МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

СЕКЦІЯ ІКТ

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Порівняльний аналіз алгоритмів чисельного розв'язання задач нелінійного програмування. Генетичний алгоритм»

Завідувач

випускаючої кафедри **Довбиш А. С.**

Керівник роботи **Шаповалов С. П.**

Студент групи ІКмз – 91с **Грицюк В. В.**

СУМИ 2020

Сумський державний університет

(назва вузу)

Факультет *ЦЗДВФН* Кафедра *Комп'ютерних наук*

Спеціальність *«Інформатика»*

Затверджую:

зав.кафедрою _____

_____

"_____" _____ 20___р.

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Грицюк Ігорю Миколайовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *«Порівняльний аналіз алгоритмів чисельного розв'язання задач нелінійного програмування. Генетичний алгоритм»*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми. Постановка задачі дослідження. 2) Інформаційний огляд. 3) Математична модель та вибір методу рішення 4) Розробка інформаційного та програмного забезпечення системи*

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

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Аналіз проблеми. Постановка задачі дослідження*		
2.	*Інформаційний огляд*		
3.	*Математична модель та вибір методу рішення*		
4.	*Розробка інформаційного та програмного забезпечення системи*		
5.	*Оформлення пояснювальної записки до дипломної роботи*		

Студент – дипломник _____

(підпис)

Керівник проекту _____

(підпис)

РЕФЕРАТ

Записка: 43 стор., 10 рис., 1 додаток, 9 джерел інформації.

Об’єкт дослідження — задачі нелінійного програмування.

Мета роботи — адаптація генетичного алгоритму до рішення завдань нелінійного програмування.

Методи дослідження — математичне моделювання, генетичний алгоритм, комп’ютерна реалізація алгоритмів на ЕОМ.

Результати — розроблено інформаційне та програмне забезпечення комп’ютерного чисельного розв’язання задач нелінійного програмування генетичним алгоритмом. Проведено огляд відомих рішень, обрано генетичний алгоритм для розв’язання проблеми. Розроблено комп’ютерну реалізацію за допомогою алгоритмічної мови програмування С++.

НЕЛІНІЙНЕ ПРОГРАМУВАННЯ, ГЕНЕТИЧНИЙ АЛГОРИТМ,
МАТЕМАТИЧНА ПОСТАНОВКА ЗАДАЧІ,
КОМПЮТЕРНА РЕАЛІЗАЦІЯ, ТЕСТОВІ РОЗРАХУНКИ

ЗМІСТ

ВСТУП

Методи еволюційного моделювання можна розглядати як ефективні засоби вирішення завдань оптимізації, самоорганізації і моделювання [1-9].

Особливе місце тут займає генетичний алгоритм, який містить в собі всі існуючі лінгвістичні конструкції сучасних мов програмування (зокрема можливості паралельного програмування). ГА є відправною точкою для створення його модифікацій – нових генетичних алгоритмів для розв'язання спеціальних прикладних задач.

Задачі нелінійного програмування зустрічаються в прикладних сферах набагато частіше ніж випадок лінійний і вимагають більш складніших алгоритмів для їх рішення [2]. Тому дослідження в застосуванні генетичних алгоритмів для рішення таких задач представляє актуальність.

Таким чином, технологія еволюційного моделювання дозволяє за допомогою використання природних методів еволюції створити розгорнуті інформаційні системи та алгоритми, причому маючи обмежений набір вхідних даних та параметрів. При цьому із застосуванням технології об'єктно-орієнтованого програмування можна реалізувати перелічені вище задачі максимально гнучко і зручно для програміста. Такі технології є ефективними за подальшого їх впровадження та супроводження в ІС. До таких проблем відносяться, наприклад, задачі проектування технічних систем або систем, що впливають на навколишнє середовище.

Генетичний алгоритм – один з новітніх алгоритмів, але не єдино можливий спосіб вирішення завдань нелінійної оптимізації. З давніх-давен відомі два основних шляхи вирішення таких завдань - переборний і локально-градієнтний. У цих методів свої переваги і недоліки, і в кожному конкретному випадку слід подумати, який з них вибрати. Тому дослідження щодо адаптації генетичного алгоритму для рішення задач нелінійного програмування є актуальними.

Застосування наукового підходу до проблеми реального світу здебільшого означає, насамперед, математичне моделювання проблеми, можливо, з жорсткими обмеженнями, ідеалізаціями або спрощеннями, потім розв'язання

математичної задачі і, нарешті, висновок про реальну проблему на основі розв’язків математична задача.

Приблизно з 60 років відбулася зміна парадигм - в якомусь сенсі в моду ввійшов протилежний шлях. Справа в тому, що світ досяг успіху навіть у ті часи, коли про математичне моделювання нічого не було відомо.

Точніше кажучи, у нашому світі існує величезна кількість надзвичайно витончених процесів та механізмів, які завжди викликали інтерес дослідників завдяки своїй чудовій досконалості. Математично наслідувати такі принципи та використовувати їх для вирішення ширшого класу задач виявилось надзвичайно корисним у різних дисциплінах. Просто коротко, згадаємо наступні три приклади:

Багато що з того, що ми бачимо і спостерігаємо, можна пояснити єдиною теорією: теорією еволюції через спадковість, мінливість і відбір.

Тому не дивно, що вчені, які займаються комп'ютерними дослідженнями, звернулися до теорії еволюції в пошуках натхнення. Можливість того, що обчислювальна система, наділена простими механізмами мінливості і відбору, могла б функціонувати за аналогією з законами еволюції в природних системах, була дуже приваблива. Ця надія стала причиною появи ряду обчислювальних систем, побудованих на принципах природного відбору.

Звичайно, еволюція біологічних систем не єдиний "джерело натхнення" творців нових методів, що моделюють природні процеси. Нейронні мережі [4] (neural networks), наприклад, засновані на моделюванні поведінки нейронів в мозку. Вони можуть використовуватися для ряду задач класифікації, наприклад, завдання розпізнавання образів, машинного навчання, обробки зображень і ін. Область їх застосування частково перекривається зі сферою застосування ГА.

1 АНАЛІТИЧНИЙ ОГЛЯД ПРОБЛЕМИ

1. 1 Алгоритми та методи розв’язку задач нелінійного програмування

Будь-яка цілеспрямована діяльність пов’язана з процесом прийняття рішень, під яким розуміють задачу пошуку окремого варіанта d є D з множини D альтернатив. Теорія прийняття рішень – це математична дисципліна, яка спрямована на вибір, в деякому розумінні, найкращого (найкращих) з можливих варіантів з урахуванням наявних обмежень.

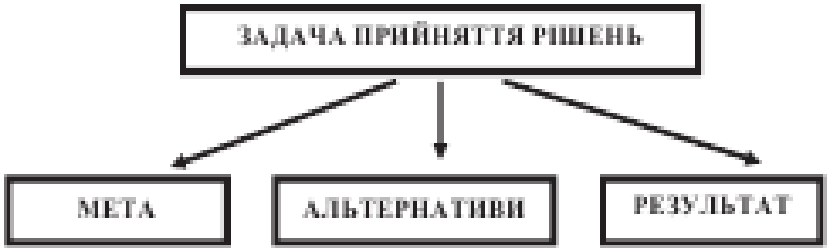

Рисунок 1.1 – Розв’язання задачі прийняття рішень

Математична модель цього процесу можна представити як описання категорійної моделі ⟨ Z, A, V ,D, P ⟩, в якій присутні:

- Z – загальний опис завдання, який повинен формалізувати проблему для рішення та необхідні модельні дані, наприклад, мету або цілі, яких потрібно досягти, а також який кінцевий результат потрібен;
- A – множина альтернативних варіантів, з котрих проводиться вибір. Їх може бути скінчена кількість або множина всіх теоретично можливих варіантів, яка може бути навіть нескінченою;
- V – множина ознак для описання варіантів та їх особливостей. За допомогою ознак характеризують альтернативи;
- D – сукупність умов, що обмежують область допустимих варіантів розв’язку задачі. Обмеження можуть бути описані як змістовним чином, так і формалізовані математично;
- P – переваги, які є основою для оцінювання та порівняння можливих варіантів рішення проблеми, відбору допустимих варіантів і пошуку найкращого або прийнятного варіанту.

Теорія прийняття рішень – це математично змодельована проблема, яка спрямована на вибір, в деякому розумінні, найкращого (найкращих) з можливих варіантів з урахуванням наявних обмежень.

Задача нелінійного програмування ставиться як задача знаходження оптимуму певної цільової функції $F(x_1,x_2,x_3,\ldots x_n)$ при виконанні умов

$G_j (x_1,x_2,x_3,\ldots x_n) \geq 0,$

де $x_1,x_2,x_3,\ldots x_n$ — параметри, Gj — обмеження, n — кількість параметрів, j — кількість обмежень.

В теорії прийняття рішень нелінійне програмування (НЛП) - це процес вирішення задачі оптимізації, де деякі обмеження або цільова функція є нелінійними. Оптимізаційною задачею є розрахунок екстремумів (максимумів, мінімумів або стаціонарних точок) цільової функції за набором невідомих дійсних змінних, що обумовлюється задоволенням системи рівності та нерівностей, які спільно називаються обмеженнями. Ця система задає область допустимих рішень. Саме підполе математичної оптимізації має справу з нелінійними задачами.

На рисунку 1.2 показано графічна інтерпретація задачі нелінійного програмування, система обмежень якої задається нерівностями

$x_1 \geq 0$

$x_2 \geq 0$

$x_1^2 + x_2^2 \geq 1$

$x_1^2 + x_2^2 \leq 2$

а сама функція цілі має лінійну форму $F= x_1 + x_2$

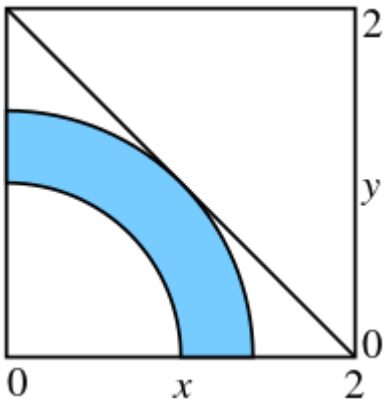

Рисунок 1.2 – Графічна інтерпретація задачі нелінійного програмування

На рисунку область блакитного кольору є областю допустимих рішень. Дотик прямої, що є лінією рівня функції лінії до можливої області представляє рішення (геометричне місце точки із заданим значенням цільової функції).

Типовою непуклою проблемою є оптимізація транспортних витрат шляхом вибору із набору транспортних методів, один або кілька з яких характеризуються економією від масштабу з різними зв'язками та обмеженнями пропускної здатності. Прикладом може бути транспорт нафтопродуктів з урахуванням вибору або комбінації трубопроводу, залізничного танкера, автоцистерни, річкової баржі або прибережного танкера. Внаслідок економічного обсягу партії функції витрат можуть мати розбіжності на додаток до плавних змін.

В експериментальній науці деякі прості аналізи даних (наприклад, пристосування спектра до суми піків відомого місця розташування та форми, але невідомої величини) можна зробити лінійними методами, але загалом ці проблеми також є нелінійними. Як правило, існує теоретична модель досліджуваної системи із змінними параметрами в ній та модель експерименту чи експериментів, які також можуть мати невідомі параметри. Намагається чисельно знайти найкращу відповідність. У цьому випадку часто потрібна міра точності результату, а також найкраща відповідність.

Методи рішення задач нелінійного програмування можна розділити на два класи: прямі методи й не прямі.

Прямими методами оптимальні розв’язки відшукують у напрямку найшвидшого збільшення (зменшення) цільової функції. Типовими для цієї групи методів є градієнтні. Непрямі методи полягають у зведенні задачі до такої, знаходження оптимуму якої вдається спростити. До них належать, насамперед, найбільш розроблені методи квадратичного та сепарабельного програмування.

Рисунок 1. 3– Загальна класифікація методів розв’язку задач нелінійного програмування

Якщо цільова функція є увігнутою (задача максимізації) або опуклою (проблема мінімізації), а набір обмежень опуклий, тоді програма називається опуклою і в більшості випадків можуть використовуватися загальні методи опуклої оптимізації.

Якщо цільова функція квадратична, а обмеження лінійні, використовуються методи квадратного програмування.

Якщо цільова функція є відношенням увігнутої та опуклої функції (у випадку максимізації), а обмеження опуклі, тоді задача може бути перетворена на опуклу оптимізаційну задачу за допомогою методів дробового програмування.

Існує кілька методів вирішення неопуклих проблем. Один із підходів полягає у використанні спеціальних формулювань задач лінійного програмування. Інший метод передбачає використання методів розгалуження та зв’язку, коли

програма поділяється на підкласи, що вирішуються опуклими (проблема мінімізації) або лінійними наближеннями, що утворюють нижчу межу загальної вартості в підрозділі. З подальшими поділами в якийсь момент буде отримано фактичне рішення, вартість якого дорівнює найкращій нижній межі, отриманій для будь-якого з наближених рішень. Це рішення є оптимальним, хоча, можливо, не унікальним. Алгоритм також може бути зупинений достроково, із запевненням, що найкраще можливе рішення знаходиться в межах допуску з найкращої знайденої точки; такі точки називаються ε-оптимальними. Завершення до ε-оптимальних точок, як правило, необхідне для забезпечення кінцевого закінчення. Це особливо корисно для великих, складних проблем та проблем з невизначеними витратами або значеннями, де невизначеність може бути оцінена за допомогою відповідної оцінки надійності.

За умови диференційованості та обмеженості, умови Каруша – Куна – Таккера (KKT) забезпечують необхідні умови для оптимального рішення. За опуклості цих умов також достатньо. Якщо деякі функції недиференційовані, доступні субдиференціальні версії умов Каруша – Куна – Таккера (KKT). [1

Генетичні Алгоритми - адаптивні методи пошуку, які останнім часом часто використовуються для вирішення задач функціональної оптимізації [18-21]. Вони засновані на генетичних процесах біологічних організмів: біологічні популяції розвиваються протягом декількох поколінь, підкоряючись законам природного відбору і за принципом "виживає найбільш пристосований" (survival of the fittest). Наслідуючи цьому процесу генетичні алгоритми здатні "розвивати" вирішення реальних завдань, якщо ті відповідним чином закодовані. Наприклад, ГА можуть використовуватися, щоб проектувати структури моста, для пошуку максимального відношення міцності / ваги, або визначати найменш марнотратне розміщення для нарізки форм з тканини. Вони можуть також використовуватися для інтерактивного управління процесом, наприклад на хімічному заводі, або балансуванні завантаження на многопроцессорном комп'ютері. Цілком реальний приклад: ізраїльська компанія Schema розробила програмний продукт Channeling для оптимізації роботи стільникового зв'язку шляхом вибору оптимальної

частоти, на якій буде вестися розмова. В основі цього програмного продукту і використовуються генетичні алгоритми.

1.2 Постановка задачі

Нехай задано задачу нелінійного програмування в загальному разі.

Поставимо наступне завдання дослідження.

А) Розробити інформаційне та програмне забезпечення застосування генетичного алгоритму до розв’язання завдання:

Б) Провести тестові дослідження та показати застосовність адаптивного генетичного алгоритму, що розв’язує завдання з заданою точністю.

В) Дослідити залежність точності розрахунків від розміру популяцій осіб в генетичному алгоритмі.

2 ВИБІР АЛГОРИТМІВ РІШЕННЯ ПРОБЛЕМИ

2.1 Короткий огляд відомих рішень

Повернемось до поставленої проблеми й алгоритмам та методам, що її зможуть розв'язати [4-5].

Математична модель задачі нелінійного програмування представимо наступне:

максимізувати(мінімізувати) функцію цілі F

$F(x_1, x_2, x_3, \ldots, x_n) \rightarrow \max(\min)$, (2.1)

в якій керовані змінні $(x_1, x_2, x_3, \ldots, x_n)$ вибираються з області допустимих рішень D, що задається сукупністю обмежень

$$\begin{cases} G(\ (x1, x2, x3, \ldots, xn) \geq 0, \\ Q(\ (x1, x2, x3, \ldots, xn) \leq 0, \\ \quad (x1, x2, x3, \ldots, xn) \geq 0 \end{cases} \qquad (2.2)$$

Нелінійність може проявлятись в аналітичному подання як самої функції цілі (2.1) , так і в обмеженнях (2.2), і взагалі й функціонально і в обмеженнях.

Математичні моделі в задачах проектування реальних об'єктів або технологічних процесів повинні відбивати реальні, як правило, нелінійні процеси. Змінні цих об'єктів або процесів пов'язані між собою фізичними нелінійними законами. У результаті, більшість задач математичного програмування, які зустрічаються в науково-дослідних проектах і в задачах проектування – це задачі нелінійного програмування.

Розв'язанням (рішенням) поставленого завдання є пошук в області допустимих рішень набору керованих змінних, яка наддасть функції цілі шуканого екстремуму.

Всі алгоритми рішення такої задачі розбиваються на дві основні групи – аналітичні та чисельні [4-5].

Застосовність аналітичних методів мізерно мала, так як вони вимагають наявності декількох одночасно обмежень: наявність диференційованості функції цілі й очевидно, її неперервності; опуклості області допустимих рішень й теж нерозривності; навіть якщо ці дві перші виконуються, наявність методу

розв’язання одержаних при виконанні умов, як правило системи рівнянь, очевидно нелінійних.

Звичайною сферою застосування даних методів є задачі з відомим аналітичним виразом критерія оптимальності, що дозволяє знайти не дуже складний аналітичний вираз для похідних. Отримані результати прирівнюємо до нуля похідних рівняння, визначаючі екстремальні рішення оптимальної задачі. Це досить рідко вдається розв’язати аналітичним шляхом, тому, як правило, застосовують обчислювальні машини. При цьому потрібно розв’язати систему кінцевих рівнянь, частіше за все нелінійних, для чого потрібно застосовувати числові методи, аналогічні методам нелінійного програмування.

Множники Лагранжа можна застосовувати для рішення задач оптимізації об’єктів з розподіленими параметрами і задач динамічної оптимізації. При цьому замість рішення системи кінцевих рівнянь для відшукання оптимуму необхідно інтегрувати систему диференціальних рівнянь.

Чисельні методи рішення задач нелінійної оптимізації розв’язують завдання з деякою наперед заданою точністю, але вони не потребують (хоча й не всі) виконання тих умов, що описані вище. Єдиним недоліком цих методів є доведення їх збіжності до шуканого результату.

Але цей недолік можна подолати проведенням тестових розрахунків й співставленням результатів з уже відомими результатами.

2.2 Вибір алгоритму рішення

Взагалі кажучи, генетичні алгоритми є імітаціями еволюції, однак у більшості випадків генетичні алгоритми - це не що інше ніж імовірнісні методи оптимізації, які базуються на принципах еволюція.

Ідея генетичного алгоритму з’являється вперше в 1967 р. в дисертації Дж. Д. Беглі «Поведінка Адаптивні системи, що використовують генетичні та корелятивні алгоритми ». Потвм на теорію та придатність алгоритму для різнлгл роду обчислень вплинули праці Дж. Х. Голланд, кого можна вважати піонером генетичних алгоритмів.

Еволюційні алгоритми відрізняються від інших числових методів особливою стратегією пошуку оптимального рішення моделюванням розвитку деякої ситуації замість безпосереднього обчислення відповіді з детермінованих формальних залежностей.

Отже, застосування генетичного алгоритму дає перевагу в невизначених ситуаціях, коли існує декілька доволі вдалих, хоча і неочевидних рішень, при цьому інші методи пошуку рішень виявляються непридатними або малоефективними.

Крім того, такі алгоритми ефективно формують шаблони адаптивної поведінки системи, оскільки хромосоми, які змогли "вижити" в процесі еволюції рішення, зберігають, по суті, колективний досвід багатьох поколінь. Тобто, еволюційне моделювання можна ефективно використовувати в таких випадках :

− застосування еволюційних алгоритмів для вивчення і модифікації окремих процесів природної еволюції;

− покращення наявних інформаційних систем наданням їм властивостей адаптивної поведінки і можливості самоорганізовуватись на основі засобів еволюційного моделювання;

− автоматизація вирішення різноманітних оптимізаційних завдань у науково-технічних галузях.

Загальна блок- схема генетичного алгоритму вказана на рис. 2.1

Задача кодується таким чином, щоб її вирішення могло бути представлено в вигляді масиву подібного до інформації складу хромосоми. Цей масив часто називають саме так «хромосома». Випадковим чином в масиві створюється деяка кількість початкових елементів «осіб», або початкова популяція. Особи оцінюються з використанням функції пристосованості, в результаті якої кожній особі присвоюється певне значення пристосованості, яке визначає можливість виживання особи. Після цього з використанням отриманих значень пристосованості вибираються особи, допущені до схрещення (селекція).

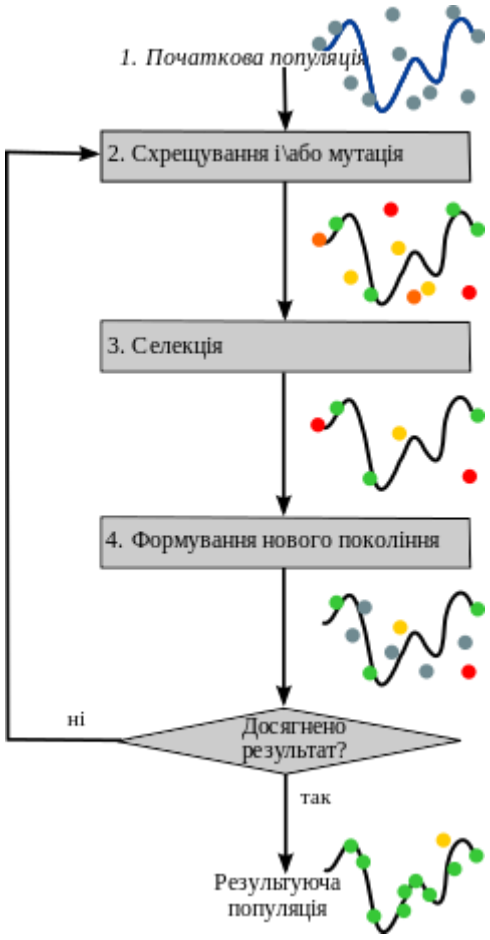

Рисунок 2.1 – Схема генетичного алгоритму

До осіб застосовується «генетичні оператори» (в більшості випадків це оператор схрещення (crossover) і оператор мутації (mutation)), створюючи таким чином наступне покоління осіб. Особи наступного покоління також оцінюються застосуванням генетичних операторів і виконується селекція і мутація. Так моделюється еволюційний процес, що продовжується декілька життєвих циклів (поколінь), поки не буде виконано критерій зупинки алгоритму. Таким критерієм може бути:

- знаходження глобального, або надоптимального вирішення;
- вичерпання числа поколінь, що відпущені на еволюцію;
- вичерпання часу, відпущеного на еволюцію.

Основною операцією в генетичному алгоритмі є схрещування, що дозволяє розвиватись популяції й еволюціонізувати.

На рис. 2.2 показані варіанти цієї процедури.

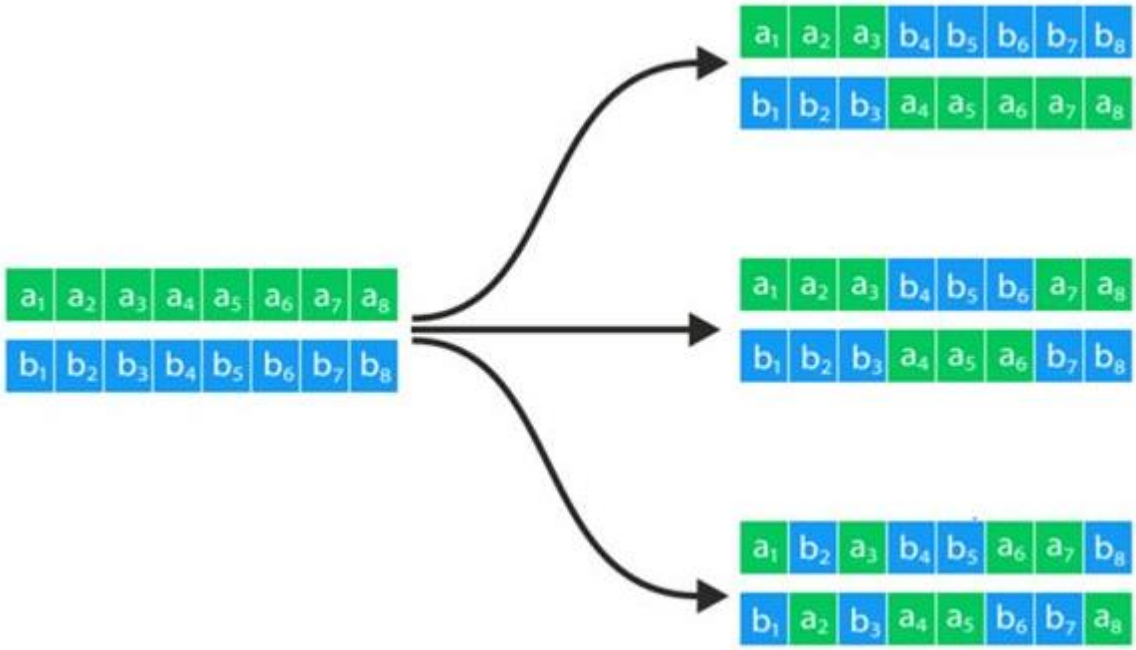

Рисунок 2.2 – Схеми операції схрещення

У генетичному алгоритмі зберігається основний принцип природного відбору - чим пристосованіше індивідуум (чим більше відповідне йому значення цільової функції), тим з більшою імовірністю він буде брати участь у схрещуванні. Тепер моделюються мутації - у декількох випадково обраних особинах нового покоління змінюються деякі гени. Потім стара популяція частково або цілком знищується і ми переходимо до розгляду наступного покоління. Популяція наступного покоління в більшості реалізацій генетичних алгоритмів містить стільки ж особин, скільки початкова, але в силу відбору пристосованість у ній у середньому вище. Тепер описані процеси відбору, схрещування й мутації повторюються вже для цієї популяції і т.д.

Псевдокод генетичного алгоритму легко представити наступне

t: = 0;

Обчислити початкову сукупність B_0;

WHILE умова зупинки не виконана DO

BEGIN

відбирати особин для розмноження;

створювати о ff джерела, схрещуючи особин;

з часом мутують деяких особин;

обчислювати нове покоління

END

Як очевидно з наведеного алгоритму, перехід від одного покоління до наступного складається з чотирьох основних компонентів:

Відбір: Механізм відбору особин (струн) для відтворення відповідно до їхньої щільності (значення цільової функції).

Кросовер: метод злиття генетичної інформації двох особин; якщо кодування обрано правильно, двоє хороших батьків родять хороших дітей.

Мутація: У реальній еволюції генетичний матеріал може випадковим чином змінюватися шляхом помилкового розмноження або інших деформацій генів, наприклад за допомогою гамма-випромінювання. У генетичних алгоритмах мутація може бути реалізована як випадкова деформація струн з певною ймовірністю. Позитивним ефектом є збереження генетичного різноманіття і, як наслідок, того, що можна уникнути локальних максимумів.

У порівнянні з традиційними методами безперервної оптимізації, такими як методи Ньютона або градієнтного спуску, ми можемо стверджувати наступне:

1. GA маніпулюють кодованими версіями параметрів проблеми замість самих параметрів, тобто простір пошуку - S замість самого X.

2. Хоча майже всі звичайні методи здійснюють пошук з однієї точки, GA завжди оперують цілою сукупністю точок (рядків). Це значно сприяє надійності генетичних алгоритмів. Це покращує шанс досягнення глобального оптимуму і, навпаки, зменшує ризик потрапити в пастку в місцевій стаціонарній точці.

3. Звичайні генетичні алгоритми не використовують будь-яку допоміжну інформацію про значення цільової функції, наприклад, похідні. Отже, вони можуть застосовуватися до будь-якої суцільної або дискретної задачі оптимізації.

Єдине, що потрібно зробити, це вказати значущу функцію декодування.

4. GA використовують імовірнісні оператори переходу, тоді як звичайні методи безперервної оптимізації застосовують детерміновані оператори переходу.

Більш конкретно, спосіб обчислення нового покоління з фактичного має деякі випадкові компоненти (ми побачимо пізніше за допомогою деяких прикладів, якими є ці випадкові компоненти)

2.3 Адаптація генетичного алгоритму для рішення задачі нелінійного програмування

Спочатку генерується початкова популяція особин (індивідуумів), тобто деякий набір рішень задачі. Як правило, це робиться випадковим образом. Потім ми повинні змоделювати розмноження всередині цієї популяції. Для цього випадково відбирається декілька пар індивідуумів, відбувається схрещування між хромосомами в кожній парі, а отримані нові хромосоми втілюються в популяцію нового покоління.

Адаптація генетичного алгоритму до рішення задач нелінійного програмування буде полягати в тому, що популяцією буде служити набір рішень задачі, а функцією відбору – функція цілі.

Загальна блок-схема алгоритму наведена на рис. 2.3.

Деякі пояснення до її виконання.

По-перше, початкова популяція формується з хромосом, генами яких є вектор керованих змінних (x_1, x_2, x_3, …, x_n). Це по суті набір точок, що формується генератором випадкових чисел.

По-друге, селекція або відбір визначається за придатності сформованих індивідуумів виконати систему нерівностей (2. 2). Ті особи які пройдуть це «сито» відбору відправляються для подальшого проходження алгоритму, тобто на схрещування та мутацію, інші випадають з алгоритму.

По- третє, функція цілі послуговує в якості фітнес-функції алгоритму, вона якраз і обирає найкращих осіб які є рішенням завдання. З них ми і формуємо нові покоління.

По-третє, останов алгоритму проведемо в момент досягнення заданої точності.

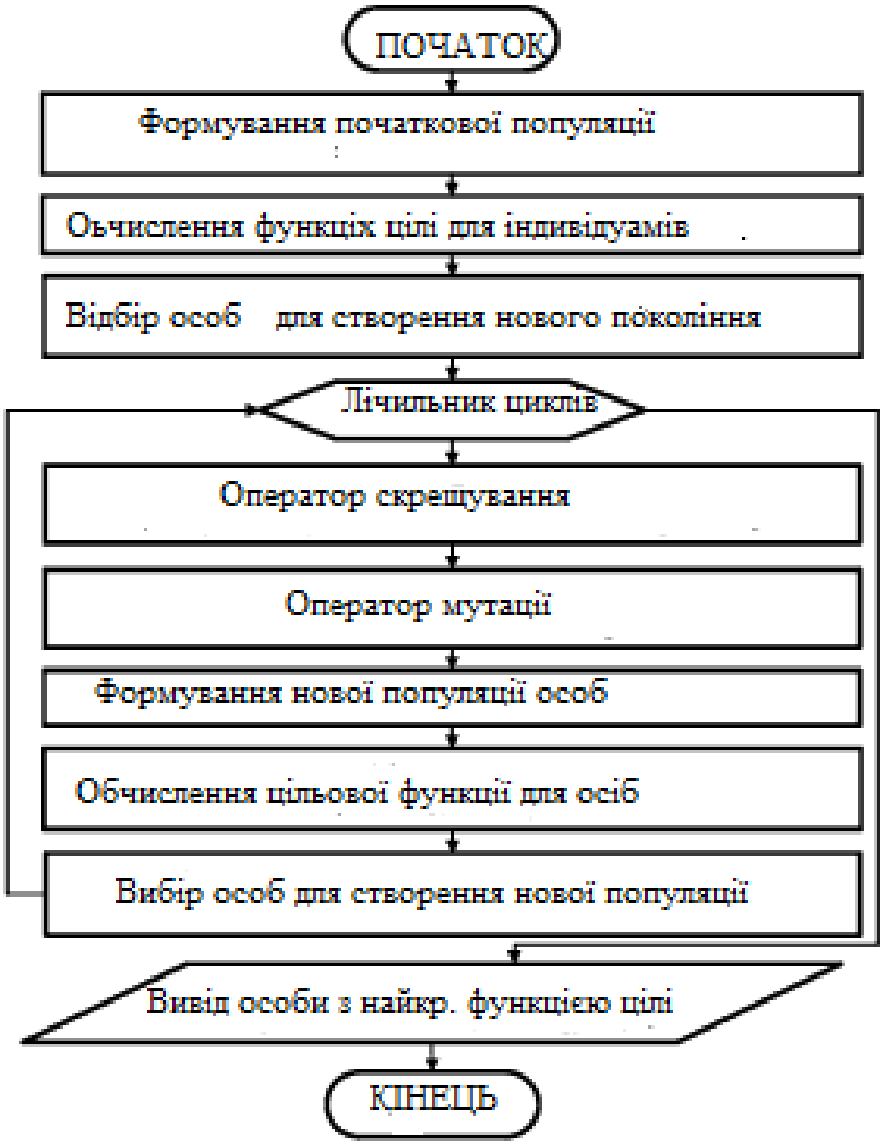

Рисунок 2.3 – Блок схема генетичного алгоритму для застосування в рішенні задач нелінійного програмування

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ТЕСТОВІ РОЗРАХУНКИ

3.1 Комп'ютерна реалізація алгоритмів та основні її складові

Задамо задачу нелінійної оптимізації в загальному разі:

$$\begin{cases} L = \sum_{j=1}^{n} c_j x_j \to \min \ (\max) \ , \\ g_1(x_1, x_2, ..., x_n) \le b_1 , \\ g_2(x_1, x_2, ..., x_n) \le b_2 , \\ ...., \\ g_m(x_1, x_2, ..., x_n) \le b_m , \\ d_j \le x_j \le D_j , \ j = 1,2,..., n. \end{cases} \quad (3.1)$$

Нелінійності можуть виникати як у самій функції цілі L, так і в системі обмежень на пошук рішення.

Реалізуємо пошук рішення задачі генетичним алгоритмом, адаптація якого представлена в пункті 2.3.

Генерація початкової популяції.

Генерування випадкових точок з координатами (x1,x2) типу double :

```
double random(double min, double max){
    return (double)(rand())/RAND_MAX*(max - min) + min;
 }
```

Вибір індивідів із поточної популяції (селекція)

Відбувається за системою (3.1). Ті індивідууми які пройдуть «сито» відбору потравляють для схрещування.

Схрещення.

В кросинговері, що представлений нижче позначення:

v1, v2 - вказівники на батьківські особи.

c1, c2 - нащадки.

```
// класичний кросинговер
    q=rand()*(EBitSize*4-1)/RAND_MAX;
```

```
    for(t=0;t<q;t++)
    {
      SetBit(c1->r,t,GetBit(v1->r,t));
      SetBit(c2->r,t,GetBit(v2->r,t));
    }
    for(t=q;t<EBitSize*4;t++)
    {
      SetBit(c1->r,t,GetBit(v2->r,t));
      SetBit(c2->r,t,GetBit(v1->r,t));
    }
```

Результатом схрещування є нове покоління, яке знову потрапляє на виконання г енетичного алгоритму.

Мутація

```
void mutation(struct Parent Child1,int childnumber)
{
   int mutation; // will be the random number generated

   cout << endl << "Child " << (childnumber+1) << endl;

   //loop through every bit in the binary string
   for (int z = 0; z < Binscale; z++)
   {
      mutation = 0;   // set mutation at 0 at the start of every loop
      mutation = rand()%100;      //create a random number

      cout << "Generated number = " << mutation << endl;

      //if variable mutation is smaller, mutation occurs
      if (mutation < MutationRate)
      {
         if(Child1.binary_code[z] == '0')
```

```
            Child1.binary_code[z] = '1';
          else if(Child1.binary_code[z] == '1')
            Child1.binary_code[z] = '0';
      }
   }
        }
```

Ті особи, що пройшли мутацію додаються до популяції й беруть участь в подальшому виконанні алгоритму.

Принцип останова.

Закінченням дії алгоритму може послужити досягнення в черговій популяції заданої точності обчислення, або задання скільки поколінь повинна пройти популяція, або задання просто часу виконання алгоритму.

Приймемо принцип останова за досягненням точності обчислення.

Комп’ютерна реалізація обраного алгоритму проводилися на комп'ютері з наступною конфігурацією:

- ✓ ПРОЦЕССОР Intel Pentium G3220 CM8064601482519;
- ✓ ОПЕРАТИВНА ПАМ'ЯТЬ (ОРЕ) 2.0 ГБ;
- ✓ ОПЕРАЦІЙНА СИСТЕМА Windows 10;
- ✓ Тип системи 32-розрядна операційна система.

Для дослідження швидкодії алгоритму було реалізовано консольний додаток на мові С ++ в середовищі розробки Microsoft Visual Studio 2010. Консольний додаток надає змогу заміряти процесорний час роботи алгоритму.

3.2 Тестові розрахунки та порівняльний аналіз

Проведемо тестові розрахунки. В якості прикладу оберемо задачу нелінійної оптимізації, що має рішення іншим алгоритмом розв’язання з ціллю тестування можливостей вибраного методу розрахунку.

Застосовуючи генетичний алгоритм й його адаптацію, описану в п. 2.3 розв’яжемо наступне завдання.

Нехай задана функція цілі $F = (x_1 - 4)^2 + (x_2 - 3)^2 \rightarrow \max (\min)$ (3.1)

Область рішень описується системою нерівностей:

$$\begin{cases} 2x_1 + 3x_2 \geq 6 \\ 3x_1 - 2x_2 \leq 18 \\ -x_1 + 2x_2 \leq 8 \\ x_1, x_2 \geq 0 \end{cases} \quad (3.2)$$

Графічна інтерпретація рішення представлена на рис. 3.1.

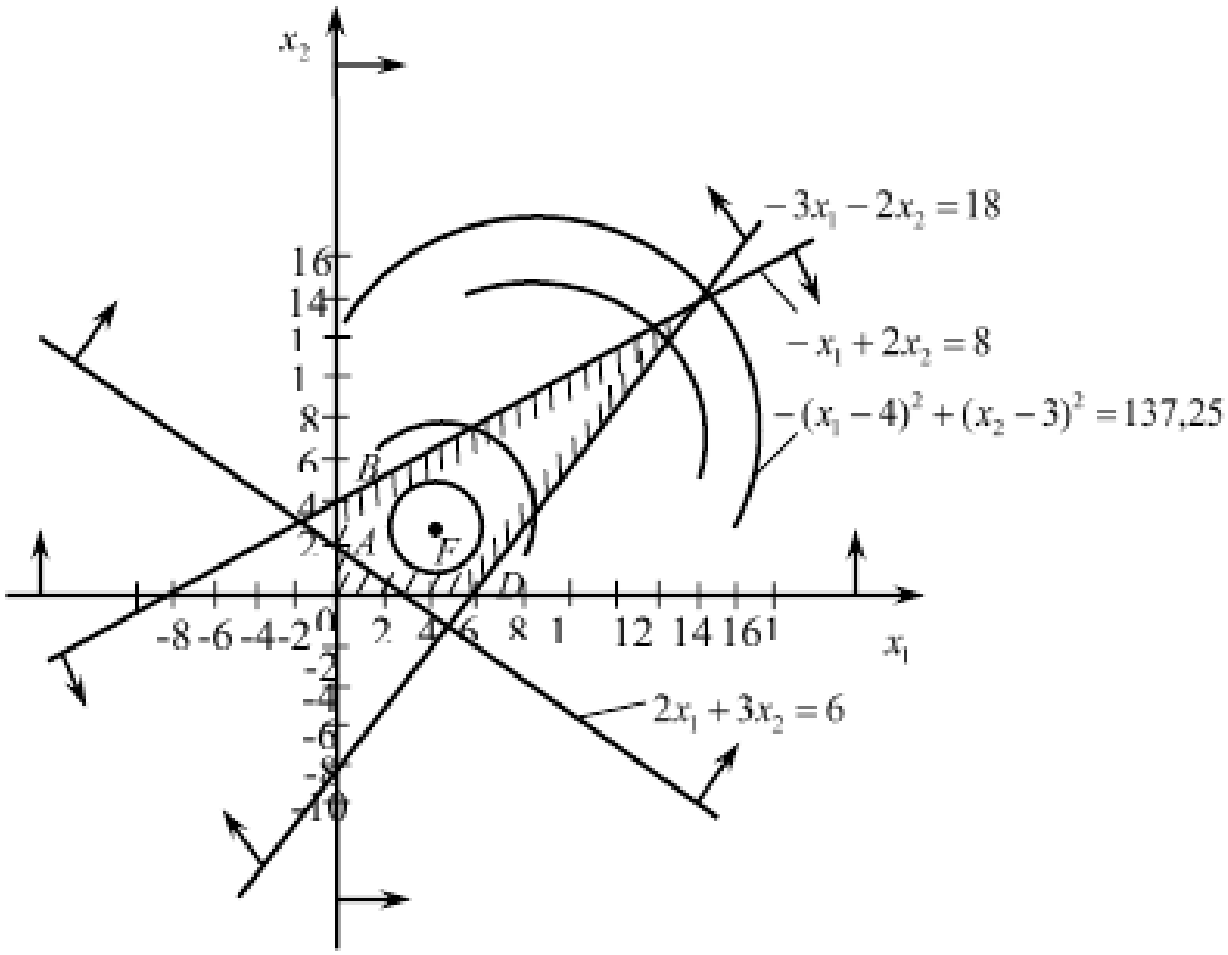

Рисунок 3.1 – Область допустимих рішень

Згідно графічного методу розв'язання поставлена задача має наступні рішення Fmax = 137, 25 при значеннях координат: $x_1 = 13$, $x_2 = 1,5$.

Fmin = 0 при значеннях координат: $x_1 = 4$, $x_2 = 3$.

Розв'яжемо це завдання генетичним алгоритмом, блок схема якого надана в п. 2.3.

Формування початкової популяції.

1. Генеруємо індивідууми популяції з набором хромосом, що складається з двох генів. Значення цих генів задаємо генератором випадкових

чисел. По-суті вони характеризують координати точки, що входить в область допустимих рішень, що задається нерівностями (3. 1).

Генерування випадкових точок з координатами (x1,x2) типу double :

```
double random(double min, double max){
    return (double)(rand())/RAND_MAX*(max - min) + min;
}
```

В результаті одержимо початкову популяцію індивідуумів з наборами хромосом кожна з яких має гени

Рисунок 3.2 – Хромосома з набором генів

По суті, кожен індивідуум буде мати хромосому, як вектор дійсних чисел, кожен компонент якого являє собою змінну завдання. Бінарне представлення підходить скоріше для дискретних змінних і для багатовимірних задач неефективно через неминучих перетворень між бінарною і речової формами кодування.

2. Проводимо відбір для створення нового покоління.

В цьому відборі спрацьовує селекція. Якщо набір набір генів хромосом індивідуума дозволяє виконати всі нерівності системи (3.2) то він буде відібраним для схрещування, якщо ні то такий індивідуум «помирає».

```
//Відбір
void groupSelection(vector<double>& x, vector<double>& y, vector<double>& res,
        vector<double>& x1, vector<double>& y1, vector<double>& res1) {

    if (minMaz == 1) {
            for (short i = 0; i < 600; i++) {

                int indexMax = maxElementIndex(res);

                x1.push_back(x[indexMax]);
                y1.push_back(y[indexMax]);
```

```
                res1.push_back(res[indexMax]);

                x[indexMax] = -1;
                y[indexMax] = -1;
                res[indexMax] = -1;

        }

        x.clear();
        y.clear();
        res.clear();
}
if (minMaz == 0) {
        for (short i = 0; i < 600; i++) {

                int indexMax = minElementIndex(res);

                x1.push_back(x[indexMax]);
                y1.push_back(y[indexMax]);
                res1.push_back(res[indexMax]);

                x[indexMax] = -1;
                y[indexMax] = -1;
                res[indexMax] = -1;

        }

        x.clear();
        y.clear();
        res.clear();
```

```
        }
};
```

3. Проведемо схрещування

Схрещування відбувається за схемою, що описана в п. 3.1. Особи що пройшли відбір потрапляють на кросинговер, після якого зявляється нове покоління, що знову потрапляє на відбір і т. д.

```
//Схрещування
void reproduction(vector<double>& x, vector<double>& y, vector<double>& res,
        vector<double>& x1, vector<double>& y1, vector<double>& res1) {
        for (short i = 0; i < res1.size() / 2; i++) {

                x.push_back(x1.back());
                x1.pop_back();
                y.push_back(y1[i]);
                y1[i] = 0;
        }

        for (short i = 0; i < res1.size() / 2; i++) {

                y.push_back(y1.back());
                y1.pop_back();
                x.push_back(x1[i]);
                x1[i] = 0;
        }

        for (short i = 0; i < res1.size(); i++)
                res.push_back((pow((x[i] - 4), 2) + pow((y[i] - 3), 2)));



        if (x.size() != y.size())
```

```
            cout << "Eror" << x.size() << "x-y" << y.size();


        x1.clear();
        y1.clear();
        res1.clear();
};
```

4. Проведемо мутацію

Мутацію ми проводимо з хромосомами тих індивідуумів, що не пройшли для схрещування. Основна мета, щоб елітні особи не повели всю популяцію до локального екстремуму. Потрібно покрити всю область допустимих рішень для розв’язання.

```
//Мутація
void mutation(vector<double>& x, vector<double>& y, vector<double>& res) {
        for (int i = 0; i < (rand() % res.size()); i++) {

            double bufX = 0, bufY = 0;
            bool flag1 = 0;
            bool flag2 = 0;
            bool flag3 = 0;

            do
            {
                flag1 = 0;
                flag2 = 0;
                flag3 = 0;

            bufX = round((double)(rand()) / RAND_MAX * 50 * 10) / 10;
            bufY = round((double)(rand()) / RAND_MAX * 50 * 10) / 10;

                if ((2 * bufX + 3 * bufY) >= 6)
```

```
                flag2 = 1;

            if ((3 * bufX - 2 * bufY) <= 18)
                flag3 = 1;

            if (-1 * bufX + 2 * bufY <= 8)
                flag1 = 1;

        } while (!flag1 || !flag2 || !flag3);

        x[i] = bufX;
        y[i] = bufY;

        res[i] = pow((bufX - 4), 2) + pow((bufY - 3), 2);
    }

};
```

5. Перевіряємо умови закінчення алгоритму

Якщо задати точність обчислення, то її можна досягти або за деякий встановлений час, або щляхом порівняння результатів двох поколінь. Якщо результати розрахунків будуть відрізнятися на менше ніж задана точність, дію генетичного алгоритму призупиняємо ьа виводимо результат.

```
//Перевірка умови завершення алгоритму
bool checkBestResult(vector<double>& x, vector<double>& y, vector<double>&
res) {
    if (minMaz == 1) {
        int indexMax = maxElementIndex(res);

        if (res[indexMax] == 137.25) {
```

```
                cout << "Для максимального значення ф-ї: " << endl
                    << "X - " << x[indexMax] << endl
                    << "Y - " << y[indexMax] << endl
                    << "Максимальне значення становить - " <<
res[indexMax];

                return 1;
            }
            else return 0;
        }
        if (minMaz == 0) {
            int indexMax = minElementIndex(res);

            if (res[indexMax] == 0) {

                cout << "Для мінімального значення ф-ї: " << endl
                    << "X - " << x[indexMax] << endl
                    << "Y - " << y[indexMax] << endl
                    << "Мінімальне значення становить - " <<
res[indexMax];

                return 1;
            }
            else return 0;
        }
};
```

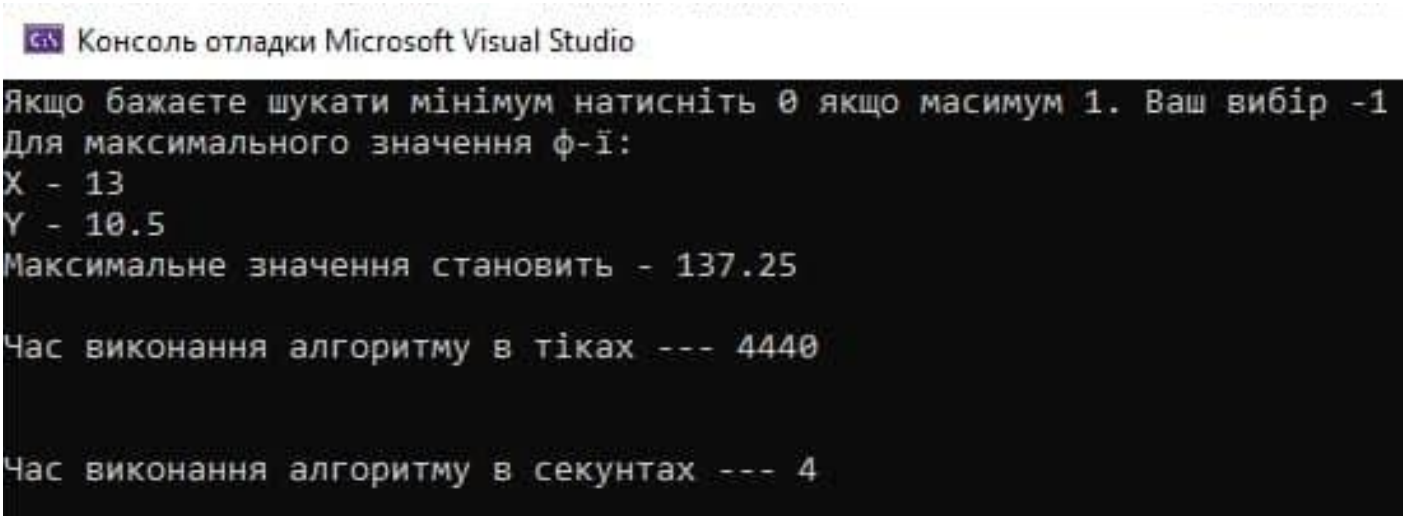

Рисунок 3. 3 – Результати обчислення max для функції цілі

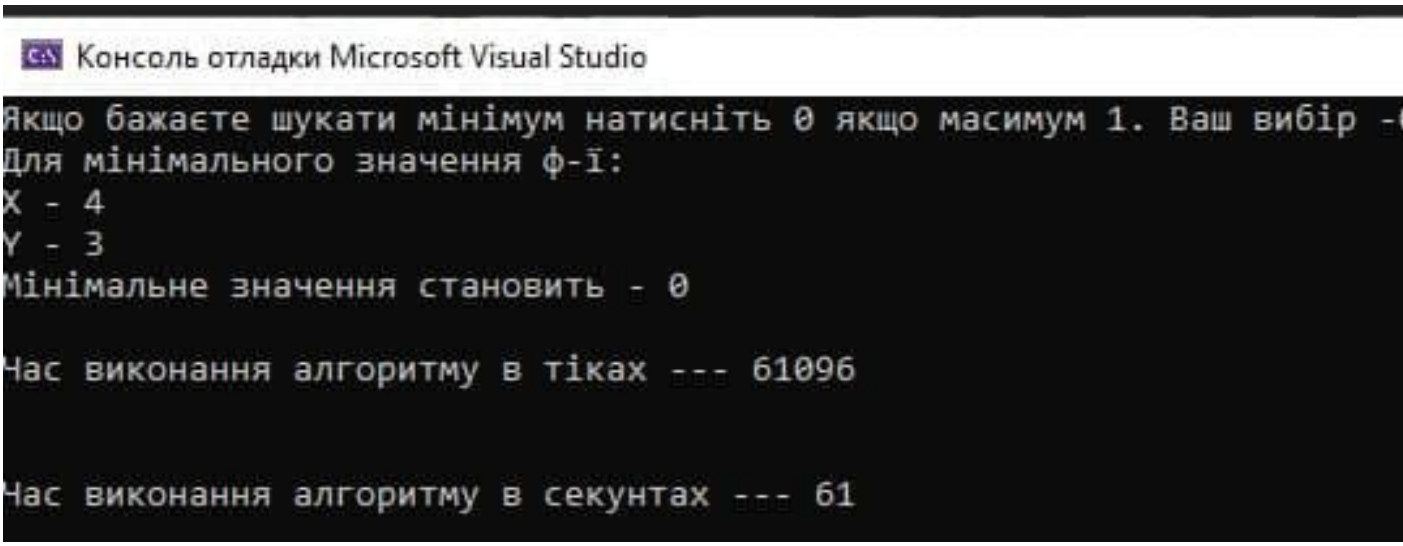

Рисунок 3. 4 – Результати обчислення min для функції цілі

Привертає увагу цікавий момент в одержаних результатах.

По-перше, генетичний алгоритм досяг точного результату як для максимального значення функції цілі, так і для мінімального її значення.

По-друге, час виконання алгоритму для пошуку мінімального значення набагато більший ніж для максимума. Якщо повернутися ло постановки задачі та її рішення графічно, то цікаво, що точка максимума лежить на границі, а точка мінімума знаходиться всередині області допустимих рішень.

То зробимо висновок, що якщо точка в задачах нелінійної оптимізації не лежить на границі області пошуку, то для її знаходження генетичному алгоритму потребується час набагато більший, аніж точка оптимуму була б граничною.

І цей час достатньо значний (t = 61 с).

ВИСНОВКИ

В магістерській роботі проведені дослідження по застосуванні генетичних алгоритмів до розв’язання задач нелінійного програмування.

Розроблено інформаційне та програмне забезпечення, яке протестоване на конкретних прикладах застосування.

За результатами досліджень можна зробити наступні висновки.

1. Генетичні алгоритми застосовні до розв’язання нелінійних завдань оптимізації та приводять до наближених рішень з достатньою точністю.
2. Проведено тестування на виконанні конкретних прикладів.
3. Показано залежність результатів від кількості осіб в популяції генетичного алгоритму.

СПИСОК ЛІТЕРАТУРИ

1. І.В. Калініна, О.І. Лісовиченко Використання генетичних алгоритмів в задачах оптимізації// Міжвід. науково-технічний збірник «Адаптивні системи автоматичного управління», 2015, № 1(26).-с. 48-61.
2. David G. Luenberger, Yinyu Ye Linear and Nonlinear Programming. - Springer, 2015. - 546 p.
3. Н. І. Бойко, В. Ю. Михайлишин Ефективність застосування генетичних алгоритмів для пошуку оптимізованих рішень// Інформаційні системи та мережі, 2016, №854. – с. 249-257.
4. Buontempo F. Genetic Algorithms and Machine Learning for Programmers. – Pragmatic Bookshelf, 2019. – 223 p.
5. Paul E. Black. Dictionary of Algorithms and Data Structures [Електронний ресурс] – Режим доступу до ресурсу: https://xlinux.nist.gov/dads/
6. Мелешко Є.В., Якименко М.С., Поліщук Л.І. Алгоритми та структури даних: Навчальний посібник для студентів технічних спеціальностей денної та заочної форми навчання. – Кропивницький: Видавець – Лисенко В.Ф., 2019. – 156 с.
7. Бірінцев М. О. , Шаповалов С. П. Адаптація генетичного алгоритму для розв'язання проблеми розподілу робіт// МА:: ІМА – 2019. Матеріали та програма науково-технічної конференції СумДУ. 2019. - Суми. С. 36-37
8. Кончатний В. В, Шаповалов С. П. Адаптація генетичного алгоритму для вирішення задачі комівояжера// Матеріали X студентської конференції «Перший крок у науку». – 2019, Сумду.- с. 101.
9. Тененев В.А., Шаура А.С. Решение задач нелинейного программирования общего вида генетическим алгоритмом // Интеллектуальные системы в производстве. 2019. Т. 18. № 4. С. 137–142.

ДОДАТОК

```
#include <iostream>
#include <Windows.h>
#include <vector>
#include <ctime>
#include <cmath>
#include <algorithm>

using namespace std;

//Прототипи фукцій
int maxElementIndex(vector<double> mas);
int minElementIndex(vector<double> mas);
void groupSelection(vector<double>& x, vector<double>& y, vector<double>& res,
	vector<double>& x1, vector<double>& y1, vector<double>& res1);
void reproduction(vector<double>& x, vector<double>& y, vector<double>& res,
	vector<double>& x1, vector<double>& y1, vector<double>& res1);
void mutation(vector<double>& x, vector<double>& y, vector<double>& res);
bool checkBestResult(vector<double>& x, vector<double>& y, vector<double>&
res);

bool minMaz;

int main() {

	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	unsigned int start_time = clock();
```

```
srand(time(0));

vector<double> x;
vector<double> y;
vector<double> res;
vector<double> x1;
vector<double> y1;
vector<double> res1;

short count = 0;
bool wyh = 1;

cout << "Якщо бажаєте шукати мінімум натисніть 0 якщо масимум 1. Ваш вибір -";
cin >> minMaz;

for (int i = 0; i < 1000; i++) {

	double bufX = 0, bufY = 0;
	bool flag1 = 0;
	bool flag2 = 0;
	bool flag3 = 0;

	do
	{
		flag1 = 0;
		flag2 = 0;
		flag3 = 0;
```

```
                bufX = round((double)(rand()) / RAND_MAX * 50 * 10) /
10;
                bufY = round((double)(rand()) / RAND_MAX * 50 * 10) /
10;


                if ((2 * bufX + 3 * bufY) >= 6)
                        flag2 = 1;

                if ((3 * bufX - 2 * bufY) <= 18)
                        flag3 = 1;

                if (-1 * bufX + 2 * bufY <= 8)
                        flag1 = 1;

        } while (!flag1 || !flag2 || !flag3);

        x.push_back(bufX);
        y.push_back(bufY);

        res.push_back((pow((bufX-4),2 )+ pow((bufY - 3), 2)));
}

do {

        groupSelection(x, y, res, x1, y1, res1);

        reproduction(x, y, res, x1, y1, res1);

        if (count == 1) {
```

```
                count = 0;
                mutation(x, y, res);
            }
            else count++;
            mutation(x, y, res);
            if (checkBestResult(x, y, res))
                wyh = 0;

        } while (wyh);

        unsigned int end_time = clock();
        unsigned int search_time = end_time - start_time;

        cout << endl << endl << "Час виконання алгоритму в тіках --- "
            << search_time << endl;
        cout << endl << endl << "Час виконання алгоритму в секунтах --- "
            << search_time / CLOCKS_PER_SEC << endl;
        return 0;
}

//Пошук індексу максимального елемента
int maxElementIndex(vector<double> mas) {
        int index = 0,
            max = mas[0];

        for (int i = 1; i < mas.size(); i++)
            if (max < mas[i])
            {
                max = mas[i];
                index = i;
```

```
		}

	return index;
};

//Пошук індексу мінімального елемента
int minElementIndex(vector<double> mas) {
	int index = 0,
		min = mas[0];

	for (int i = 1; i < mas.size(); i++)
		if (min > mas[i])
		{
			min = mas[i];
			index = i;

		}

	return index;
};

//Відбір
void groupSelection(vector<double>& x, vector<double>& y, vector<double>& res,
	vector<double>& x1, vector<double>& y1, vector<double>& res1) {

	if (minMaz == 1) {
		for (short i = 0; i < 600; i++) {

			int indexMax = maxElementIndex(res);
```

```
                x1.push_back(x[indexMax]);
                y1.push_back(y[indexMax]);
                res1.push_back(res[indexMax]);

                x[indexMax] = -1;
                y[indexMax] = -1;
                res[indexMax] = -1;

        }

        x.clear();
        y.clear();
        res.clear();
}
if (minMaz == 0) {
        for (short i = 0; i < 600; i++) {

                int indexMax = minElementIndex(res);

                x1.push_back(x[indexMax]);
                y1.push_back(y[indexMax]);
                res1.push_back(res[indexMax]);

                x[indexMax] = -1;
                y[indexMax] = -1;
                res[indexMax] = -1;

        }
```

```
            x.clear();
            y.clear();
            res.clear();
        }
};

//Схрещування
void reproduction(vector<double>& x, vector<double>& y, vector<double>& res,
        vector<double>& x1, vector<double>& y1, vector<double>& res1) {
        for (short i = 0; i < res1.size() / 2; i++) {

            x.push_back(x1.back());
            x1.pop_back();
            y.push_back(y1[i]);
            y1[i] = 0;
        }

        for (short i = 0; i < res1.size() / 2; i++) {

            y.push_back(y1.back());
            y1.pop_back();
            x.push_back(x1[i]);
            x1[i] = 0;
        }

        for (short i = 0; i < res1.size(); i++)
            res.push_back((pow((x[i] - 4), 2) + pow((y[i] - 3), 2)));


        if (x.size() != y.size())
```

```
                cout << "Eror" << x.size() << "x-y" << y.size();

        x1.clear();
        y1.clear();
        res1.clear();
};

//Мутація
void mutation(vector<double>& x, vector<double>& y, vector<double>& res) {
        for (int i = 0; i < (rand() % res.size()); i++) {

                double bufX = 0, bufY = 0;
                bool flag1 = 0;
                bool flag2 = 0;
                bool flag3 = 0;

                do
                {
                        flag1 = 0;
                        flag2 = 0;
                        flag3 = 0;

                        bufX = round((double)(rand()) / RAND_MAX * 50 * 10) /
10;
                        bufY = round((double)(rand()) / RAND_MAX * 50 * 10) /
10;

                        if ((2 * bufX + 3 * bufY) >= 6)
                                flag2 = 1;
```

```
                    if ((3 * bufX - 2 * bufY) <= 18)
                        flag3 = 1;

                    if (-1 * bufX + 2 * bufY <= 8)
                        flag1 = 1;

                } while (!flag1 || !flag2 || !flag3);

                x[i] = bufX;
                y[i] = bufY;

                res[i] = pow((bufX - 4), 2) + pow((bufY - 3), 2);
        }

};

//Перевірка умови завершення алгоритму
bool checkBestResult(vector<double>& x, vector<double>& y, vector<double>&
res) {
        if (minMaz == 1) {
                int indexMax = maxElementIndex(res);

                if (res[indexMax] == 137.25) {

                    cout << "Для максимального значення ф-ї: " << endl
                        << "X - " << x[indexMax] << endl
                        << "Y - " << y[indexMax] << endl
                        << "Максимальне значення становить - " <<
res[indexMax];
```

```
                return 1;
            }
            else return 0;
        }
        if (minMaz == 0) {
            int indexMax = minElementIndex(res);

            if (res[indexMax] == 0) {

                cout << "Для мінімального значення ф-ї: " << endl
                    << "X - " << x[indexMax] << endl
                    << "Y - " << y[indexMax] << endl
                    << "Мінімальне значення становить - " <<
res[indexMax];

                return 1;
            }
            else return 0;
        }
};
```