45^aтакож має бути кратно 27, тобто $a \geq 2$. Отже, $45^a = 5^a9^a$ ділиться на $9^2 = 81$. Якщо $k \geq 2$, тоді число $3^3k \cdot k^{3k}$ ділилося б на 81, тому і число 1998 має ділитися на 81, що невірно. Отже, $k = 1$, тобто $b = 3$, і тому $5^a = 1998 + 3^3 = \cdots = 2025 = 45^2$, або $a = 2$.

Відповідь: (2; 3).

1.3.7 Метод нескінченного спуску

Методом нескінченного спуску називають міркування, які проходять за наступною схемою: припустивши, що у задачі є розв'язки, будуємо деякий нескінченний процес, тоді як у самому сенсі завдання цей процес має чимось закінчитися.

Часто метод нескінченного спуску застосовується у простішій формі. Припустивши, що вже досягнуто кінця, бачимо, що зупинитися не можливо.

Приклад 7 Доведемо нерозв'язність у натуральних числах рівняння:

$$8x^4 + 4y^4 + 2z^2 = t^4.$$

Припустимо, що розв'язок є і $x = m$, $y = n$, $z = p$, $t = q$ – розв'язок з найменшим можливим x.

З виду рівняння випливає, що $q = 2q_1$.

Підставимо розв'язок у рівняння і скоротимо на 2:

$$4m^4 + 2n^4 + p^4 = 8q_1^4.$$

Отримуємо, що $p = 2p_1$, отже,

$$2m^4 + n^4 + 8p_1^4 = 4q_1^4.$$

Аналогічно, $n = 2n_1$,

$$m^4 + 8n_1^4 + 4p_1^4 = 2q_1^4$$

і $m = 2m_1$,

$$8m_1^4 + 4n_1^4 + 4p_1^4 = q_1^4.$$

Отже, $x = m_1$, $y = n_1$, $z = p_1$, $t = q_1$– також розв’язок нашого рівняння.

Але $m_1<m$, що суперечить вибору вихідного розв’язку. Значить, розв’язків немає.

Відповідь: розв’язків немає.

З доказу видно, що застосування методу спуску в цьому прикладі ґрунтується на тому факті, що будь-яка непорожня множина натуральних чисел має мінімальний елемент. Іншими словами, метод нескінченного спуску полягає в побудові нескінченної послідовності спадних цілих додатних чисел. Оскільки спадна послідовність цілих додатних чисел має лише кінцеве число членів, отримуємо протиріччя.

1.3.8 Метод, що ґрунтується на алгоритмі Евкліда

Приклад 8 Розв’яжіть рівняння

$$5x - 3y = -1.$$

(5, 3, −1) = 1. Це діофантове рівняння, НСД(5, 3) = 1. Дане рівняння можна розв'язати в цілих числах. Знайдемо розв’язок (x_0, y_0) за допомогою алгоритму Евкліда:

$$5/3 \Rightarrow 5 = 3 \cdot 1 + 2; 3 = 2 \cdot 1 + 1.$$

Знайдемо лінійне подання НСД:

$$1 = 3 - 2 \cdot 1 = 3 - (5 - 3 \cdot 1) = 3 \cdot 2 - 5 \cdot 1.$$

$$5 \cdot (-1) - 3 \cdot (-2) = 1;$$

$$5 \cdot 1 - 3 \cdot 2 = 1;$$

$$x_0 = 1, y_0 = 2;$$

$$\begin{cases} x = 1 + 3t, \\ y = 2 - 5t, t \in Z. \end{cases}$$

Відповідь: (1 + 3t, 2 − 5t), $k \in Z$.

1.3.9 Метод, що ґрунтується на теорії ланцюгових дробів

Приклад 9 Розв'язати рівняння:

$$45x - 13y = 2.$$

(45; −13; 2) = 1 і (45; −13) = 1, отже це діофантове рівняння і його можна розв’язати в цілих числах:

$$\frac{a}{b} = \frac{45}{13};$$

$$\frac{45}{13} = [3; 2; 6].$$

Розрахунок відповідних дробів приведено у таблиці 1.1.

Таблиця 1.1 – Таблиця розрахунку відповідних дробів

k	0	1	2
a_k	3	2	6
p_k	3	7	45
q_k	1	2	13

$$\frac{45}{13} = \frac{7}{2} = \frac{(-1)^1}{13 \cdot 2}.$$

$$45 \cdot 2 - 13 \cdot 7 = -1,$$

$45 \cdot (-4) - 13 \cdot (-14) = 2 \Rightarrow x_0 = -4$, $y_0 = -14$, тому

$$\begin{cases} x = -4 - 13t, \\ y = -14 - 45t, t \in Z. \end{cases}$$

Відповідь: $(-4 - 13t, -14 - 45t)$, $k \in Z$.

1.3.10 Метод, що ґрунтується на теорії порівнянь

Приклад 10 Розв'язати рівняння

$$14x - 10y = 6.$$

$(14, 10, 6) \neq 1$, $7x - 5y = 3$ – діофантове рівняння, тому що $(7, 5, 3) = 1$, а оскільки $(7, 5) = 1$, то $\exists x, y \in Z$ – розв’язок даного діофантового рівняння.

$y = (7x-3)/5 \in Z \Leftrightarrow (7x-3)/5 \Leftrightarrow 7x \equiv 3 (\mathrm{mod}\ 5)$, оскільки $(7, 5) = 1$, то порівняння має єдиний розв’язок. $2x \equiv 8 (\mathrm{mod}\ 5)$, $x \equiv 4 (\mathrm{mod}\ 5)$, $x = 4 + 5t$, $t \in Z$.

Відповідь: $\begin{cases} x = 4 - 5t, \\ y = 5 - 7t, t \in Z. \end{cases}$

2 ГЕНЕТИЧНІ АЛГОРИТМИ ДЛЯ РОЗВ'ЯЗАННЯ ДІОФАНТОВИХ РІВНЯНЬ

2.1 Теорія генетичних алгоритмів

Генетичні алгоритми – це клас алгоритмів пошуку, побудованих на ідеї, подібній до принципів природного відбору в генетиці. Вони об'єднують у собі принцип збереження найперспективніших рішень та структурований обмін інформацією, у якому є елемент випадковості. Даний алгоритм є евристичним алгоритмом пошуку, що використовується для розв'язання задач оптимізації та моделювання шляхом випадкового підбору, комбінування та варіації параметрів, що шукаються, з використанням механізмів, аналогічних природному відбору в природі. Вони є різновидом еволюційних обчислень, за допомогою яких вирішуються оптимізаційні завдання з використанням таких методів природної еволюції, як успадкування, мутації, відбір і кросинговер. Відмінною особливістю генетичного алгоритму є акцент на використанні оператора схрещування, який проводить операцію рекомбінації розв'язків-кандидатів, роль якої аналогічна ролі схрещування в живій природі[13].

Завдання даного алгоритму формалізується таким чином, щоб його розв'язок міг бути закодований у вигляді вектору (генотипу) генів, де кожен ген може бути бітом, числом або якимось іншим об'єктом. У класичних реалізаціях генетичного алгоритму (ГА) передбачається, що генотип має фіксовану довжину. Однак є варіації ГА, вільні від цього обмеження.

Деяким, зазвичай випадковим, чином створюється безліч генотипів початкової популяції. Вони оцінюються з використанням функції пристосованості (її часто називають функцією фітнесу), в результаті чого з кожним генотипом асоціюється певне числове значення, яке визначає наскільки добре фенотип, який він описує, вирішує поставлене завдання.

З отриманої множини розв’язків («покоління») з урахуванням значення пристосованості вибираються розв’язки (зазвичай найкращі особини мають більшу ймовірність бути обраними), до яких застосовуються «генетичні оператори» (у більшості випадків схрещування і мутація), результатом чого є отримання нових розв’язків, після чого також обчислюється значення їх пристосованості. Потім проводиться відбір кращих розв’язків з отриманих особин у наступне покоління [14].

Цей набір дій повторюється ітеративно, моделюючи еволюційний процес, що триває кілька життєвих циклів (поколінь), доки не буде виконано критерій зупинки алгоритму. Таким критерієм може бути:

– знайдення глобального, чи субоптимального розв’язку;
– вичерпання числа поколінь, відпущених на еволюцію;
– вичерпання часу, відпущеного на еволюцію.

Таким чином, можна виділити такі етапи генетичного алгоритму:

– Задання цільової функції фітнесу для особин популяції;
– Створення початкової популяції;
– Відбір (селекція);
– Розмноження (схрещування) і/або мутація;
– Обчислення значення функції фітнесу усіх особин.

Після чого, якщо виконуються умови зупинки, то вихід із циклу, а інакше перехід до селекції нового покоління.

Алгоритм генетичного алгоритму у вигляді схеми представлено на рис. 2.1.

Розглянемо детальніше вищеназвані стадії.

Задання цільової функції

Функція фітнесу – це тип цільової функції, яка використовується для підсумовування, як єдиного показника якості, того, наскільки данийпроєктний розв’язок наближений до досягнення встановлених цілей.Функція має бути адекватно заданою. Це означає, що для успішного

пошуку необхідно, щоб розподіл значень співпадав із розподілом реальної якості розв’язків.

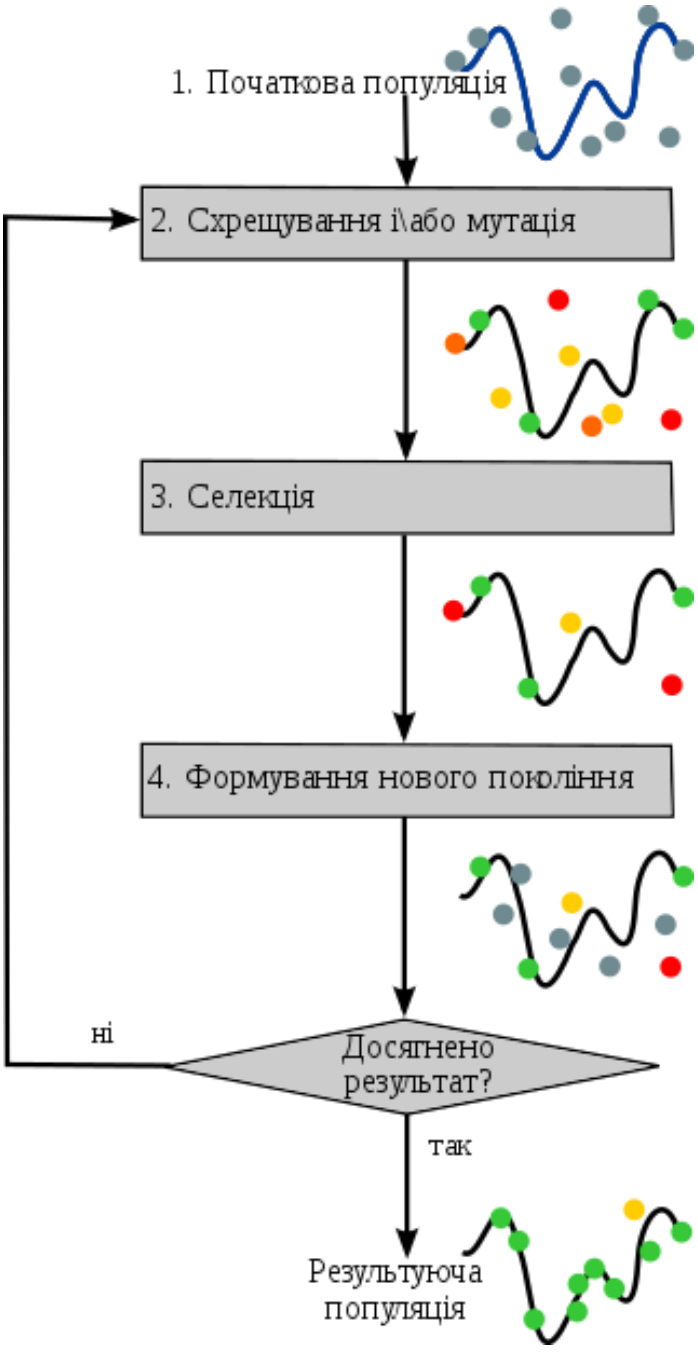

Рисунок 2.1 – Схема роботи генетичного алгоритму

При виборі функції фітнесу важливо стежити, щоб її рельєф був гладким, оскільки в інакшому випадку алгоритм не вибереться с локального максимуму.

Функція повинна мати різноманітний рельєф без великих «пласких» ділянок. Тобто, незважаючи на те, що розв’язки різняться, вони мають однакову оцінку, з чого виходить, що алгоритм не має можливості вибрати найкращий розв’язок, вибрати напрямок подальшого розвитку. Ця проблема ще згадується як «проблема поля для гольфу», де весь простір абсолютно однаковий, за винятком лише однієї точки, і є оптимальним розв’язком – у цьому випадку алгоритм просто зупиниться або блукатиме випадково.

Функція пристосованості має вимагати мінімум ресурсів. Оскільки це найчастіше використовувана деталь алгоритму, вона істотно впливає на його швидкість роботи [15].

Створення початкової популяції

Перед першим кроком потрібно випадково створити початкову популяцію. Напершому кроці можна особливо не намагатися зробити занадто пристосованих особин, достатньо, щоб вони відповідали формату особин популяції, і на них можна було підрахувати функцію фітнесу. Підсумком першого кроку є популяція, що складається з N особин.

Селекція

На етапі відбору потрібно з усієї популяції вибрати певну її частку, яка залишиться на цьому етапі еволюції. Імовірність виживання особини повинна залежати від значення функції фітнесу. Сама частка тих, хто вижив, зазвичай є параметром генетичного алгоритму, і її просто задають заздалегідь. Існують різні способи проводити відбір.

- турнірна селекція – спочатку випадково вибирається встановлена кількість особин (зазвичай дві), а потім з них вибирається особина з кращим значенням функції пристосованості;
- метод рулетки – ймовірність вибору особини тим ймовірніше, чим краще її значення функції пристосованості;
- метод ранжування – ймовірність вибору залежить від місця у списку особин, відсортованому за значенням функції пристосованості.

Розмноження (схрещування)

Розмноження в генетичних алгоритмах вимагає для виробництва нащадку кількох батьків, зазвичай двох.

Можна виділити кілька операторів вибору батьків:

- панміксія – обидва батьки вибираються випадково, кожна особина популяції має рівні шанси бути обраною;
- інбридинг – перший батько вибирається випадково, а другим вибирається такий, який найбільше схожий на першого батька;
- аутбридинг – перший батько вибирається випадково, а другим вибирається такий, який найменш схожий на першого батька.

Інбридинг та аутбридинг бувають у двох формах: фенотипної та генотипної. У разі фенотипної форми схожість вимірюється в залежності від значення функції пристосованості (чим ближче значення цільової функції, тим особини більш схожі), а у випадку генотипної форми схожість вимірюється в залежності від представлення генотипу (чим менше відмінностей між генотипами особин, тим особини більш схожі)[16].

Розмноження у різних алгоритмах визначається по-різному – воно, звичайно, залежить від представлення даних. Головна вимога до розмноження – щоб нащадок чи нащадки мали можливість успадкувати риси обох батьків, «змішавши» їх у будь-який спосіб.

Зазвичай особини для розмноження вибираються з усієї популяції, а не з елементів, які вижили на попередньому кроці (хоча останній варіант теж має право на існування).Робиться те тому, що головний недолік багатьох генетичних алгоритмів – відсутність різноманітності в особинах. Досить швидко виділяється один-єдиний генотип, який є локальним максимумом, а потім всі елементи популяції програють йому відбір, і вся популяція «забивається» копіями цієї особини.

Існують різні способи боротьби з таким небажаним ефектом; один з них – вибір для розмноження не найпристосованіших, але взагалі всіх

особин. Однак такий підхід змушує зберігати всіх особин, що існували раніше, що збільшує обчислювальну складність завдання. Тому часто застосовують методи відбору особин для схрещування таким чином, щоб «розмножувалися» не тільки найпристосованіші, але й інші особини, що мають погану пристосованість. За такого підходу для різноманітності генотипу зростає роль мутацій.

До мутацій, якщо вони присутні, відноситься все те саме, що і до розмноження: є деяка частка мутантів, що є параметром генетичного алгоритму, і на кроці мутацій потрібно вибрати частку особин, а потім замінити їх відповідно до заздалегідь визначених операцій мутації.

2.2 Реалізація генетичного алгоритму для розв'язання прикладних завдань

Розглянемо детальніше принцип роботи генетичного алгоритму для розв'язання діофантових рівнянь, розберемо проблеми його реалізації та шляхи їх вирішення на практичному прикладі на мові програмування C++.

2.2.1 Реалізація генетичного алгоритму для розв'язання діофантових рівнянь

Для прикладу будемо розв'язувати в натуральних числах рівняння

$$3a_1 + 5a_2 + 11a_3 + 17a_4 + 23a_5 = 1171.$$

Побудуємо таблицю $6 \cdot POPULATION\_SIZE$, де 6 – кількість змінних у рівнянні плюс вільний член, а *POPULATION_SIZE* – розмір популяції. Розмір популяції зручніше задавати парним числом, щоб було зручно ділити популяцію навпіл для селекції більш успішних особин. Для прикладу

допустимо розмір популяції рівний 10. Заповнюємо перших п’ять стовпчиків випадковими числами від 1 до *MAX_INITIAL_VALUE* (у коді проєкту дорівнює 5), а шостий, що є вільним членом рівняння, рахуємо. У таблиці 2.1 представлено першу популяцію.

Таблиця 2.1 – Перша популяція

a_1	a_2	a_3	a_4	a_5	b
3	1	1	4	3	162
5	5	4	4	2	198
4	4	2	1	5	186
3	4	5	5	3	238
1	2	2	4	2	149
3	4	4	3	1	147
1	1	3	4	5	224
2	2	1	5	3	181
1	4	1	1	1	74
2	1	3	2	5	193

Розв’язанням рівняння є знаходження таких коренів, щоб їх вільний член співпадав з вільним членом заданого рівняння. Тепер треба якісно оцінити, наскільки той чи інший набір коренів близький до розв’язку рівняння. Іншими словами, треба задати функцію пристосованості (функцію фітнесу), максимум якої є розв’язком рівняння, а її значення чисельно визначає якість індивіду.

У даному проєкті функція фітнесу виглядає так:

```
intget_fitness(vector<int>&x) {
intsum = 3 * x[0] + 5 * x[1] + 11 * x[2] + 17 * x[3] + 23 * x[4];
inttarget = 1171;
intfitness = -abs(sum - target);
```

```
return fitness;
}
```

Повний код проєкту представлено у ДодаткуА.

Задання від’ємного модулю різниці між отриманим вільним членом (контрольною сумою) індивіда та вільним членом рівняння задовольняє поставленим вимогам, оскільки в такому випадку максимумом функції є нуль і тоді ці корені є розв’язком, а ближчі до нуля значення відповідають більш пристосованим індивідам. До того ж обчислення функції фітнесу таким чином потребує мало ресурсів, так як виконуються прості операції над цілими числами типу Integer.Якщо у результаті обчислення функції фітнесу її значення для якогось індивіда дорівнює 0, то програма зупиняє цикл і виводить отримані значення змінних, що є розв’язком рівняння.

Тепер розрахуємо значення цільової функції для отриманих раніше випадкових індивідів першого покоління. Розрахунок представлено у табличній формі у табл. 2.2.

Таблиця 2.2 – Пристосованість першої популяції

a_1	a_2	a_3	a_4	a_5	*fitness*
3	1	1	4	3	-1009
5	5	4	4	2	-973
4	4	2	1	5	-985
3	4	5	5	3	-933
1	2	2	4	2	-1022
3	4	4	3	1	-1024
1	1	3	4	5	-947
2	2	1	5	3	-990
1	4	1	1	1	-1097
2	1	3	2	5	-978

Тепер відсортуємо популяцію за зростанням, вибравши п’ять кращих індивідів для наступного покоління. Їх буде продубльовано у другу половину таблиці, де з ними відбудеться процес мутації. В даній реалізації алгоритму мутацією є зміна на 1 або –1 в одній випадковій змінній. Оскільки рівняння розв’язується в натуральних числах, треба додатково обробити випадок, коли змінна дорівнює одиниці, щоб у результаті мутації не отримати 0.

У даному проєкті функція мутації задана таким чином:

```
voidmutate_individual(Individual&individual) {
intindex = rand() % VARIABLE_COUNT;
intsign = rand() % 2;
if(sign) {
individual.variables[index]++;
   } else {
individual.variables[index]--;
   }
   // Якщо змінна нуль - перетворити на 2, наче ми додали 1, а не
відняли
if(individual.variables[index] == 0) {
individual.variables[index] = 2;
   }
```

Результат першої селекції та мутації, який є другим поколінням, представлено у таблиці 2.3. Вертикальною лінією розділено найкращих індивідів попереднього покоління та їх нащадків після мутації.

Після отримання нового покоління і обчислення значень функції фітнесу її індивідів, виконується перевірка того, чи знайдені корені заданого рівняння. У проєкті перевірка і подальший вивід коренів відбувається наступним чином:

```
// Перевірити, чи знайшли ми розв'язок
for(int i = 0; i < POPULATION_SIZE; i++) {
if (population[i].true_fitness == 0) {
cout<< "Foundsolutionatstep " <<step_id<< endl;
for (int j = 0; j < VARIABLE_COUNT; j++) {
cout<<population[i].variables[j] << " ";
        }
cout<< endl;
return 0;
      }
    }
```

Таблиця 2.3 – Друга популяція

a_1	a_2	a_3	a_4	a_5	*fitness*
3	4	5	5	3	-933
1	1	3	4	5	-947
5	5	4	4	2	-973
2	1	3	2	5	-978
4	4	2	1	5	-985
3	4	5	6	3	-916
1	1	3	4	4	-970
5	5	5	4	2	-962
2	1	2	2	5	-989
5	4	2	1	5	-982

У випадку, якщо на даній ітерації розв’язку не знайдено, повторюється процес селекції та мутації доти, поки або не буде знайдено розв’язок, або кількість ітерацій не досягне наперед заданого значення *MAX_STEPS*, після якого будемо вважати, що алгоритм зайшов у глухий кут і ніколи з нього не

вийде. У даній реалізації алгоритму *MAX_STEPS* = 100000. Можливо, після деякої скінченної кількості ітерацій, наприклад, двохсот тисяч або мільйона, алгоритм і знайде шукані корені, проте таке розв'язання важко назвати оптимальним.

2.2.2 Проблеми збіжності генетичного алгоритму

У багатьох завданнях генетичні алгоритми мають тенденцію збігатися до локального, а не глобального максимуму даної задачі. Цей ефект показано на рис. 2.2. Це означає, що вони не знають, яким чином пожертвувати короткочасною високою пристосованістю для досягнення довгострокової вигоди.

Імовірність цього залежить від форми ландшафту задачі: окремі проблеми можуть мати виражений напрямок до глобального максимуму, тоді як інші можуть вказувати напрямок фітнес-функції на локальний максимум.

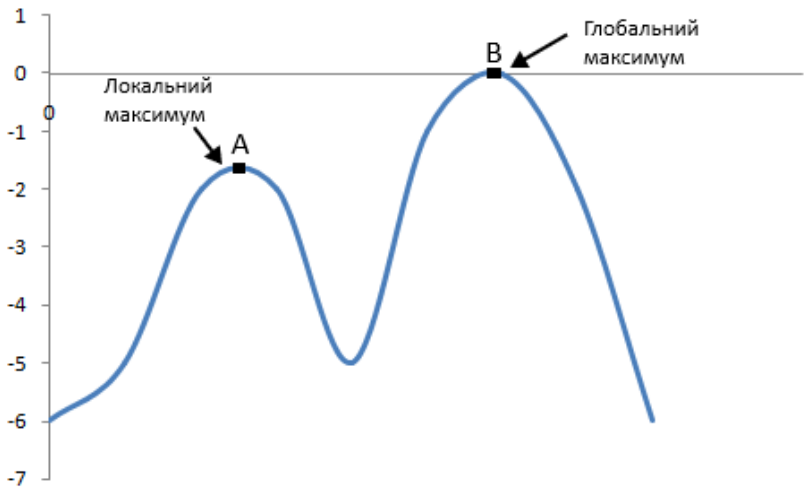

Рисунок 2.2 – Ілюстрація локального максимуму

Покажемо на прикладі глухий кут у роботі генетичного алгоритму, що прийшов до локального максимуму. Нехай у популяції на деякому етапі з'явиться індивід зі змінними 100, 9, 8, 19 і 18, тоді його функція фітнесу дорівнює –1. Параметри індивида представлено у табл. 2.4.

Таблиця 2.4 – Індивід, що прийшов у локальний максимум

a_1	a_2	a_3	a_4	a_5	*fitness*
100	9	8	19	18	-1

Виконаємо вручну всі можливі мутації, додавши або віднявши одиницю від кожної зі змінних. Усі можливі нащадки представлені у таблиці 2.5.

Таблиця 2.5 – Нащадки індивіда, що прийшов у локальний максимум

a_1	a_2	a_3	a_4	a_5	*fitness*
101	9	8	19	18	-2
99	9	8	19	18	-4
100	10	8	19	18	-4
100	8	8	19	18	-6
100	9	9	19	18	-10
100	9	7	19	18	-12
100	9	8	20	18	-16
100	9	8	18	18	-18
100	9	8	19	19	-22
100	9	8	19	17	-24

Як бачимо, функція фітнесу усіх нащадків даного індивіда менша, ніж його власна. Тобто, будь-яка мутація приводить до гіршого результату, ніж уже є у його пращура. Це і є локальний максимум функції фітнесу. Тепер даний індивід буде створювати дуже близьких до правильного розв’язання нащадків, що посідатимуть верхні рядки таблиці та продовжуватимуть рід. А так як найкращою мутацією для цих нащадків є повернення до попереднього стану, то дуже швидко популяція втратить всю різноманітність, наповнившись копіями одного і того самого тупикового фенотипу.

Дану ситуацію показано на рисунку 2.3. Усі можливі мутації з точки локального максимуму створюють деяку область, вийти з якої або неможливо зовсім при даному алгоритмі, або це займе занадто багато ітерацій.

У результаті тестування алгоритму було виявлено, що він знаходить розв’язки даного рівняння у 603 випадках із 1000, а досягає результату у середньому за 102 покоління. Оскільки у майже 40% запусків програма не приходить до результату за визначену максимальну кількість поколінь (100 тисяч), важко назвати цей алгоритм дуже успішним.

Алгоритм програми для тестування представлено у Додатку Б.

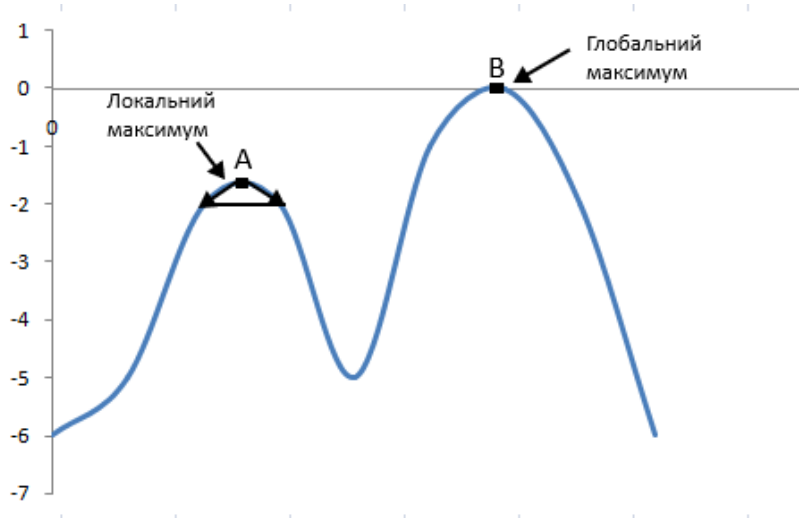

Рисунок 2.3 – Мутація не може вийти з інтервалу навколо точки А

2.2.3 Методи рішення проблем збіжності генетичного алгоритму

Проблему збіжності алгоритму можна вирішити використанням іншої фітнес-функції, збільшенням ймовірності мутацій, або використанням методів відбору, які підтримують різноманітність рішень у популяції[13].

Різноманітність важлива для генетичних алгоритмів, тому що перехрест генів у гомогенній популяції не несе нових рішень. Якщо з кожного стану популяції є мала кількість можливих переходів, то легше

зайти у локальний максимум функції фітнесу, а якщо таких переходів багато, то цього легше уникнути.

У базовій реалізації алгоритму, основу якого викладено у підрозділі 2.2.1, у майже 40% запусків програма не приходить до результату через застрягання поблизу локального максимуму. В ході написання коду алгоритму було апробовано декілька різних методів рішення проблеми збіжності, але краще за все себе показало додання шуму, випадково розподіленого за нормальним законом, до значення функції фітнесу.

Отримане випадкове значення множиться на коефіцієнт шуму *NOISE_SCALE* і додається до функції фітнесу. Далі саме за значенням цієї зашумленої функції фітнесу і сортуються індивіди, а перевірка розв'язку рівняння відбувається за чистою функцією фітнесу, визначеною раніше. Завдяки додаванню шуму збільшується мінливість генів і з локального максимуму стає легше вибратись.

Очевидно, що додавання до значення функції фітнесу випадкового шуму окрім переваг у кращій збіжності несе і недоліки. Коли шум додається до функції фітнесу індивіда з високим її значенням, то через штучно створений елемент хаосу йому стає важче потрапити до її найкращого значення, що сповільнює роботу алгоритму і збільшує кількість поколінь для вирішення задачі.

Отже, треба знайти баланс між покращенням відсотку збіжності та її швидкістю, підібравши оптимальне значення коефіцієнту шуму *NOISE_SCALE*. Протестуємо значення від 0 (базовий алгоритм) до 20 по 1000 разів кожен, щоб визначити оптимальний з них при збереженні максимальної швидкодії.

Результати тестування різних коефіцієнтів шуму на відсоток не вирішених задач представлено на рисунку 2.4.

Так, якщо для базової версії алгоритму не будо вирішено поставлену задачу у 40% випадків, після реалізації шуму цей відсоток зменшується до 2% при коефіцієнті шуму 6 та до 0,7% при значення *NOISE_SCALE* = 11.

Результати тестування різних коефіцієнтів шуму на середню кількість поколінь, що потребуються для рішення задачі, представлено на рисунку 2.5.

З рисунку 2.5 слідує, що швидкодія програми зменшується зі збільшенням коефіцієнту шуму і цю залежність можна досить точно представити у вигляді лінійної функції.

Рисунок 2.4 – Залежність відсотку невирішених задач від значення коефіцієнту шуму

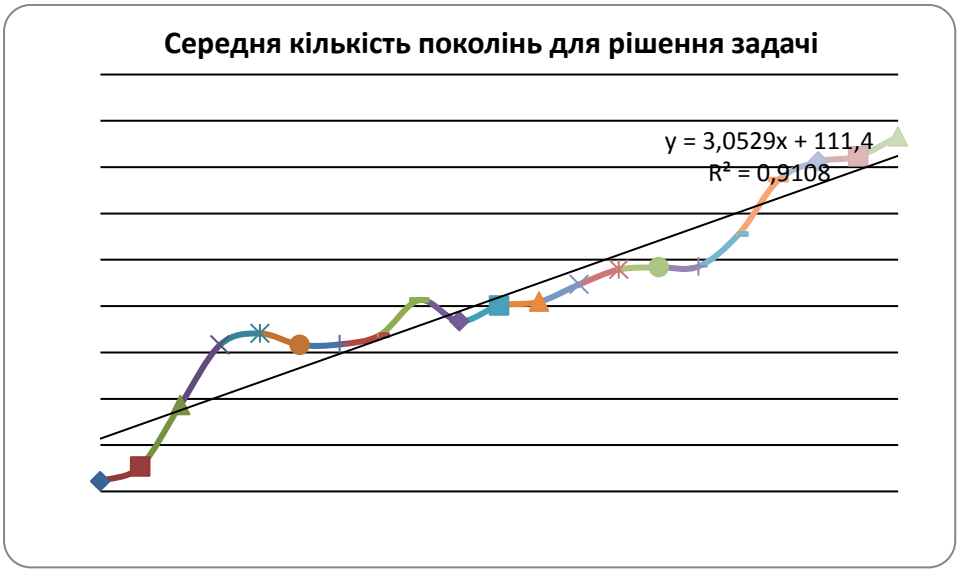

Рисунок 2.5 – Залежність середньої кількості поколінь для розв’язання задачі від значення коефіцієнту шуму

Для більшої наочності залежність обох параметрів від коефіцієнту шуму представлено графічно на рисунку 2.6 та у табличному вигляді у таблиці 2.6.

Отже, протестувавши значення від 0 до 20, було експериментально визначено, що оптимальним значенням даного коефіцієнту є 10-11. Подальше його збільшення вже не призводить до значного покращення збіжності алгоритму, проте збільшує кількість необхідних поколінь, яка росте лінійно від шуму, що погіршує швидкодію програми.

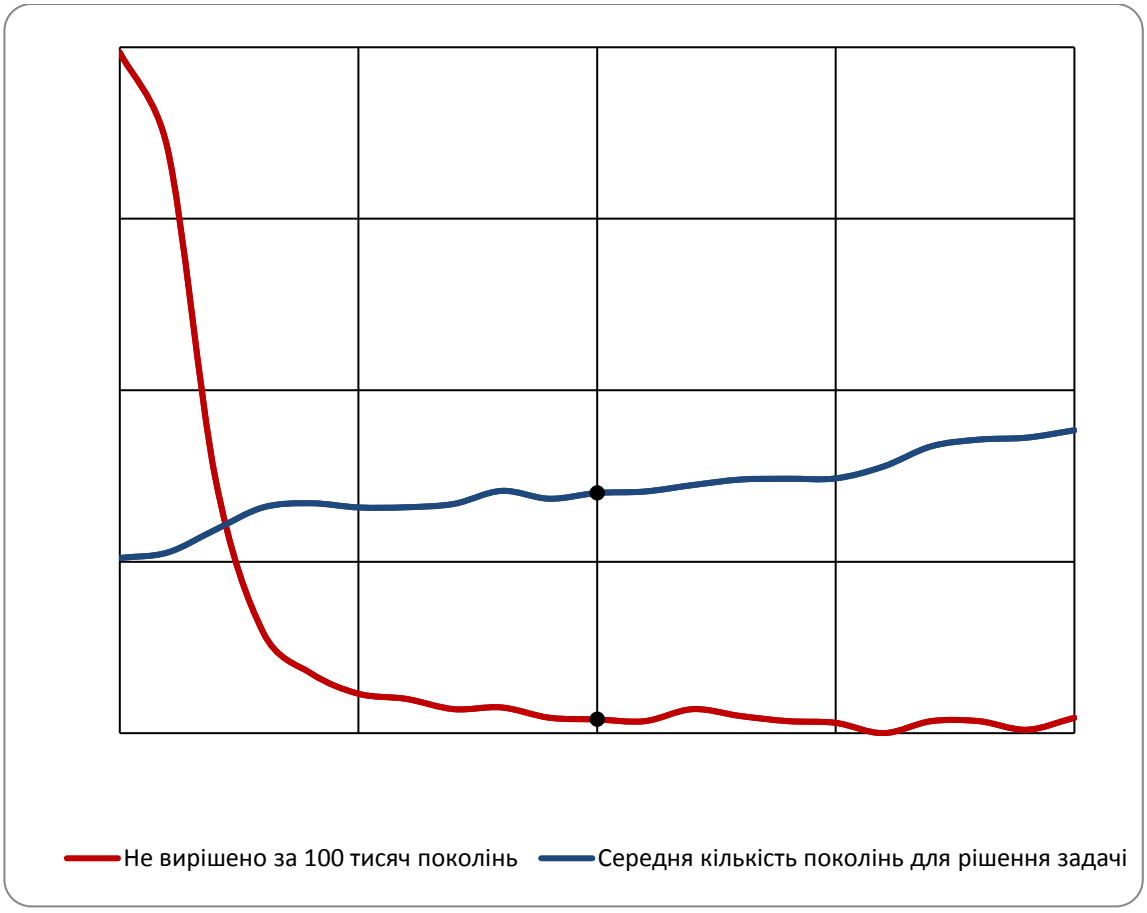

Рисунок 2.6 – Залежність відсотку невирішених задач та швидкодії програми від значення коефіцієнту шуму

Таблиця 2.6 – Залежність відсотку невирішених задач та швидкодії програми від значення коефіцієнту шуму

Значення коефіцієнту шуму	Не вирішено за 100 тисяч поколінь	Середня кількість поколінь для розв’язання задачі
0 (без шуму)	397	102
1	341	105
2	149	119
3	59	132
4	35	134
5	23	132
6	20	132
7	14	134
8	15	141
9	9	137
10	8	140
11	7	141
12	14	145
13	10	148
14	7	148
15	6	149
16	0	155
17	7	167
18	7	171
19	2	172
20	9	177

2.2.4 Гра «Жук»

При розв’язанні генетичним алгоритмом діофантових рівнянь досягнення локального максимуму є вкрай небажаною подією, яка гальмує процес розв’язання і заважає прийти до справжніх коренів рівняння. Справа в тому, що у функції фітнесу даної задачі відоме значення максимуму, до якого треба прийти, оскільки розв’язком рівняння є знаходження таких коренів, щоб їх вільний член точно співпадав з вільним членом заданого рівняння. А що якщо вирішувати за допомогою генетичного алгоритму задачу, максимум функції фітнесу якої не обмежений зверху?

Як приклад розберемо, на перший погляд, просту гру, пограти в яку можна за електронною адресою https://buglab.ru/index.asp. У цій грі треба побудувати незамкнутий (можливий для проходу) лабіринт для «жука», який рухається цим лабіринтом за певним алгоритмом. Мета гри – затримати жука у побудованому Вами лабіринті максимальний час. Вихід знаходиться у правому нижньому кутку. Якість лабіринту визначається кількістю ходів, які потрібні жуку для того, щоб вибратися з лабіринту.

Детальніше про алгоритм дій «жука» можна дізнатися за посиланням https://acmp.ru/index.asp?main=task&id_task=399 .

Оскільки невідомо, яке значення кількісті ходів максимальне (а повний перебір усіх можливих варіантів на даному полі вимагає розгляду 2^549 варіантів, що приблизно дорівнює $7 \cdot 10^{160}$ і значно більше кількості атомів у Всесвіті), то будь-яке значення, що більше за максимальне попередньо досягнуте нами, є успіхом.

Спробуємо знайти максимально довгий маршрут жучка за допомогою генетичного алгоритму. При розв’язанні даної задачі мутацією буде зміна значень декількох клітинок на протилежні. Тобто, блоки будуть ставитися на деякі пусті місці, а на деякі пусті місця будуть ставитися нові блоки.

У коді проєкту, представленого у Додатку В, у половини кращих особин змінюються значення чотирьох клітинок, після чого знову сортуються за функцією фітнесу, що відповідає кількості ходів, які треба зробити жучку для подолання лабіринту.

Насправді у даній задачі нам все одно, який самий алгоритм руху у жучка, оскільки якщо він не випадковий, то рано чи пізно генетичним алгоритмом буде знайдено оптимальний розв’язок. З кожного стану поля є приблизно 500 різних переходів, тому потрапити у локальний максимум стає складніше. І навіть потрапляння у локальний максимум у задачах такого типу не є занадто поганою подією, оскільки значення цього максимуму може бути високим і в результаті може бути отримано деякий цікавий патерн, який потім можна додатково дослідити.

На даний момент кращим результатом, отриманим у цьому проєкті за допомогою цієї програми, є близько 2 мільйонів очок. Його можна покращити, вдосконаливши сам алгоритм або просто виконавши більше прогонів даного. Однак, цей результат уже посідає 23-є місце у турнірній таблиці сайту з 1197 представлених розв’язків. На рисунку 2.7 представлено отриманий лабіринт, а на рисунку 2.8 – теплову карту руху жучка (скільки разів він побував у кожній клітинці).

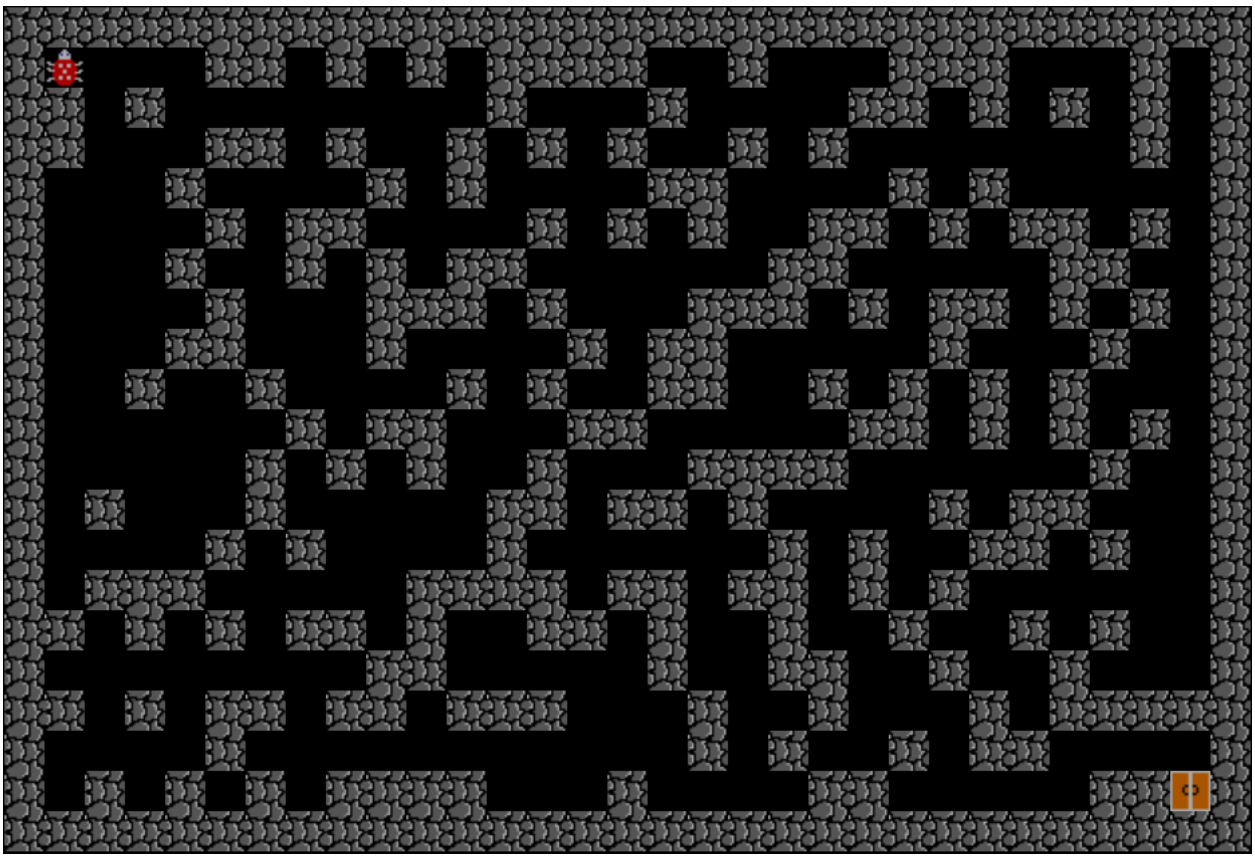

Рисунок 2.7 – Лабіринт, по якому жучок зробить 1984034 кроків

****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****
****	4459	4585	4465	4491	****	****	2509	****	2509	****	2513	****	****	****	****	6447	6517	****	6288	9513	6286	****	****	****	16312	13594	16310	****	15942	****
****	****	4474	****	4535	4503	4436	4518	2509	4519	2513	4514	****	3581	7340	3581	****	6535	6413	9514	6288	****	****	8280	****	13594	****	13595	****	23922	****
****	****	4476	4541	4479	****	****	2511	****	2511	4518	****	3608	****	3593	****	6510	6581	****	6307	****	9331	9234	16588	13591	16315	13594	16310	****	15943	****
****	4389	5488	4411	****	1696	2552	4515	2512	****	2511	****	7143	3608	7287	3594	****	****	6307	9408	9262	9350	****	8280	****	13593	16313	23924	15942	23923	****
****	5488	4388	5493	4386	****	1697	****	****	2503	5051	3723	3725	****	3614	****	3624	****	9407	6307	****	****	16	****	16	****	****	15942	****	15942	****
****	4389	5489	4387	****	1696	2553	****	1182	****	2504	****	****	3613	7255	3624	7212	6492	6499	****	****	16	41	17	31	18	****	****	15941	23923	****
****	5490	4388	5492	4384	****	1779	1184	1780	****	****	****	808	****	3623	7210	3624	****	****	****	43	****	33	****	****	17	****	0	****	23501	****
****	4390	5492	4388	****	****	1184	1780	1183	****	1321	1285	1679	808	****	3639	****	****	47	55	53	44	35	****	7	19	7	****	23607	23761	****
****	5491	4389	****	5301	5352	****	1183	1781	1304	1334	****	824	****	3638	7141	****	****	54	48	****	34	****	2	****	8	****	23864	23695	23692	****
****	4390	5491	4392	5490	5310	5225	****	1181	****	****	827	1654	825	****	****	48	53	48	61	48	****	****	3	****	8	****	23687	****	23522	****
****	5492	4390	5492	4392	5491	****	741	****	741	****	1603	827	****	55	47	54	****	****	****	****	2	3	4	6	10	6	****	23519	35276	****
****	4388	****	4391	5493	4392	****	873	743	874	744	870	****	****	45	****	****	74	****	1	2	2	2	****	3	****	****	23518	35279	23519	****
****	5495	4389	5494	4391	****	391	****	870	744	871	744	****	36	82	76	76	162	75	****	1	****	4	1	****	****	22355	****	23519	35277	****
****	4386	****	****	****	391	811	771	786	873	****	****	****	****	37	****	****	105	****	****	1	****	2	****	31893	31885	44749	23517	35277	23519	****
****	****	389	****	397	****	414	****	****	742	****	276	230	****	****	262	****	157	106	****	1	1	****	29247	43883	****	22355	****	23518	35277	****
****	389	820	397	795	414	781	435	224	****	****	231	277	232	277	276	****	106	157	****	****	1	0	****	29249	43883	****	23516	35277	23518	****
****	****	708	****	768	****	****	225	****	****	221	****	****	****	233	277	232	****	233	150	****	2	1	1	****	29249	****	****	****	****	****
****	765	745	778	786	****	221	468	282	286	293	224	292	226	291	231	277	****	151	****	1	2	****	1	****	****	1	1	1	1	****
****	741	****	736	****	0	****	221	****	****	****	****	225	291	226	****	232	226	233	150	****	****	****	0	1	1	1	1	****	****	****
****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****	****

Рисунок 2.8 – Теплова карта руху жучка

ВИСНОВКИ

Робота присвячена дослідженню і вдосконаленню методів розв’язаннядіофантових рівнянь за допомогою генетичних алгоритмів. У роботі розкривається історичний контекст дослідженнядіофантових рівнянь та викликів, пов'язаних із їх розв'язанням. Крім того, розглянуті основні теоретичні аспекти генетичних алгоритмів і їх застосування для розв’язаннядіофантових рівнянь.

У ході роботи були розглянуті різні методи розв'язання діофантових рівнянь, розробленопрограмнуреалізаціюрозв’язаннядіофантових рівнянь за допомогою генетичного алгоритму. Виявлено проблеми збіжності генетичного алгоритму і досліджено ефективні методи їх вирішення.

У ході роботи було вдосконалено базовий генетичний алгоритм додаванням випадкового шуму до цільової функції, протестовано набір різних значень початкових параметрів програми і експериментально визначено оптимальні за швидкодією та збіжністю.

Отримані результати підтверджують, що генетичні алгоритми є потужним та перспективним інструментом для розв'язання складних діофантових рівнянь.

У цій роботі поєднання теоретичних аспектів та практичних розробок дозволяє зрозуміти можливості та переваги генетичних алгоритмів у розв'язанні діофантових рівнянь. Отримані результати відкривають нові перспективи для подальших досліджень у цій області та можливості застосування отриманих знань у вирішенні прикладних завдань в різних галузях науки та техніки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Клесов О. Елементарна теорія чисел та елементи криптографії. Київ: ТвіМС. 2016. 412 с.

2. Гнезділова Т. Діофантові рівняння: різні способи розв'язування лінійних і не лінійних рівнянь. *Математика. Шкільний світ*. 2009. № 38. С.13-21.

3. Гнезділова Т. Діофантові рівняння: різні способи розв'язування лінійних і не лінійних рівнянь. *Математика. Шкільний світ*. 2009. № 39. С.14-20.

4. Гнезділова Т. Діофантові рівняння: різні способи розв'язування лінійних і не лінійних рівнянь. *Математика. Шкільний світ*. 2009. № 41. С.10-21.

5. Mordell L. J. Diophantine equations. NewYork : Academic Press. 1969. 312 p.

6. Грохольська А. В. Невизначені рівняння. *Математика в школі*. 2003. № 5. С. 36-43.

7. Лейфура В. М. Діофантові рівняння. У світі математики. 1985. № 16. С. 57-69.

8. Krumbiegel B., Amthor A. Das Problema Bovinum des Archimedes. *Historisch-literarische Abteilung der Zeitschrift fürMathematik und Physik*. 1880. № 25. P. 121-136.

9. Dörrie, Heinrich. Archimedes' Problema Bovinum. *100 Great Problems of Elementary Mathematics.* New York : Dover Publications. 1965. P. 3-7.

10. Щербаков В. І. Простір Гільберта. Київ : Веселка. 1989.205 с.

11. Sprindzuk V. G. Classical Diophantine Equations. Berlin : Springer. 1993. 228 p.

12. Завало С. Т, Левіщенко С. С., Пилаєв В. В., Рокицький І. О. Алгебра і теорія чисел : Практикум. Частина 2. Київ : Вища школа. 1986. 264 с.

13. Poli R., Langdon W.B., McPhee N. F. A Field Guide to Genetic Programming. Raleigh : Lulu Enterprises. 2008. 233p.

14. J. H. Holland. Adaptation in natural and artificial systems. Ann Arbor : University of Michigan Press. 1992. 232 p.

15. Ішин С. Самопрограмування : Еволюціонуючі алгоритми, самопрограмування та самомодифікація програм. Київ : Техніка. 1997. 28 с.

16. Srinivas N., Deb K. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*. 1994. № 12. P. 221-248.

ДОДАТОК А

Програмна реалізація розв’язання діофантових рівнянь генетичним алгоритмом

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <ctime>
#include <cmath>
using namespace std;

#define VARIABLE_COUNT 5
#define POPULATION_SIZE 10
#define MAX_INITIAL_VALUE 5
#define NOISE_SCALE 10.0
#define MAX_STEPS 100000
#define LOG_STEP 1000
constdouble PI = 3.14159265358979323846;
structIndividual {
vector<int>variables;
inttrue_fitness;
intnoised_fitness;
};
boolcomp_true_fitness(Individual&a, Individual&b) {
returna.true_fitness>b.true_fitness;
}

boolcomp_noised_fitness(Individual&a, Individual&b) {
```

```
returna.noised_fitness>b.noised_fitness;
}

intget_fitness(vector<int>&x) {
intsum = 3 * x[0] + 5 * x[1] + 11 * x[2] + 17 * x[3] + 23 * x[4];
inttarget = 1171;
intfitness = -abs(sum - target);
returnfitness;
}

floatstandard_normal_distribution() {
float u = (float) rand() / RAND_MAX;
float v = (float) rand() / RAND_MAX;
returnsqrt(-2 * log(u)) * cos(2 * PI * v);
}

intnoisify_fitness(intfitness) {
returnfitness + NOISE_SCALE * standard_normal_distribution();
}
voidrandom_init(vector<Individual>&population) {
for(int i = 0; i < POPULATION_SIZE; i++) {
population[i].variables.resize(VARIABLE_COUNT);

for(int j = 0; j < VARIABLE_COUNT; j++) {
population[i].variables[j] = rand() % MAX_INITIAL_VALUE + 1;
      }

population[i].true_fitness = get_fitness(population[i].variables);
population[i].noised_fitness = noisify_fitness(population[i].true_fitness);
   }
```

```
}

voidmutate_individual(Individual&individual) {
   // Додати 1 або відняти 1 від випадкової змінної
intindex = rand() % VARIABLE_COUNT;
intsign = rand() % 2;

if(sign) {
individual.variables[index]++;
   } else {
individual.variables[index]--;
   }

   // Забезпечити, щоб змінна не вийшла за межі допустимих значень (натуральні числа). Якщо змінна нуль - перетворити на 2, наче ми додали 1, а не відняли
if(individual.variables[index] == 0) {
individual.variables[index] = 2;
   }

   // Перерахувати фітнес-функцію
individual.true_fitness = get_fitness(individual.variables);
individual.noised_fitness = noisify_fitness(individual.true_fitness);
}

intmain(intargc, char **argv) {
ios::sync_with_stdio(false);
if(argc< 2) {
srand(time(NULL));
   } else {
```

```
srand(atoi(argv[1]));
   }

vector<Individual>population(POPULATION_SIZE);
random_init(population);

for(intstep_id = 0; step_id< MAX_STEPS; step_id++) {
      // Перевірити, чи знайшли ми розв'язок

for(int i = 0; i < POPULATION_SIZE; i++) {
if (population[i].true_fitness == 0) {
cout<< "Foundsolutionatstep " <<step_id<< endl;
for (int j = 0; j < VARIABLE_COUNT; j++) {
cout<<population[i].variables[j] << " ";
            }
cout<< endl;
return 0;
         }
      }

sort(population.begin(), population.end(), comp_noised_fitness);

      // Скопіювати першу половину популяції в другу половину та
виконати мутацію
for(int i = POPULATION_SIZE / 2; i < POPULATION_SIZE; i++) {
population[i] = population[i - POPULATION_SIZE / 2];
mutate_individual(population[i]);
      }

if(step_id&&step_id % LOG_STEP == 0) {
```

```
cout<< "Step " <<step_id<< ": ";
for(int i = 0; i < POPULATION_SIZE; i++) {
cout<<population[i].true_fitness<< " ";
        }
cout<<endl;
      }
   }
cout<< "Solutionnotfoundin " << MAX_STEPS << " steps" <<endl;
return -1;
```

ДОДАТОК Б

Програма для тестування алгоритму

```
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <string>
usingnamespacestd;
intmain(intargc, char** argv) {
srand(time(NULL));
intruns = atoi(argv[2]);
intunsuccessful_runs = 0;
for(int i = 0; i <runs; i++) {
intseed = rand() % 1000000;
intresponse = system((argv[1] + string(" ") +to_string(seed)).c_str());
cout<< "Run " << i << " withseed " <<seed<< " returned "
      <<response<<endl;
if(response != 0) {unsuccessful_runs++;}
}
cout<< "Unsuccessfulruns: " <<unsuccessful_runs<<endl;
cout<< "Totalruns: " <<runs<<endl;
return 0;
```

ДОДАТОК В

Програмна реалізація пошуку кращого рішення у грі «Жук» генетичним алгоритмом

```
#include<iostream>
#include<vector>
#include<algorithm>
#include<queue>
#include<string>
#include<time.h>
#include<fstream>

#define INF 2000000000

usingnamespacestd;

boolcheck(vector<vector<int>> v)
{
pair<int,int>start = {1,1};
pair<int,int>finish = {v.size()-2,v[0].size()-2};
queue<pair<int,int>> q;
q.push(start);
while(!q.empty())
   {
pair<int,int>tmp = q.front();
q.pop();
if(v[tmp.first][tmp.second]) continue;
```

```
        v[tmp.first][tmp.second] = 1;

if(!v[tmp.first + 1][tmp.second])
q.push({tmp.first + 1, tmp.second});

if(!v[tmp.first][tmp.second + 1])
q.push({tmp.first, tmp.second + 1});

if(!v[tmp.first - 1][tmp.second])
q.push({tmp.first - 1, tmp.second});

if(!v[tmp.first][tmp.second - 1])
q.push({tmp.first, tmp.second - 1});

    }
if(v[finish.first][finish.second] == INF || v[1][1] == INF) returnfalse;
return v[finish.first][finish.second];
}

intrun(vector<vector<int>> v)
{
boolflag = check(v);
if(!flag)return -1;

pair<int,int>pos = {1,1};
pair<int,int>finish = {v.size()-2,v[0].size()-2};

intcnt = 0;
pair<int,int>dir = {1,0};
autodirs = vector<pair<int,int>>{{1,0},{0,1},{-1,0},{0,-1}};
```

```
while(pos != finish)
   {
cnt++;
      v[pos.first][pos.second] ++;
intm_steps = INF;
pair<int,int>offset;
int steps;

steps = v[pos.first + 1][pos.second];
if(steps<m_steps || steps == m_steps&&dir == make_pair(1,0))
      {
offset = {1,0};
m_steps = steps;
      }

steps = v[pos.first][pos.second + 1];
if(steps<m_steps || steps == m_steps&&dir == make_pair(0,1))
      {
offset = {0,1};
m_steps = steps;
      }

steps = v[pos.first - 1][pos.second];
if(steps<m_steps || steps == m_steps&&dir == make_pair(-1,0))
      {
offset = {-1,0};
m_steps = steps;
      }
```

```
steps = v[pos.first][pos.second - 1];
if(steps<m_steps || steps == m_steps&&dir == make_pair(0,-1))
        {
offset = {0,-1};
m_steps = steps;
        }

pos.first += offset.first;
pos.second += offset.second;
dir = offset;
   }
returncnt;
}

vector<vector<int>>makefield (int n, int m)
{
vector<vector<int>> v(n,vector<int> (m,0));
for(int i=0;i<n;i++)
   {
for(int j=0;j<m;j++)
        {
if(i==0 || j==0 || i == n-1 || j == m-1) v[i][j] = INF;
        }
   }
return v;
}

voidoutvector(vector<vector<int>>&v)
{
int n = v.size();
```

```
int m = v[0].size();
for(int i=0;i<n;i++)
   {
for(int j=0;j<m;j++)
      {
if(v[i][j] == INF) cout<<'#';
elsecout<<'.';
      }
cout<<endl;
   }
}

vector<vector<int>>mutation(vector<vector<int>> v)
{
int n = v.size();
int m = v[0].size();
vector<vector<int>>ans = v;

for(intcounter=0;counter<5;counter++)
   {
int i = rand() % (n - 2) + 1;
int j = rand() % (m - 2) + 1;
if(ans[i][j] == INF)ans[i][j] = 0;
elseans[i][j] = INF;
   }
returnans;
}

void
show_population(vector<pair<int,vector<vector<int>>>>&population)
```

```
{
for(int i=0;i<population.size();i++)
   {
cout<<population[i].first<<endl;
outvector(population[i].second);
cout<<endl<<endl;
   }
}

boolcomp(pair<int,vector<vector<int>>> a,
pair<int,vector<vector<int>>>b)
{
returna.first>b.first;
}

voidshow (pair<int,vector<vector<int>>> p)
{
stringname = to_string(p.first) + ".txt";
ofstreamfout (name.c_str());
int n = p.second.size();
int m = p.second[0].size();
for(int i=0;i<n;i++)
   {
for(int j=0;j<m;j++)
      {
if(p.second[i][j] == INF) fout<<'#';
elsefout<<'.';
if(p.second[i][j] != INF &&p.second[i][j] != 0) fout<<'@';
      }
fout<<endl;
```

```
    }
fout.close();
}

intmain()
{
srand(time(NULL));
intpopulation_size = 10, q_i = 21, q_j = 31;
vector<pair<int,vector<vector<int>>>>population(population_size);
for(int i=0;i<population_size;i++) population[i].second = makefield(q_i,
q_j);
for(int64_t infinity = 0; true; infinity++)
    {
cout<<infinity<<"\r";
for(int i = (population_size / 2); i <population_size; i++)
population[i].second = mutation(population[i - population_size /
2].second);
for(int i = 0; i <population_size; i++) population[i].first =
run(population[i].second);
sort(population.begin(), population.end(), comp);
if(infinity % 1000 == 0 &&infinity != 0)
        {
cout<<endl;
show(population[0]);
cout<<population[0].first<<endl;
outvector(population[0].second);
cout<<endl<<endl;
        }
    }
}
```