КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота

на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

НЕОДНОРІДНІ ГЕНЕТИЧНІ АЛГОРИТМИ

Виконала: студентка 4-го курсу
Ольга ДУДАР

(підпис)

Науковий керівник:
професор кафедри теоретичної кібернетики
доктор фіз.-мат. наук, професор
Анатолій ПАШКО

(підпис)

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

(підпис)

Роботу розглянуто й допущено до захисту
на засіданні кафедри теоретичної кібернетики
«____» _____ 2022 р.,
протокол № ___
Завідувач кафедри
доктор фіз.-мат. наук, професор
Юрій КРАК

_____
(підпис)

КИЇВ-2022

РЕФЕРАТ

Обсяг роботи складає 46 сторінок, 20 ілюстрацій, 1 таблиця, 20 джерел посилань.

ГЕНЕТИЧНІ АЛГОРИТМИ, ЕВОЛЮЦІЙНІ АЛГОРИТМИ, МУТАЦІЯ, НЕОДНОРІДНІ АЛГОРИТМИ, КЕРУВАННЯ ПОПУЛЯЦІЄЮ, ЕВОЛЮЦІЙНА СТРАТЕГІЯ, АЛГОРИТМИ ПОШУКОВОЇ ОПТИМІЗАЦІЇ.

Об’єктом дослідження є процес побудови генетичного алгоритму для знаходження локального екстремуму двовимірної функції.

Предметом роботи є програмний засіб для обчислення екстремуму функції двох змінних.

Метою роботи є створення навчального програмного засобу розв’язування задач на розрахунок екстремумів функцій на заданих проміжках за допомогою генетичного алгоритму та його модифікацій для покращення результатів його виконання.

Інструменти розроблення: безкоштовне, вільно поширюване інтегроване середовище розробки Microsoft Visual Studio Community, мова програмування C++.

Результати роботи: виконано огляд еволюційних, а саме генетичних алгоритмів та їх різновидів для задачі пошуку локального екстремуму, проаналізовано способи вдосконалення результату розрахунків та проведено порівняльний аналіз отриманих результатів графічним та табличним способами.

Результуючий продукт може використовуватися як навчальний засіб для опановування генетичних алгоритмів та методів роботи з ними. Програмний продукт стане в нагоді для практичної роботи як з нейронними мережами при пошуку кращого варіанту, так і з залежними даними для знаходження точок локального екстремуму на різних проміжках та значень незалежних змінних в цій точці.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об'єкта дослідження. Об'єктом дослідження виступають генетичні алгоритми (такі, що використовують механізми, подібні до біологічної еволюції) та їх різновиди. Станом на 2022 рік існують реалізовані клітинні, однорідні, неоднорідні зі зміною величини мутації, із дійсними та бінарними кросоверами, недомінованні сортувальні (NSGA-II) генетичні алгоритми та багато інших їх видів.

Генетичні алгоритми беруть свій початок в роботах Джона Голанда (John Holland) у 1960-тих роках, хоча набули широкого розголосу аж в 90-тих роках: у 1992 році Джон Коза (John Koza) використав генетичний алгоритм для того, щоб впровадити програми для виконання певних завдань і назвав цей метод «генетичним програмуванням».

Ці алгоритми називають «натхненними природою», тому що вони імітують модель пристосування живих істот до навколишнього середовища. Фокусуючись на еволюції генетичного матеріалу всередині певної групи осіб, вони в кожному поколінні моделюють відтворення та передавання генетичного матеріалу (генів, сотні яких містяться у хромосомах) від батьків до нащадків – так, як це відбувається у природі.

Дані алгоритми досі широко використовуються. Вони мають право на існування, тому що вони добре справляються з пошуком у потенційно величезному просторі даних, комбінуючи дані і різними способами урізноманітнюючи пошукові області, таким чином отримуючи перевагу над іншими алгоритмами пошуку.

Актуальність роботи та підстави для її виконання. У теперішньому світі науки ці алгоритми активно застосовуються у розробці штучного інтелекту, як і інші пошукові алгоритми, для визначення поля потенційних вирішень проблем, щоб виокремити таке з них, що буде вирішувати завдання найкращим чином.

Серед таких застосувань – кластеризація даних, обробка зображень і нейронні мережі. Сучасні алгоритми представляють різні способи оптимізації даного алгоритму, проте все ще необхідні модифікації та вдосконалення, які могли б забезпечити покращення результату без великих затрат часу.

Мета й завдання роботи.

Мета роботи – виокремити напрямки та способи вдосконалення рішень, націлених на пошук найбільш вдалих результатів розв’язання проблем.

Завдання роботи:

а. систематизувати існуючий досвід втілення модифікацій вище згадуваного алгоритму;
б. виокремити його недоліки;
в. окреслити шляхи підвищення ефективності його роботи;
г. імплементувати різні за принципом роботи версії алгоритму, що розглядається;
д. порівняти результати із фактичним значенням та зі значеннями, отриманими в результаті роботи різних модифікацій алгоритму;
е. продемонструвати наслідки виконаних досліджень.

Об'єкт, методи й засоби дослідження і розробки. Об’єктом дослідження виступає функція двох змінних і пошук її локальних екстремумів на певних інтервалах. Методом дослідження слугує клас пошукових еволюційних алгоритмів.

Засобами розробки є мова програмування C++ у середовищі розробки Visual Studio 2022 та інструмент для розробки та подання проектів в інтерактивному вигляді Jupyter Notebook.

Можливі сфери застосування.

Розроблена програма може застосовуватися як для практичної роботи із штучним інтелектом, нейронними мережами для пошуку найкращого рішення, так і з залежними даними для знаходження точок локального екстремуму на

різних проміжках та значень незалежних змінних в цій точці. Також застосунок стане в нагоді як навчальний засіб для людей, які опановують генетичні алгоритми і методи роботи з ними і досліджують їх варіації.

Взаємозв’язок з іншими роботами.

Дану роботу було виконано на основі відомих еволюційних пошукових алгоритмів та теоретичних відомостей про них.

Робота може виконуватися сумісно з іншими роботами в рамках комплексних досліджень або розробок оптимізації функцій, навчанню штучного інтелекту, обробки зображень і так далі. Принципи та методи роботи написаної програми тісно переплітаються з навчальним предметом парадигм програмування.

РОЗДІЛ 1. Загальне представлення алгоритмів

1.1 Загальна інформація

Генетичні алгоритми – пошукові алгоритми, адаптивні та засновані на принципах теорії природної еволюції Чарльза Дарвіна (Charles Darwin), спадкоємства та відбору. Вони являються метаевристичними, що означає, що вони є вищими рівнями евристик чи процедур, які можуть згенерувати чи знайти таку евристику, що може представити дуже вдале вирішення оптимізаційної проблеми.

Особливо необхідними вони є в ситуаціях, коли дані, що розглядаються – явно неповні, неідеальні, як-от коли невідомо й приблизно, якими мають бути результуючі дані, чи взагалі пошук рішення для яких перевищує обчислювальну здатність електронно-обчислювальних машин. Метафорично вони використовуються для задач, оцінка якості рішення яких можлива тільки тоді, коли це рішення буде знайдене.

Відмінності між генетичними алгоритмами та іншими традиційними способами оптимізації і пошуку полягають у деяких ключових і навіть базових аспектах. По-перше, вони проводять обробку закодованих форм параметрів поставленої задачі, не використовують напряму їх значення.

По-друге, вони шукають рішення не з огляду тільки на одну точку, а з огляду на цілу популяцію точок – оскільки чим більше значень, які мали різні параметри, тим більше шансів знайти оптимальне рішення.

По-третє, він оброблює тільки цільову функцію – без «посередників» чи помічників у вигляді похідних, інтегралів і так далі, оскільки пряме використання забезпечить точніший результат та відсутність певних недоліків, зумовлених тією чи іншою зміною вигляду функції, її областей значень та визначень тощо.

І по-четверте, цей вид еволюційних алгоритмів відбирає нові значення таким чином, що для даних вираховуються імовірності, за якими вони можуть потрапити у нове покоління – і тільки від цього залежить майбутнє даних – їх спосіб вибору не детермінований.

Ці властивості надають генетичним алгоритмам перевагу над багатьма іншими алгоритмами і призводять до стійкості.

Схема загального генетичного алгоритму представлена на рисунку 1.

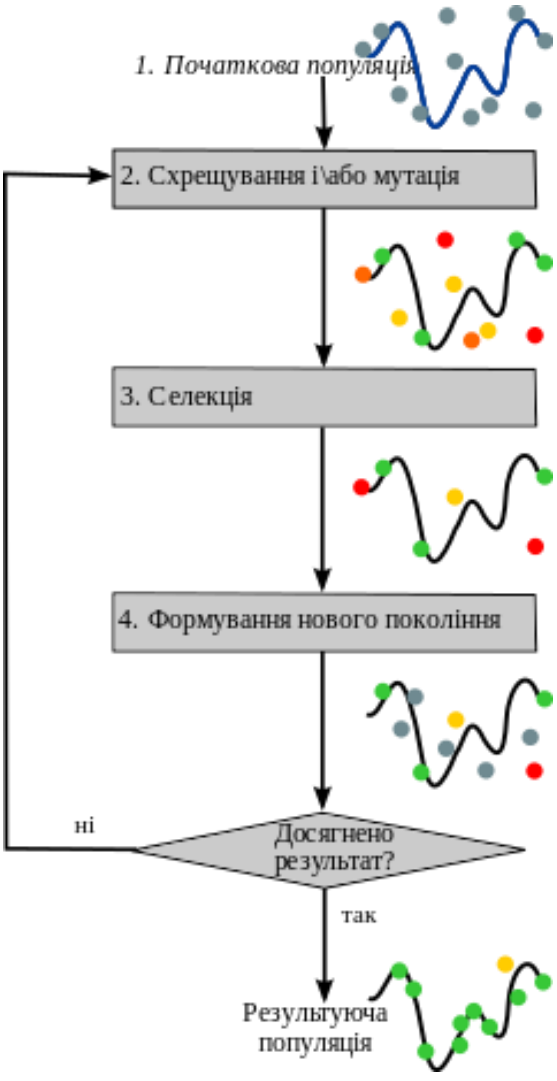

Рисунок 1.1

1.2 Основні поняття генетичних алгоритмів

Для того, щоб описувати ключові поняття генетичних алгоритмів, використовуються назви, запозичені власне з генетики. Коли мова йде про поточне покоління, зручними та інтуїтивно зрозумілими є такі назви, як популяції, хромосоми, гени, фенотипи. Крім того, застосовуються визначення з технічного лексикону, які відповідають зазначеним вище, наприклад: двійкова послідовність, структура, ланцюг.

Розберемо детальніше, що означають згадані терміни.

а) Покоління – сукупність даних, об’єктів, які належать до однієї популяції.

Кожне наступне покоління утворюється з попереднього за допомогою схрещування генів. У генетичних алгоритмах чим більше поколінь, тим краще зазвичай результат.

б) Популяція – це множина даних (особин), які відносяться до одного покоління. Це набір усіх поточних варіантів вирішення проблеми, що вирішує алгоритм.

Дані, що входять у популяцію, представляють у вигляді хромосом, в яких закодовані так звані точки в просторі пошуку (тобто search points – множини параметрів задачі).

в) Хромосома – це набір параметрів (генів), який визначає вирішення заданої проблеми. Ще його називають кодовою послідовністю чи ланцюжком.

г) Ген, або знак, детектор, властивість – це параметр, який визначає міру однієї із властивостей особини чи вказує на її присутність або відсутність. Ген є атомарною одиницею хромосоми.

На рис. 2 представлений графічний вигляд популяції у випадку бітового кодування хромосом.

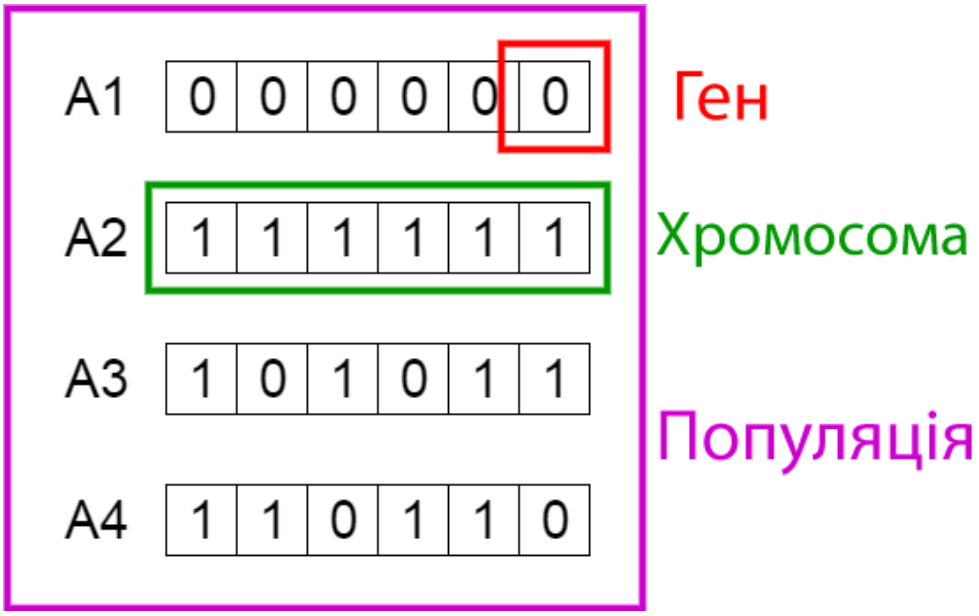

Рисунок 1.2 – Представлення осіб у бітовому вигляді

д) Генотип чи структура – набір хромосом певної особини. У більшості модифікацій алгоритму генотип містить лише одну хромосому, тому часто їх значення можна прирівняти.

е) Фенотип – це фактичні значення характеристик певної особи, які відповідають заданому генотипу. Фактично – це знайдена точка простору пошуку, тобто один із розв’язків.

ж) Алель – це значення, яке приймає конкретний ген для певної хромосоми, що означає, що він є значенням (варіантом) властивості.

з) Локус або позиція – термін, який позначає розміщення конкретного гена у ланцюжку (хромосомі). Локи – множина позицій розташування генів.

РОЗДІЛ 2. Етапи роботи алгоритму

Виділимо основні етапи генетичного алгоритму, які використовуються у більшості його варіацій та є його кістяком.

а) створюємо початкове покоління – популяцію з заданої кількості особин; зазвичай перше покоління ініціалізується випадковим чином і точність обчислень на цьому етапі дуже низька;

б) обчислюємо функцію пристосованості (так звану фітнес-функцію – англ. fitness function) для осіб популяції . Цей етап – це оцінювання якості результатів. На практиці – чим більше значення фітнес-функції, тим краще;

в) повторюємо до виконання критерію зупинки алгоритму наступні кроки:

1) вибираємо індивідів із поточної популяції (селекція) за певними критеріями: випадково, з певною імовірністю або найкращих;
2) проводимо операцію схрещення для отримання подальшої популяції;
3) використовуємо мутацію для урізноманітнення результатів та виключення можливості зациклення на одному із вже знайдених локальних екстремумів;
4) обчислюємо функції узгодженості для всіх осіб;
5) формуємо нове покоління.

Опишемо процес детальніше.

2.1 Створення початкової популяції

Із самого початку необхідно створити деяку початкову популяцію. Для цього при відсутності допоміжної апріорної інформації випадковим чином встановлюються значення генів. Навіть попри те, що це покоління абсолютно точно виявиться неконкурентоздатним, наш алгоритм відносно швидко перетворить його в наступних поколіннях на кращу версію.

Отже, на цьому кроці можна не старатись створити досить пристосованих осіб – це було б навіть зайвим, адже тоді алгоритм втрачає будь-який сенс: він має працювати з даними, у яких неможливо сказати одразу, яким має бути результат. Головне тут – щоб рандомно згенеровані дані відповідали необхідному формату, можливо – лежали в необхідних межах, і найголовніше – на них можна було без великих затрат сил порахувати функцію пристосованості.

2.2 Функція пристосованості

Дана функція є дуже важливим поняттям. Вона визначається як ступінь пристосованості поточної особини. Саме вона визначає, наскільки поточна особа відповідає поставленим вимогам і які її шанси потрапити у наступну генерацію. По-іншому її ще називають функцією оцінки – і вона відіграє найвпливовішу роль серед усіх функцій, дозволяючи на пізніших етапах вибрати з популяції найпристосованіших особин. Цей процес добре перегукується з природним процесом добору – виживають «найсильніші», найбільш підходящі на даний момент, як і в природі.

Функція пристосованості чи фітнес-функція отримала свою назву з генетики. Вона сильно впливає на хід розвитку подій у генетичних алгоритмах і повинна бути коректно визначена. У задачах оптимізації функцію

пристосованості, як правило, називають цільовою функцією та максимізують: знаходять максимальне із можливих значень.

З кожним кроком алгоритму фітнес кожного члена популяції оцінюється за допомогою функції пристосованості, на базі результату створюється наступне покоління особин, що являють собою багато точок потенційних вирішень проблем. Кожна наступна популяція в генетичному алгоритмі має назву покоління, до осіб якого застосовують термін «нащадки» чи «діти».

Розкриваючи тему фітнес-функції слід зазначити, що наведеним нижче вимогам має відповідати будь-яка функція пристосованості:

а) Чітка визначеність: користувач має достеменно розуміти, як розраховується її значення та які значення за що відповідають: менші – за кращу пристосованість, а більші – за гіршу чи навпаки; які значення вважаються допустимими, а які – поганим результатом.

б) Ефективність реалізації даної функції: якщо вирахування придатності – повільне, неефективне та/або потребує великої кількості ресурсів ЕОМ, то загальна ефективність роботи алгоритму сильно знизиться.

в) Функція придатності має кількісно визначати те, наскільки поточне рішення годиться для вирішення завдання.

г) Фітнес-функція має давати інтуїтивні і очевидні результати. Найкращі і найгірші кандидати мають володіти найкращою і найгіршою функцією пристосованості відповідно.

Однозначного правила, як будувати фітнес-функцію для кожного окремого випадку немає, хоча відповідно до типу задачі виділені окремі типи функцій пристосованості.

Як правило, в якості такої функції широко використовуються Евклідова та Мангеттенська відстань як мірила величини помилок для тих задач, де використовується контрольоване навчання. Евклідова відстань є найкоротшим шляхом між двома точками, тоді як Мангеттенська є метрикою відстані між двома

точками у N-вимірному векторному просторі – це сума усіх можливих відстаней між двома точками, кожна з яких є прямою лінією.

Для задач же оптимізації в якості останньої функції нерідко використовують базову суму набору обчислених параметрів.

2.3 Селекція

Селекція чи відбір – етап, на якому окремі особини (геноми, хромосоми) поточного покоління відбираються для того, щоб утворити нащадків для чергового покоління – наступної популяції, використовуючи оператор кросинговеру.

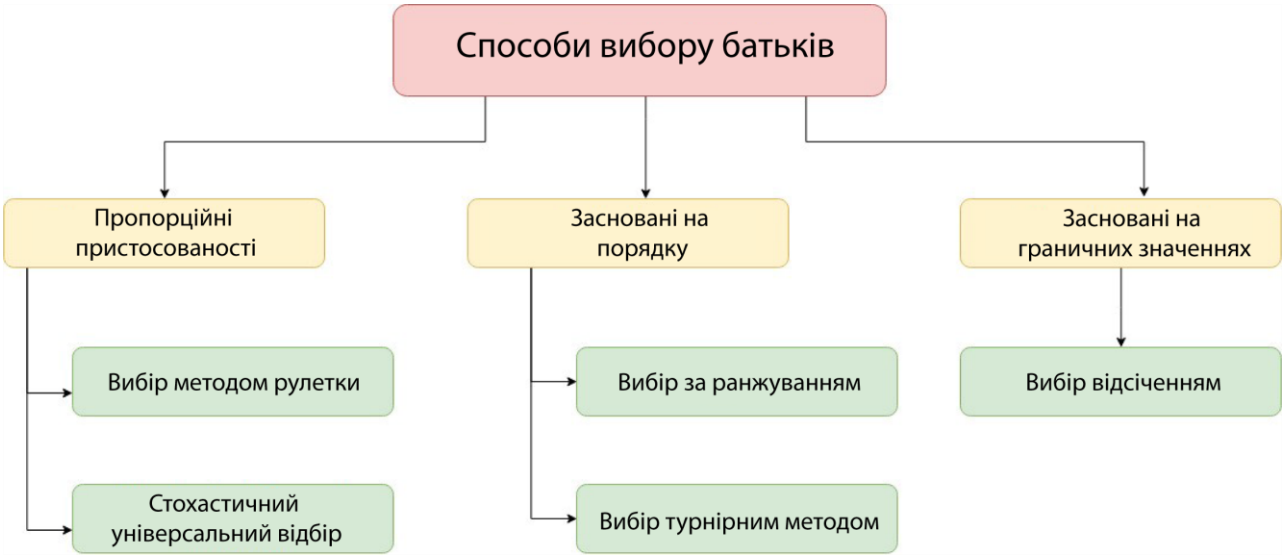

Рис. 2.1 – методи селекції

Загальну процедуру можна представити у наступному вигляді:

Для кожного геному обчислюємо фітнес-функції, яке потім нормалізуємо, що означає поділ кожного значення фітнесу на суму всіх отриманих значень пристосованості так, щоб сума усіх дорівнювала одиниці.

Після цього розраховуємо кумулятивні показники пристосованості: для кожної особи вираховуємо суму обчислених вище відносних пристосувань від першої до поточної особини популяції включно. Для перевірки правильності

розрахунків слід перевірити, чи дорівнює одиниці кумулятивний фітнес останньої особини. Якщо так, то помилки не допущено і можна продовжувати.

Обираємо псевдовипадкове число від 0 до 1 і порівнюємо його з кумулятивним фітнесом першого елемента популяції. В залежності від його значення ми або перетворюємо поточний елемент на перший, або перетворюємо його на такий перший знайдений елемент, кумулятивна пристосованість якого менше, ніж значення випадкового числа, а пристосованість наступного елемента навпаки – більша за рандомне число. Таким чином проходимо по всій популяції і отримуємо наступне покоління.

Виокремимо деякі інші відмінні оператори вибору осіб:

a) **Метод рулетки (roulette)** – також називають методом пропорційного відбору (proportional sampling). Простий метод, ідея в тому, щоб імовірність вибору особини s_i поточного покоління визначається за формулою (2.1):

$$\xi_{si} = \frac{\varphi(s_i)}{\sum_{j=1}^{|S|} \varphi(s_j)}, \; i \in [1 : |S|]. \tag{2.1}$$

Для кожного генома визначаємо пристосованість за формулою (2.1). Генеруємо випадкове число $u = U_1(0; 2\pi)$, ставимо цю особу в відповідному кутку u, в проміжну популяцію. Якщо необхідна кількість особин в популяції вже є, то завершуємо обчислення, інакше – знову генеруємо випадкове число.

Приймаємо, що кут розкриття сектора рулетки пропорційний імовірності (величині пропорційної пристосованості), якій він відповідає. На рисунку (3) можна легко бачити, що чим більша імовірність, тим більший відповідний

сектор рулетки – що означає, що відповідна особина може бути вибрана з більшою імовірністю.

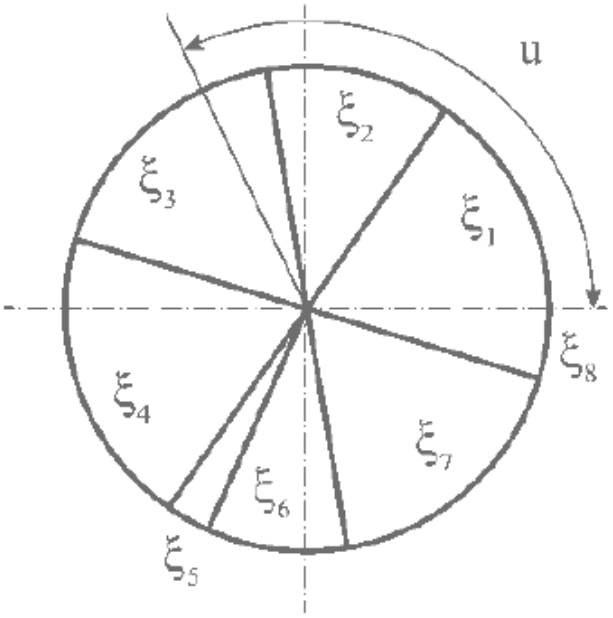

Рис. 2.2 – Рулетка: вісім осіб в популяції; відібрана особа №3

б) **Метод пропорційного відбору** – розвиток методу рулетки.

Він заснований на визначенні відношення величин пристосованості поточного члена популяції до середньої пристосованості всього покоління. Ціла частина цього відношення вказуватиме на те, скільки раз особина має бути записана в нову популяцію, а дробова – імовірність того, що вона потрапить туди ще раз.

в) **Метод відбору на основі елітизму –** цей метод заснований на попередньому виборі особин, які пристосовані найкраще.

Вичисливши фітнеси усіх елементів ті відсортувавши їх у порядку спадання відбираємо необхідну їх кількість в залежності від розміру селекції.

Важливою перевагою вважається те, що метод забезпечує збереження найбільш пристосованих осіб для наступних поколінь – часто залишають лише найкращу.

Серед модифікацій цього методу – включення до нового покоління лише десятої частини від кількості попереднього; частину, що залишилась, вибирають за допомогою одного чи декількох інших методів, в тому числі – як на початку алгоритму за допомогою випадкової їх ініціалізації.

г) Окремою модифікацією вважають **метод відбору відсіченням** (truncation), в якому особини популяції впорядковуємо спадаючим чином і задаємо поріг пристосованості.

Серед всіх осіб, фітнес яких більше за це значення, випадково вибираємо одну і поміщаємо її в нову популяцію; повторюємо цю процедуру необхідне число разів.

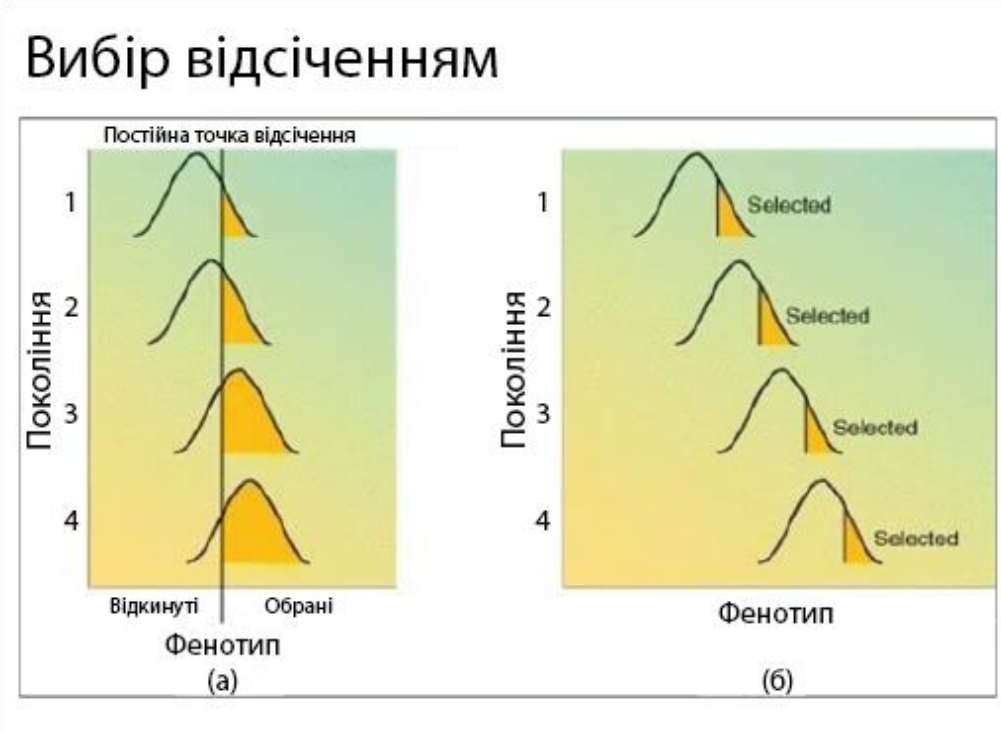

Рис. 2.3 – Схема вибору відсіченням

При такому підході цікавим виявляється деталь, що певні особи можуть потрапити декілька разів у проміжну популяцію, а якісь – не потраплять ні разу.

д) **Метод турнірного відбору** – визначають, як більш ефективний, ніж метод рулетки.

Він базується на випадковому формуванні деякого числа груп із кількістю особин в кожній, яка називається розміром турніру. Вибираємо по особі з найкращою пристосованістю з кожної групи та додаємо її до проміжної популяції.

Величина групи визначає тиск селекції (selection pressure). Чим вища пристосованість особини , тим більша імовірність її потрапляння в проміжну популяцію. А в найпростішому випадку – коли розмір турніру становить один – цей метод забезпечує випадковий відбір.

е) Метод рангового відбору.

Визначивши пристосованість усіх особин популяції, сортуємо їх у порядку спадання. Після цього присвоюємо кожній особині ранг, який дорівнює її номеру у відсортованій популяції; ставимо у відповідність рангу імовірність, яка є деякою простою спадною функцією від останнього. Це може бути лінійна, експоненційна функція тощо. Останнім кроком на основі цих імовірностей вибираємо необхідну кількість геномів за допомогою методу рулетки. Перевага цього алгоритму полягає у гарантованій різноманітності популяції на усіх ітераціях, що урізноманітнює результати і забезпечує уникнення потрапляння в лише один екстремум.

2.4 Схрещування

Після етапу селекції батьків відбувається етап схрещування. Тут для кожної пари певним чином вибираємо так звану «точку схрещування», що означає вибір номеру гена, який буде ключовим у визначенні того, які з генів батьків отримає нащадок. В результаті отримуємо таких нащадків:

а) нащадок, гени якого від першого до точки схрещування складаються із генів першого члена пари, а ті, що залишились – з генів іншого.

б) нащадок, гени якого від першого до точки схрещування складаються із генів другого із батьків, а інші – з генетичного матеріалу першого.

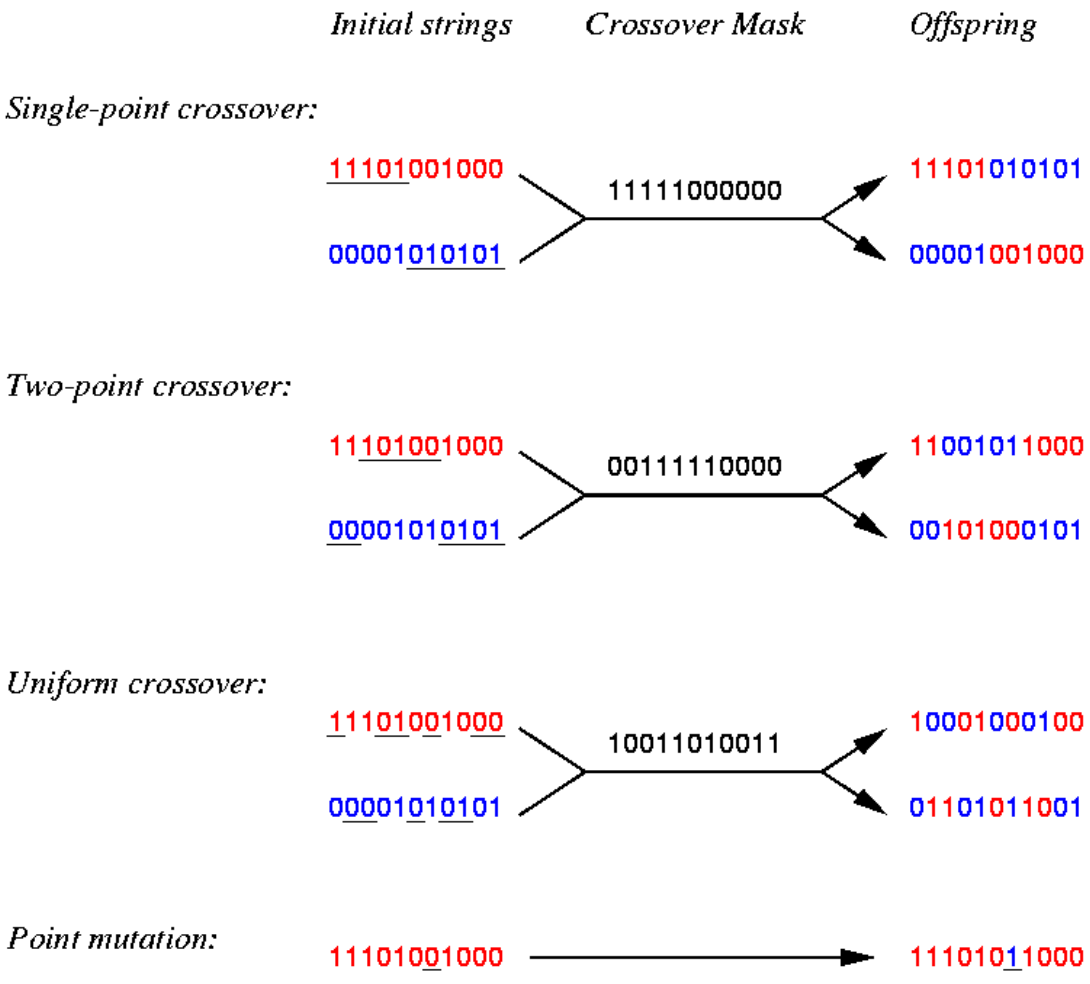

Рис. 2.4 – Приклади операторів мутації

На першому кроці цього оператора необхідно вибрати пари хромосом з батьківського пулу. Такі особи зазвичай вибираються зі всієї популяції, хоча існує і такий варіант, коли батьками можуть стати лише члени проміжної популяції, відібрані на попередньому кроці.

Для власне ж схрещування існує багато варіантів – деякі з них наведені на рисунку (2.4): одноточковий, двоточковий, однорідний та точковий мутатори.

а) **Метод панміксії** – цей оператор відбору – найпростіший.

Він заключається у випадковому рівноймовірному вибору пари батьків із проміжної популяції: із пронумерованої популяції кожному з номерів у відповідність ставиться інший номер, обраний випадково: пари з однаковими числами не схрещуються. Особливість метода полягає в тому, що будь-яка особа з популяції може декілька разів стати членом пари.

У методу є і недолік: критичність до розміру популяції, адже з його збільшенням зменшується ефективність алгоритму знижується.

б) **Метод селективного відбору**. В цьому випадку батьками можуть стати лише ті особи, які мають вищу за середню пристосованість популяції. Більш того, всі члени, які задовільняють вказаній особливості, можуть утворити пару з рівною імовірністю.

в) **Інбридинг** – метод на основі генотипу, в якому першого з батьківської пари вибирають випадково, а другого – такого, який найбільш схожий за генотипом на першого.

В якості мірила близькості при бінарному кодуванні генів використовують метрику Гемінга (Hamming distance), при дійсному ж кодуванні варто застосовувати різні норми, наприклад, евклідову. Перевага у тому, що в результаті створюються окремі групи хромосом, які досить близкі між собою і відрізняються від осіб інших груп, що дозволяє ефективно знайти як мінімум квазіоптимальне рішення.

Послідовність кроків у цьому методі:

1) Вибираємо випадково першого члена пари з поточної популяції.
2) Обчислюємо відстань від нього до всіх інших осіб популяції.
3) У відповідність кожній з осіб ставимо імовірність, пропорційну відстані.
4) За принципом метода рулетки на основі знайдених величин ймовірностей шукаємо пару для поточної особи. Очевидно, схрещування з самим собою не дасть необхідного результату, тому так робити не варто.

г) **Аутбридинг** – використовуючи схему панміксії, проводимо вибір першої особи з пари.

Другого ж члена пари вибираємо, вирахувавши відстань від першого до усіх членів популяції і обравши того, відстань до якого – найбільша. Метод дуже схожий на інбридинг з відмінністю в тому, що кожній особі ставимо у відповідність прямо пропорційну відстані між парами імовірність.

Аутбридинг забезпечує високу диверсифікацію пошуку, різноманітність популяції та попереджає передчасну збіжність алгоритму.

2.5 Мутації

Цей етап представляє собою процедуру зміни певної кількості елементів нової популяції: у певної частини (зазвичай її величина у відсотках вказана апріорно) геномів за допомогою деяких перетворень змінюються значення генів.

Зміни відбуваються згідно з обраним алгоритмом мутації, яких існує досить велике різноманіття.

Наведемо приклади дійсних мутаторів.

а) **Оператор випадкової мутації**. Він змінює значення деякого гена на випадкове з інтервалу області його значень. Найпростіший мутатор має таку послідовність кроків:

1) Генеруємо необхідну кількість чисел, які позначатимуть елементи популяції, які мають змінитись в ході мутації.
2) Генеруємо випадкове число, яке лежить в межах області значень першого з членів генерації, які мають змінитись.
3) Присвоюємо йому це випадкове значення.
4) Повторюємо кроки 2-3, доки не мутують усі необхідні елементи.

б) Частковим випадком оператора випадкової мутації прийнято вважати **оператор граничної мутації**: поточному гену присвоюється не випадкове значення, а з рівною вірогідністю нижню чи верхню границю його ОДЗ.

в) **Арифметичний оператор дійсного зсуву** (arithmetic real number creep operator). Нове значення гену, що мутує, визначається за формулою:

$$x_i' = x_i - b(2u-1),\ i \in [1:|X|], \tag{2.2}$$

в якій випадкове число u = U_1(0; 1), b – коефіцієнт зсуву, вільний параметр. При цьому нове імовірне значення гену буде розподілене по інтервалу

$$[x_i - b;\ x_i + b].$$

г) **Геометричний оператор дійсного зсуву** (geometrical real number creep operator) визначає нове значення члену популяції визначає за подібною формулою:

$$x_i' = x_i - x_i b(2u-1),\ i \in [1:|X|], \tag{2.3}$$

У цьому випадку розподілення нових значень в інтервалі

$$[x_i * (1 - b); x_i * (1 + b)].$$

При цьому нове значення виявляється пропорційним старому і має такий же знак.

д) **Оператор Гаусівської (Gauss) мутації** має принцип дії такий: заміна батьківського гена на випадкове значення, яке визначається за даною формулою:

$$x_i' = x_i + N_1(m, \sigma);\ i \in [1 : |X|], \tag{2.4}$$

де $N_1(m, \sigma)$ – вектор незалежних випадкових дійсних чисел, які розподілені за нормальним законом з середнім квадратичним відхиленням рівним σ та математичним очікуванням m. Зазвичай встановлюють мат. очікування рівним нулю.

е) BGA-оператор мутації (Breeder Genetic Algorithm operator). Згідно з алгоритмом, нове значення знаходимо випадково в деякому околі особини за формулою:

$$x_i' = x_i \pm b\left(x_i^+ - x_i^-\right)\delta_i,\ i \in [1 : |X|]. \tag{2.5}$$

Тут b – вільний параметр оператора, знаки плюс і мінус вибирають з рівною імовірністю, а величину δ_i можна визначити, наприклад, за такою формулою:

$$\delta_i = 2^{-k\,u} \in [2^{-k}; 1] \tag{2.6}$$

В даному випадку u = $U_1(0; 1)$ – випадкове число, k – вільний цілий параметр на діапазоні значень [4;20]. Він визначає можливі значення параметру δ_i.

ж) **Степеневий оператор мутації.** Визначає нове значення гена за формулою, наведеною нижче:

$$x_i' = \begin{cases} x_i - u_1(x_i - x_i^-), & \delta_i < u_2, \\ x_i + u_1(x_i^+ - x_i), & \delta_i \geq u_2, \end{cases} \quad i \in [1 : |X|] \tag{2.7}$$

,

де - вільний параметр, який визначає інтенсивність мутацій;

u_1 – випадкова величина, щільність розподілу якої визначається за формулою:

$$\xi(v) = bv^{b-1} \tag{2.8}$$

v ∈ [0; 1],

$u_2 = U_1[0; 1]$ – випадкове число,

$$\delta_i = \frac{x_i - x_i^-}{x_i^+ - x_i} \tag{2.9}$$

з) **Оператор нерівномірної мутації Михалевича** (Michalewicz's non-uniform operator).

Цей оператор являється нестаціонарним мутатором та відноситься до відповідного класу подібних до нього мутаторів. Це означає, що розмір мутації напряму залежить від номеру покоління. Формула, що визначає нове значення гену, така:

$$x_i' = \begin{cases} x_i + \delta(t, x_i^+ - x_i), & u_1 = 0, \\ x_i - \delta(t, x_i - x_i^-), & u_1 = 1, \end{cases} \quad i \in [1 : |X|] \tag{2.10}$$

Тут $u_1 = u_{sign}^{\pm}$ - це випадкове ціле число, яке з рівною імовірністю приймає значення нуль або один, x_i^+ -права границя ОДЗ поточного гену, x_i^- - ліва границя його ОДЗ. Наступна ж формула визначає функцію *δ(t, y):*

$$\delta(t, y) = y\left(1 - u_2^{\left(1-\frac{t}{\hat{t}}\right)^b}\right) \tag{2.11}$$

де $u_2 = U_1[0; 1]$ – випадкове число, b – вільний параметр, який визначає силу залежності нового значення від номеру покоління.

При обрахуванні області значень функції *δ(t, y)* виявляється, що вона приймає значення з інтервалу [0; y]. Більше того: чим більший номер покоління, тим вужчий даний інтервал. Із цього можемо зробити висновок, що в результаті роботи даного алгоритму на перших його стадіях отримані значення більше нагадують випадковий мутатор, а на більш пізніх відбуваються мутації, в результаті яких нове і старе значення є досить близькими один до одного.

и) **Динамічний оператор мутації** – визначає імовірність мутації гену, використовуючи оцінку близькості між батьківськими особинами s_1, s_2, яку вираховує за формулою:

$$\xi_m(s_1, s_2) = c\ \rho(s_1, s_2), \tag{2.12}$$

де c – вільний параметр, який бажано встановлювати рівним 0,9;

$$\rho(s_1, s_2) = 1 - \frac{1}{|X|} \sum_{i=1}^{|X|} \left| \frac{x_{1,i} - x_{2,i}}{x_i^+ - x_i^-} \right|^b, \quad (2.13)$$

де |X| – кількість генів одного елемента, b – вільний член оператора; значення, що рекомендують йому присвоювати дорівнює 0,2.

Наведені та інші алгоритми мутації запроваджують для того, аби урізноманітнювати значення елементів популяції: як в живій природі мутації інколи призводять до підвищення адаптивних властивостей організмів, покращують їх спротив несприятливим умовам і чинникам.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ АЛГОРИТМІВ

3.1 Мова програмування та середовище розробки

Завдання було виконане в програмі для персонального комп'ютера Microsoft Visual Studio, яка містить в собі інтегроване середовище для можливості розробки програмного забезпечення, а також широкий ряд додаткових інструментальних засобів, за допомогою яких є змога розробляти веб-сайти, веб-служби і веб-застосунки як у рідному, так і в керованому коді для усіх доступних на сьогодні платформ, а також програми з графічним інтерфейсом (так само і з використанням технології Windows Forms) та консольні застосунки.

Для реалізації використані засоби мови програмування C++, тому що серед іншого цією мовою можна писати програми будь-якої складності, вона має переваги в області швидкості виконання коду, а також він є статично типізованим і підтримує низькорівневу роботу з пам'яттю.

3.2 Опис ключових частин програми

В коді проекту встановлюємо необхідні параметри для роботи алгоритму:

а) розмір популяції – кількість особин у кожному з поколінь. Чим більше елементів, тим точніше працюватиме алгоритм, проте перебільшувати не варто: цей параметр також дуже впливає на час роботи програми.

б) кількість поколінь: алгоритм виконує задачу визначену кількість разів і видає результати, отримані на останньому кроці. Вважається, що ці значення найкраще із знайдених вирішують поставлене завдання.

в) кількість генів: в даному випадку гени – це змінні заданої функції. Обрана їх кількість – два для можливості наглядно представити вигляд функції і продемонструвати точність результатів.

г) ймовірність кросинговеру визначає, який відсоток елементів популяції проходитимуть процедуру схрещування.

д) ймовірність мутації визначає, який відсоток членів кожного покоління будуть мутувати, змінюючи свій геном.

е) шість змінних, що приймають булеві значення і вказують на використання тої чи іншої модифікації алгоритму: істина (true), коли використовується одна з модифікацій і хиба (false), коли вона не застосовується.

Інший важливий параметр - структура, яка задає особину як таку, тобто генотип. Дана структура складається із двохелементного масиву значень генів, величини пристосованості, масивів областей допустимих значень генів, а також кумулятивного та відносного значень фітнесу.

Такий вибір оформлення даних не займає багато пам’яті – оскільки не зберігає гігантських масивів з наявними значеннями всіх поколінь, а також дозволяє без зайвих зусиль орієнтуватись у популяції і просто отримувати дані кожного з елементів.

Генотип представлений у вигляді структури на рис. (3.1).

```
struct genotype
{
    double gene_List[XNUM], fitness,
        upLimit[XNUM], lowLimit[XNUM], rfitness, cfitness;
};
```

Рисунок 3.1

3.3 Однорідний генетичний алгоритм

Слід зауважити, що під назвою «однорідний генетичний алгоритм» будемо мати на увазі такий, що має однорідний оператор мутації, селекції та схрещування. Назва «неоднорідний» же вказуватиме на зворотнє – алгоритм з неоднорідним типом мутації та/або схрещування, селекції.

Тож, у цьому розділі представимо деталі реалізації базового генетичного алгоритму, який пізніше буде модифікований зі збереженням основного принципу роботи.

Порівняння ефективності однорідного генетичного алгоритму було проведено з його зміненими версіями, в яких мутація відбувалась за різними схемами. Крім того, були реалізовані два власних варіанти мутації, покликані покращити результат.

Перед нами стоїть задача знаходження екстремуму функції на будь-якому заданому проміжку. Для прикладу візьмемо функцію двох змінних, яка має такий вигляд:

$$f(x,y) = 10 + x^2 - 10\cos(2\pi x) + y^2 - 7\cos(4\pi y) \qquad (3.1)$$

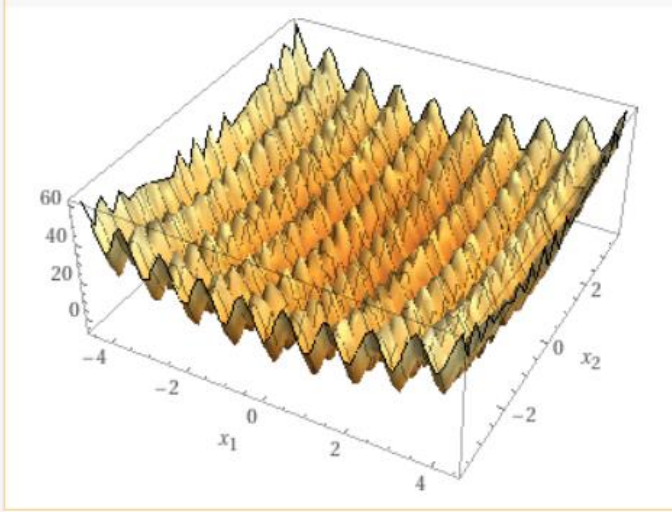

Рис. 3.2 – Графік заданої функції

При розгляді контурного представлення функції (прирівнявши її до нуля) отримуємо такий графік:

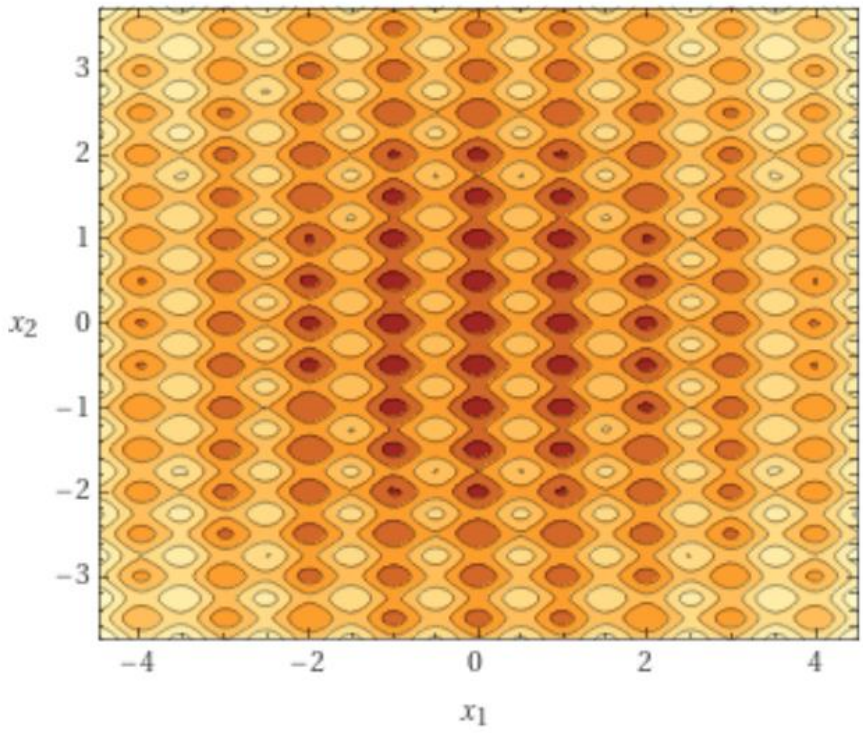

Рис. 3.3

Тож поговоримо детальніше про подробиці реалізації.

Початкову популяцію ініціалізуємо випадковими значеннями з ОДЗ кожного гену, яке встановлює користувач в окремому файлі; пристосованості встановлюємо рівними нулю. Алгоритм селекції, що застосовується далі, представлений модифікованим методом рулетки – методом пропорційного відбору.

Після визначення власне пристосованості кожного з елементів популяції селектор знаходить їх відносне та кумулятивне значення пристосованості. Це дає змогу особам з вищим значенням пристосованості мати більшу імовірність потрапити в нову популяцію. Таким чином зберігаються найкращі значення алгоритму.

```
for (i = 0; i < POPULATION; i++)
{
    p = abRandom8(a, b, seed);
    if (p < population[0].cfitness)
        newpopulation[i] = population[0];
    else
        for (j = 0; j < POPULATION; j++)
            if (population[j].cfitness <= p && p < population[j + 1].cfitness) {
                newpopulation[i] = population[j + 1];
                break;
            }
}
for (i = 0; i < POPULATION; i++)
    population[i] = newpopulation[i];
return;
```

Рис. 3.4 – Частина алгоритму селектора

Для схрещування обираються послідовні пари елементів покоління і обмінюються випадковою кількістю своїх генів. Такий шлях був обраний у зв’язку із малою величиною генів, тому особи обмінюються одним, двома генами чи не обмінюються взагалі.

```
void cross(int& seed)
{
    const double a = 0.0, b = 1.0;
    double x; int mem, one = 0, first = 0;

    for (mem = 0; mem < POPULATION; ++mem)
    {
        x = abRandom8(a, b, seed);
        if (x < XOVER)
        {
            ++first;
            if (first % 2 == 0)
                Xover(one, mem, seed);
            else
                one = mem;
        }
    }
    return;
}
```

Рис. 3.5

Після цього функція мутує із заданою імовірністю. В загальному випадку мутація відбувається зі сталою імовірністю випадковим чином. Останнім кроком алгоритму ми визначаємо найкраще значення функції пристосованості в поточній популяції. Для покращення результату найкраще поточне значення пристосованості зберігається в останньому елементі масиву - і це значення не бере участь в процесі мутації чи схрещування. Якщо воно виявиться меншим, ніж результат функції пристосованості в попередньому поколінні, то ми не втратимо його, а знову запишемо в останній елемент вже поточного покоління.

3.4 Неоднорідний генетичний алгоритм

Проблемою більшості генетичних алгоритмів є збіжність до гомогенної популяції, де майже всі особини мають однакові генотипи, оскільки з деякого моменту, коли знаходиться найкращий результат, наступна популяція забивається її копіями і алгоритм більше не розвивається. Так рішенням може стати мутація.

У роботі проводились спостереження над шістьома варіантами мутацій: арифметичний оператор дійсного зсуву, геометричний оператор дійсного зсуву, BGA-оператор та оператор нерівномірної мутації Михалевича. Принципи їх дії описані у попередньому розділі.

Між тим варто додати, що концентрація уваги саме на мутаціях викликана цікавістю і дослідженнями вчених щодо впливу мутацій на можливість бактерій протистояти впливу зовнішніх факторів: вражаючий експеримент Бейма (Baym) та його колег з з Гарвардської Медичної Школи (Harvard Medical School), де була використана гігантська чашка Петрі з різною концентрацією антибіотиків на різних її частинах зі збільшенням ближче до середини. Потім дослідники помістили в неї бактерії Escherichia coli і впродовж півтора тижні спостерігали, як вони розмножуються. Результати показали, що бактерії з часом мутували і

завдяки цим мутаціям могли переносити дози антибіотиків, які в тисячі разів вищі за необхідні для знищення цих бактерій.

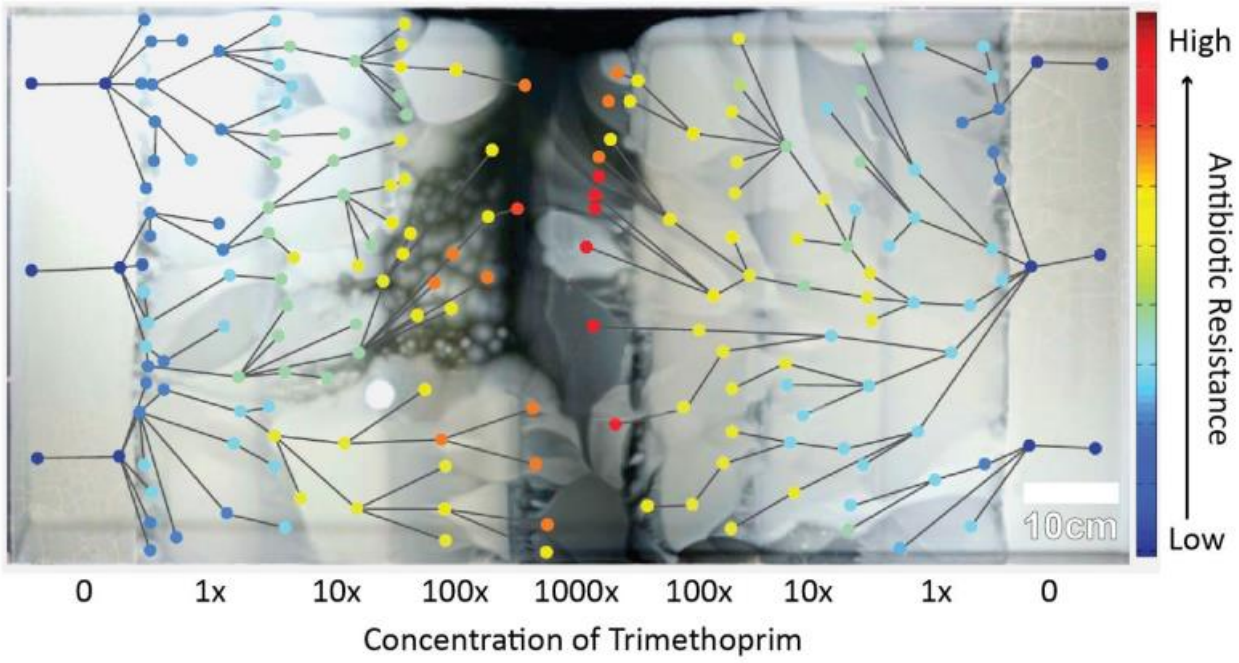

Рис. 3.6 – Результати експерименту

Отже, такі результати вказують на безпрецедентну важливість ролі, яку грають мутації в живій природі, а отже повинні активно використовуватись та досліджуватись в створених штучно алгоритмах, натхненних природою і її успішними моделями поведінки.

Отож, на рисунку (3.7) бачимо реалізацію алгоритму **арифметичного оператору дійсного зсуву**.

```
double B = 5, u, x_old;

for (int i = 0; i < POPULATION; i++){
    for (int j = 0; j < 2; j++){
        x = abRandom8(a, b, seed);
        if (x < mutation_prob){
            u = abRandom8(a, b, seed);
            x_old = population[i].gene_List[j];
            population[i].gene_List[j] += -1 * B * (2.0 * u - 1.0);
            population[i].gene_List[j] = normalize(x_old-B, x_old+B, x_old, population[i].gene_List[j], j);
        }
    }
}
```

Рис. 3.7 – Арифметичний оператор дійсного зсуву

Окремої уваги заслуговує розгляд функції нормалізації отриманого результату, яка необхідна для зсуву нового значення гену: після мутації границі його значень можуть не співпадати з ОДЗ даного гена і виходити за його межі. Для уникнення цього явища вираховуємо можливі границі нового значення і стискаємо або розширяємо отриманий інтервал до розмірів вхідного і відповідно змінюємо результат мутації.

```
double normalize(double lb_new, double ub_new, double x_old, double x, int i)
{
    double proportion = (double)(x_old - lbound[i]) / (ubound[i] - lbound[i]),
        prop_new = (x - lb_new) / (ub_new - lb_new), res;

    res = x * proportion;
    res += lbound[i] - lb_new;

    return res;
}
```

Рис. 3.8 – Функція нормалізації

Наступний алгоритм, що був розглянутий – **BGA-оператор**. Швидкість його мутації обернено пропорційна кількості параметрів, які необхідно оптимізувати, а діапазон для мутації – фіксований. Був обраний такий його варіант, в якому значення δ_i з формули (2.5) обчислюється так:

$$\delta = \sum_{i=0}^{15} \alpha_i 2^{-i} \quad \alpha_i \in 0,1 \tag{3.2}$$

Перед мутацією значення усіх α_i встановлюється рівним нулю. Потім кожен α_i мутує до одиниці з імовірністю одна шістнадцята. Таким чином в середньому δ_i складатиметься лише з одного степеня числа два. Після обрахунку нових значень за заданими формулами (2.5) та (3.2), а також областей, на яких вони приймають значення, відбувається процес нормалізації.

```
for (int i = 0; i < POPULATION; i++) {
    for (int j = 0; j < 2; j++)
    {
        x = abRandom8(a, b, seed);
        if (x < mutation_prob) {
            alpha_prob = rand() % 16;
            alpha[alpha_prob] = 1;

            for (int k = 0; k < 16; k++) {
                delta += alpha[k] * pwr_of_two;
            }

            chosen_sign = rand() & 1;
            x_old = population[i].gene_List[j];
            population[i].gene_List[j] += sign[chosen_sign] * mut_range_len[j] * delta;

            if (sign[chosen_sign] == -1) { // if minus
                population[i].gene_List[j] = normalize(lb_minus[j], ub_minus[j],
                                                       x_old, population[i].gene_List[j], j);
            }
            else { //if plus
                population[i].gene_List[j] = normalize(lb_plus[j], ub_plus[j],
                                                       x_old, population[i].gene_List[j], j);
            }
        }
```

Рис. 3.9 – частина BGA-алгоритму

За обрахунками виявилось, що при рандомному виборі знака мінус у формулі (3.9) інтервал можливих значень має вигляд:

$$[(11 * lbound - ubound) * 0.1;\ ubound - \frac{(ubound - lbound) * 0.1}{2^{15}}],$$

де lbound – нижня границя ОДЗ поточного гена, ubound – права границя ОДЗ поточного гена.

У випадку ж обрання знаку плюс цей інтервал виглядатиме так:

$$[\frac{(ubound - lbound) * 0.1}{2^{15}} + lbound;\ (11 * ubound - lbound) * 0.1].$$

Тож функції нормалізації подаємо на вхід ці значення разом із щойно створеним мутованим геном і його старим значенням – і отримуємо результат.

Крім того, був розглянутий **геометричний оператор дійсного зсуву**: його принцип дії схожий на арифметичний оператор дійсного зсуву:

```
double B = 5, u, x_old;

for (int i = 0; i < POPULATION; i++) {
    for (int j = 0; j < 1; j++) {
        x = abRandom8(a, b, seed);
        if (x < mutation_prob) {
            u = abRandom8(a, b, seed);
            x_old = population[i].gene_List[j];
            population[i].gene_List[j] += -1 * B * x_old * (2.0 * u - 1.0);
            population[i].gene_List[j] = normalize(x_old * (1 - B), x_old * (1 +  B),
                                                   x_old, population[i].gene_List[j], j);
        }
    }
}
```

Рис. 3.10

Головна відмінність у тому, що нова величина пропорційна старому значенню гена і має той же знак. При цьому результат все одно необхідно приводити до нормального вигляду.

Ще один реалізований алгоритм – **оператор нерівномірної мутації Михалевича**. Він був розглянутий, оскільки він відноситься до класу нестаціонарних мутаторів і використовує у своїх розрахунках прив'язку до покоління, що на перших кроках робить його подібним до випадкового алгоритму мутації, але чим далі – тим більша спостерігається близькість між початковим та отриманим значеннями, що є досить вдалим рішенням, адже чим більше поколінь пройшло, тим більше шанс, що алгоритм знайде близьке до справжнього рішення – тому на більш пізніх кроках немає сенсу сильно відходити від знайдених геномів. Порахувавши інтервали, на яких можуть приймати значення мутовані за цим алгоритмом гени, виявляється, що функція нормалізації не потрібна – адже при $\xi=0$ цей інтервал має вигляд [x; $ubound$], а при ξ – [$lbound$; x], де x – значення гена до мутації, що означає, що мутоване значення ніколи не вийде за межі заданих рамок.

Різноманітність популяції можна підтримувати як завгодно довго, якщо використовувати різні мутатори. Ідея полягає в тому, щоб зробити алгоритм мутації залежним від поточного значення гену – тобто від власного стану особини

у клітинному просторі. Таким чином в деяких областях величина мутації буде більшою, в інших – досить малою, тому перші з них будуть генерувати більш випадкові рішення, підтримуючи різноманітність рішень, а області іншого типу – використовуватись для кросинговеру та схрещування.

Для перших тестів було вибрано імовірність мутації за формулою (3.3).

$$p(mutation) = 4 \cdot \frac{\text{MUTATION}}{1 + |x|} \tag{3.3}$$

для компоненти x (та аналогічна формула для компоненти y), де імовірність мутації MUTATION = 0.5.

Наведемо також графік імовірності мутацій за такою формулою на рис. 3.11.

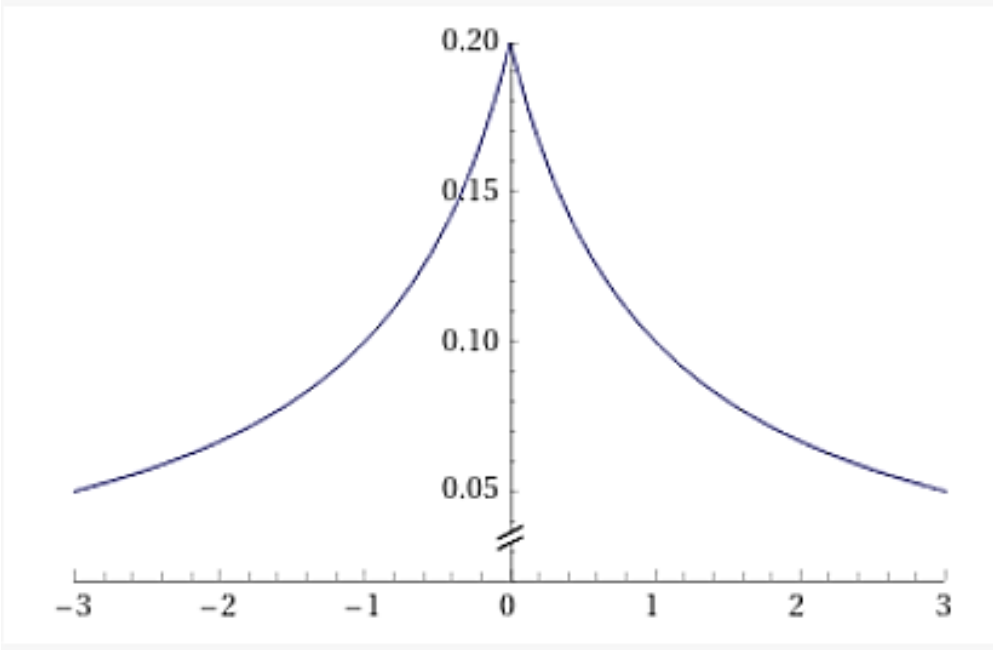

Рисунок 3.11 – Графік розподілу мутацій за першою формулою

Друга формула, яка залежна від координати точки і за якою обчислюється імовірність мутації, представлена формулою (3.4).

$$p(mutation) = 4 \cdot \frac{\text{MUTATION}}{4 - |x|} \tag{3.4}$$

для компоненти x (та аналогічна формула для компоненти y), де MUTATION = 0.05.

Графік цих імовірностей мутації представлений на рис. 3.12.

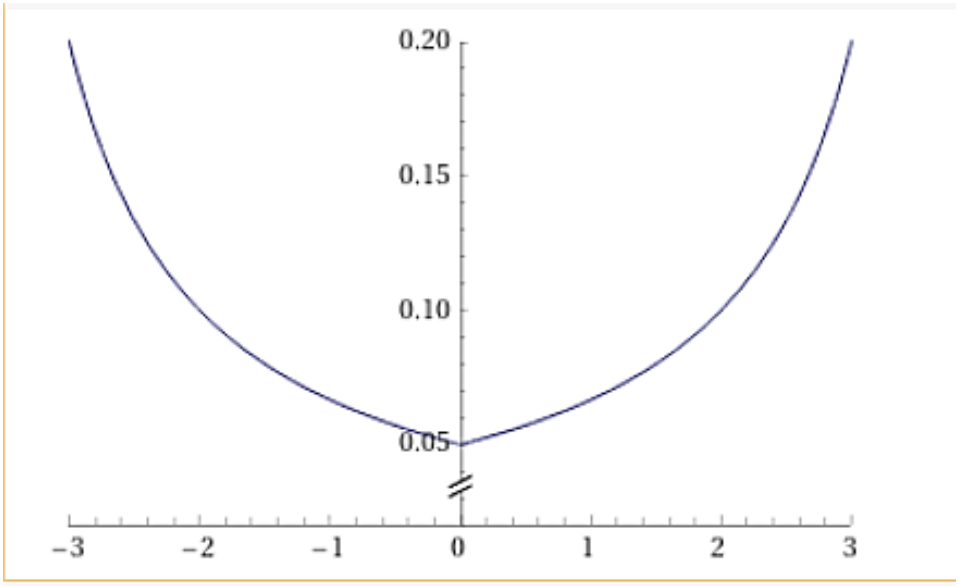

Рисунок 3.12 – Графік розподілу мутацій за другою формулою

РОЗДІЛ 4. ПОРІВНЯННЯ РЕЗУЛЬТАТІВ

Представимо результати роботи різних алгоритмів. Для експериментів було вибрано три варіанти залежності параметра від координат (а саме: мутація залежить від *x*; від *y*; від *x* та *y*) і протестовано з різними функціями мутацій.

Спочатку порівнюємо, яке значення приймають функції відповідності кожного покоління для різних мутацій, якщо мутація залежить від координат *x* та *y* в кожному з випадків:

а) Алгоритм випадкової мутації

б) Арифметичний оператор дійсного зсуву

в) Геометричний оператор дійсного зсуву

г) BGA-алгоритм

д) Алгоритм Михалевича

е) Перший власний алгоритм мутації

ж) Другий власний алгоритм мутації

Зауважимо також, що фактичний локальний мінімум функції, яку ми досліджуємо, на проміжку [-5; 5] становить мінус сім.

На рис. 4.1 представлений графік найкращих значень функцій пристосованості на кожному з поколінь. Для кращого вигляду порівнянь показані лише значення на проміжку [-7; -5.8].

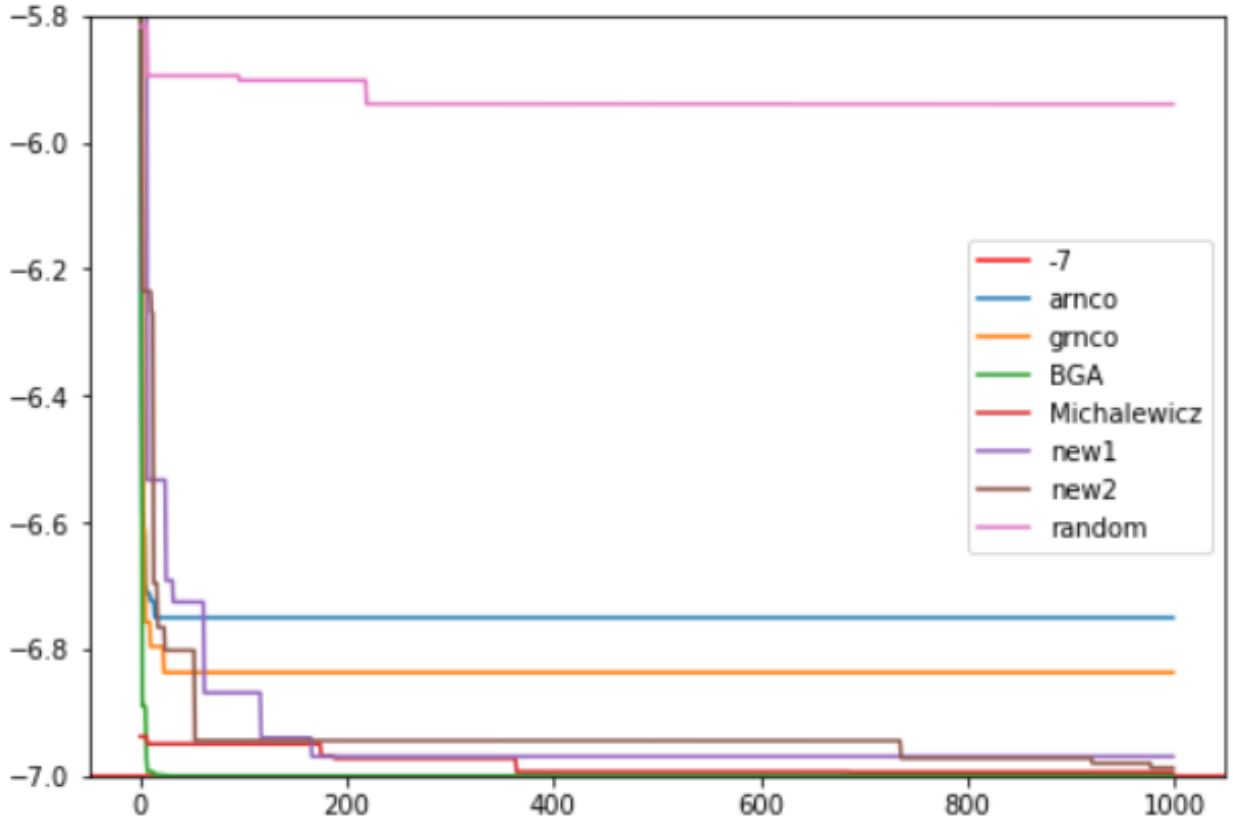

Рис. 4.1

Збільшимо ще сильніше результати. На рис. 4.2 видно, що алгоритми арифметичного і геометричного зсувів, а також алгоритм випадкової мутації «відстали» від інших алгоритмів, застрягнувши в локальних екстремумах. В той же час найкращий результат показав алгоритм BGA, досягнувши глобального мінімуму, наступний найкращий результат показав алгоритм Михалевича, після цього йдуть власні алгоритми мутації.

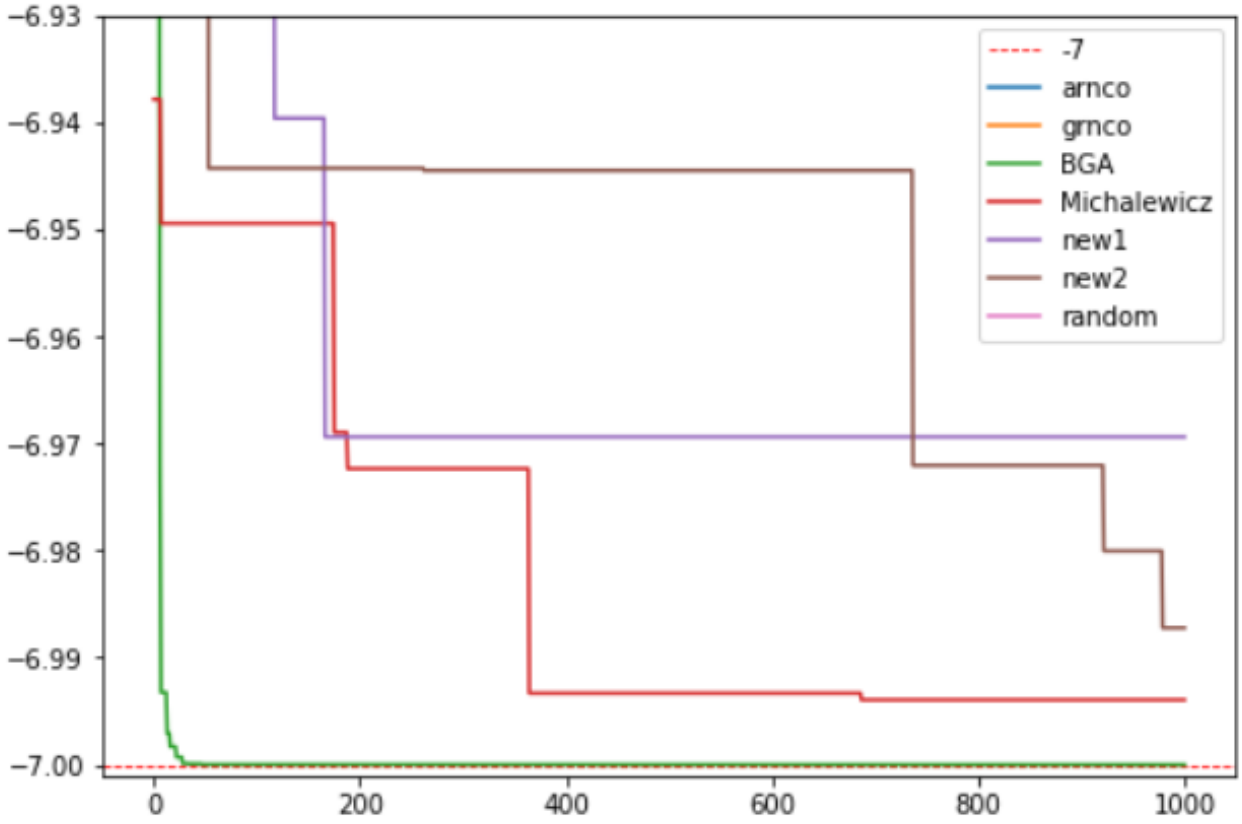

Рисунок 4.2 – Порівняння роботи різних алгоритмів

Між іншим, рисунок дає змогу оцінити швидкість збіжності до найкращого значення: алгоритм BGA, окрім точності, показав ще й дуже стрімке покращення результату. Варто також зазначити, що при проведенні багатьох тестувань на різних проміжках цей алгоритм все ще залишався лідером за швидкістю та точністю.

Алгоритм же Михалевича, як і очікувалось, спочатку нагадує випадковий алгоритм мутації, а в кінці - залишається в локальному екстремумі та не може з нього вийти, адже зі збільшенням поколінь схожість старого і нового значень мутованих генів стають дуже схожими один на одного. Не дивлячись на це, він все ще показує непоганий результат.

Другий тест: порівнюємо, яке значення приймають функції відповідності кожного покоління для різних мутацій, якщо мутація залежить тільки від координати Y, а у випадку звичайного генетичного алгоритму - мутація зі сталою ймовірністю відбувається лише по координаті Y (рис. 9). Наступні тестування проводились зі зміною лише координати X або Y. Отримані результати наведені в таблиці 4.1.

Таблиця 4.1 – Порівняння кінцевих значень цільової функції для різних алгоритмів та компонент мутацій.

	Мутація x	Мутація y	Мутація x, y
Випадкова мутація	-5.57564	-5.37564	-5.989727
ARNCO	-6.39006	-6.35644	-6.750205
GRNCO	-5.80058	-6.37586	-6.836652
BGA	**-6.76727**	**-6.82473**	**-7.000000**
Михалевича	-6.14786	-6.48392	**-6.993943**
Власний 1	**-6.7468**	**-6.79165**	-6.969367
Власний 2	-6.43727	-6.42857	-6.987213

ВИСНОВКИ

В даній роботі було виконане дослідження сучасного стану генетичних алгоритмів і проведено ознайомлення із шляхом їхнього розвитку. Був зроблений детальний огляд різновидів генетичних алгоритмів, вивчені сучасні підходи до вирішення проблем і розглянутий вплив окремих його складових на результат. Зокрема увага була направлена на модифікації оператору мутації та вивчення сили їхнього впливу на вихідний результат, проведений аналіз таких алгоритмів, як:

а) Алгоритм випадкової мутації
б) Арифметичний оператор дійсного зсуву
в) Геометричний оператор дійсного зсуву
г) BGA-алгоритм
д) Оператор нерівномірної мутації Михалевича

У роботі також досліджені результати, які були отримані унаслідок роботи власних двох алгоритмів мутації. Дослідження проводились на результатах мутації однієї, іншої та обох змінних, а також такі результати, що залежать не лише від випадковості чи поточного значення гену, а й від номеру покоління. Крім того, у розглянутих модифікаціях були виявлені їх переваги та недоліки, а також області застосування генетичних алгоритмів.

Зокрема, BGA-оператор має місце тоді, коли важлива точність виконання. Його перевага у тому, що він знаходить нове значення геному в деякому околі старого значення – це важливо тоді, коли функція має багато локальних екстремумів, розташованих досить щільно, і при сильних мутаціях результат ризикує зіпсуватись.

Арифметичний і геометричний оператори дійсного зсуву мають перевагу у тому, що не потребують великих обсягів розрахунків – їхні формули мутацій досить невибагливі і підійдуть для задач з великими кількостями обчислень –

наприклад, для навчання нейронних мереж, де тренування займає дуже багато часу і обчислювальних можливостей комп'ютера. Для цих цілей згодяться і алгоритми випадкової мутації – з ними мережі навчатимуться ще швидше. Проте в даному випадку застосовувати лише ГА з мутаціями може зіграти не на руку розробнику: невдалі мутації можуть помітно зіпсувати результати; тому тут варто застосовувати ще й додаткові компоненти генетичного алгоритму – наприклад, як реалізовані в цій роботі: запам'ятовування і передавання в наступні покоління найкращого результату і вилучення найгіршого.

Оператор нерівномірної мутації Михалевича, натомість, працюватиме найкраще при великій кількості популяцій та без збереження найкращого результату, адже тоді він легко застрягне у локальному екстремумі і не зможе досягти кращого результату.

Отже, хоч в генетичних алгоритмах є недоліки, такі, як відсутність гарантій оптимальності результату і погана пристосованість до деяких простих задач, де є доступ до похідної інформації, вони кращі в своїй області - визначення «досить гарного» рішення «досить швидко», що робить їх привабливими для вирішення оптимізаційних проблем, а також NP-складних, рішення яких іншими методами може зайняти навіть *роки обчислень* найпотужнішими системами обчислення.

Результатом роботи є імплементації генетичного однорідного алгоритму та низки його неоднорідних модифікацій в середовищі розробки Microsoft Visual Studio Community 2022, а також аналіз, порівняння отриманих даних та відповідність їх фактичним результатам. Було створено програмний засіб для обчислення екстремуму функції багатьох змінних, а також його модифікації для покращення результатів. Отримані значення вказують на те, що є доцільною подальша розробка та покращення результатів.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Карпенко А. П. Сучасні алгоритми пошукової оптимізації. Алгоритми, натхненні природою / А. П. Карпенко – 2-ге вид. – Москва: Видавництво МГТУ ім. Н. Е. Баумана, 2017. – 73-95.
2. Adaptive probabilities of crossover and mutation in genetic algorithms / M. Srinivas, L. Patnaik – IEEE Transactions on System, Man and Cybernetics, 1994. 656–667 с.
3. Genetic Programming – An Introduction / [Banzhaf, Wolfgang, Nordin та ін.] – San Francisco, CA: Morgan Kaufmann, 1998. ISBN 978-1558605107.
4. Holland, John Adaptation in Natural and Artificial Systems – Cambridge, MA: MIT Press, 1992. ISBN 978-0262581110.
5. Селекція [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Selection_(genetic_algorithm)
6. J. Koza Genetic Programming: On the Programming of Computers by Means of Natural Selection – Cambridge, MA: MIT Press, 1992. ISBN 978-0262111706.
7. G. Harik Learning linkage to efficiently solve problems of bounded difficulty using genetic algorithms (PhD) – Dept. Computer Science, University of Michigan, Ann Arbour, 1997.
8. Генетичний алгоритм [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Genetic_algorithm
9. Symbiogenetic evolution processes realized by artificial methods / Barricelli, Nils Aall – 1957. 143–167.

10. Генетичний загальний алгоритм [Електронний ресурс] – Режим доступу до ресурсу: http://www.znannya.org/?view=ga_general
11. Vijini Mallawaarachchi How to define a Fitness Function in a Genetic Algorithm : [Електронний ресурс] : Towards Data Science – 2017 – Режим доступу до ресурсу: https://towardsdatascience.com/how-to-define-a-fitness-function-in-a-genetic-algorithm-be572b9ea3b4
12. V. Michael The Simple Genetic Algorithm: Foundations and Theory. Cambridge, MA: MIT Press, 1999. ISBN 978-0262220583.
13. Schmitt, M. Lothar Theory of Genetic Algorithms – Theoretical Computer Science, 2001. 259 (1–2): 1–61.
14. Whitley D. A genetic algorithm tutorial. Statistics and Computing - 1995. 4 (2): 65–85.
15. A Field Guide to Genetic Programming / [Poli R., Langdon W. B., McPhee N. F. та ін.], 2008. ISBN 978-1-4092-0073-4.
16. Fraser Alex Simulation of genetic systems by automatic digital computers. I. Introduction / Aust. J. Biol. Sci. 1954. 10 (4): 484–491.
17. Crosby, Jack L Computer Simulation in Genetics. London: John Wiley & Sons, 1973. ISBN 978-0-471-18880-3.
18. Markoff John What's the Best Answer? It's Survival of the Fittest / New York Times / retrieved 13 July 2016.
19. A. Fraser, D. Burnell Computer Models in Genetics / New York: McGraw-Hill, 1970. ISBN 978-0-07-021904-5.
20. Falkenauer Emanuel Genetic Algorithms and Grouping Problems / Chichester, England: John Wiley & Sons Ltd, 1997. ISBN 978-0-471-97150-4.