Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування вищого навчального закладу)

Навчально-науковий інститут інформаційних технологій та робототехніки
(повна назва інституту)

Кафедра комп'ютерних та інформаційних технологій і систем
(повна назва кафедри)

Пояснювальна записка
до дипломного проекту (роботи)
магістра
(рівень вищої освіти)
на тему

Застосування квантового генетичного алгоритму у задачах синтезу програмних кодів

Виконала: студентка 2 курсу, групи 601-ТН
спеціальності
122 Комп'ютерні науки
(шифр і назва спеціальності)
Білоус Я. І.
(прізвище та ініціали)
Керівник Скакаліна О. В.
(прізвище та ініціали)
Рецензент_____
(прізвище та ініціали)

Полтава – 2021 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ « ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

спеціальність 122 «Комп'ютерні науки»

на тему

«Застосування квантового генетичного алгоритму у задачах синтезу програмних кодів»

Студентки групи 601-ТН Білоус Яни Ігорівни

Керівник роботи
кандидат технічних наук,
доцент Скакаліна О. В.

Завідувач кафедри
кандидат технічних наук,
доцент Головко Г.В.

Полтава – 2021

РЕФЕРАТ

Кваліфікаційна робота магістра: 117 с., 69 малюнки, 5 додатки, 49 джерел.

Об’єкт дослідження: квантовий генетичний алгоритм у задачах оптимізації побудови оптимального шляху.

Мета роботи: розробка алгоритму на базі генетичного алгоритму та квантових обчислень для задачі оптимізації побудови оптимального шляху.

Методи: проектування та створення квантового генетичного алгоритму для задачі пошуку оптимального шляху, моделювання його роботи на симуляторах різних середовищ розробки.

Ключові слова:генетичний алгоритм, оптимізація, задачі пошуку рішень, популяція, кросинговер, мутація, відбір, квантові обчислення, інтерференція, кубіт, факторинг, квантовий алгоритм, симуляція.

ABSTRACT

Explanatory note contains: 117 pages, 69 pictures, 5 additions, 49 references.

Object of research: quantum genetic algorithm in the problems of optimizing the construction of the optimal path.

The purpose of the qualification work: development of an algorithm based on a genetic algorithm and quantum calculations for the task of optimizing the construction of the optimal path.

Methods: design and creation of a quantum genetic algorithm for the task of finding the optimal path, modeling its operation on simulators of different development environments.

Keywords:genetic algorithm, optimization, solution problems, population, crossover, mutation, selection, quantum computing, interference, qubit, factoring, quantum algorithm, simulation.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ГА (GA) – генетичний алгоритм («geneticalgorithm»);

NP – клас складності задач з недетерміновано-поліноміальними алгоритмами («Nondeterministic Polynomial»);

Індивід – (генетичний код) набір хромосом;

ІТ – інформаційні технології;

Кросовер – це впорядкований процес обміну частинок між двома хромосомами;

Мутація – це випадкова заміна одного або декількох частинок хромосоми;

КГА (QGA) – квантовий генетичний алгоритм («quantumgeneticalgorithm»);

QGA_3 – квантовий генетичний алгоритм з кутрітрим представленням змінної;

Евклідовий простір – скінченновимірний дійсний векторний простір зі скалярним добутком;

Ротація – тутповорот квантових вентилів у евклідовому просторі;

Факторинг – розкладання числа на просто множники;

Квантова механіка – наука, що описує поведінку квантових частинок;

Інтерференція – частинний випадок інтерференції для видимої області електромагнітного спектра (тут мається на увазі інтерференція квантів – світла);

Matlab – пакет програм для числового аналізу та тестування алгоритмів;

PyCharm – середовище розробки для мови *Python.*

ВСТУП

Підраховано, що кожні два роки протягом останніх 50 років комп'ютери ставали вдвічі швидшими, а їх компоненти – вдвічі меншими. Якщо прогрес буде продовжуватися з такою швидкістю, майбутні комп'ютерні схеми будуть засновані на нанотехнологіях, і поведінку таких схем доведеться викладати в термінах квантової механіки, а не в термінах класичної фізики, оскільки в атомному масштабі матерія підкоряється законам квантової механіки. Оскільки технологія рухається до нано- і квантового рівня, виникатиме потреба в обчислювальних парадигмах, які можуть отримати вигоду від їх близькості до квантового обладнання. Такі обчислювальні парадигми потребуватимуть менше перекладу на квантову «мову машини», ніж їх класичні аналоги, і, отже, матимуть переваги щодо ефективності. Генетичні алгоритми, у поєднанні з квантовою механікою, описані тут, все ще на один крок віддалені від справжніх квантових генетичних алгоритмів (де таким поняттям, як хромосоми, буде надано квантовий або молекулярний, а не символічний опис), проте вже зараз їх можливості виконують корисну функцію, допомагаючи нинішнім дослідникам визначити потенційні тенденції у еволюційних обчисленнях. Наразі неясно, як справжні алгоритми квантових обчислень будуть пов'язані з квантовим обладнанням, наприклад, квантовими логічними вентилями. Можливо, що квантові обчислення забезпечать переваги на різних портативних квантових апаратних платформах, за умови квантових компіляторів.

Незважаючи на те, що ця область не до кінця досліджена, вона вже стала інноваційною та актуальною для великих компаній. У найближчі два роки очікується істотне нарощування інвестицій у квантові обчислення, прогнозують аналітики InternationalDataCorporation (IDC).Частка компаній, які планують виділити більше 17% своїх IT-бюджетів на квантові технології, як очікується, збільшиться з 7% в 2021 році до 19% в 2023-му. Це зростання

пояснюється тим, що глобальні компанії інвестують в хмарні платформи, які забезпечують доступ до програмних і апаратних розробок в сфері квантових обчислень, наймають фахівців за такими технологіями, навчають розробників і співпрацюють з постачальниками квантових обчислень для розробки нових бізнес-рішень. Тому, квантові обчислення – це не майбутнє, це вже сьогодні.

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД МАТЕРІАЛІВ ТА ПРОГРАМНИХ ЗАСОБІВ НА БАЗІ КВАНТОВИХ ГЕНЕТИЧНИХ АЛГОРИТМІВ

1.1 Генетичні алгоритми та проблеми які вони вирішують

Генетичні алгоритми – це сімейство обчислювальних моделей, натхнених еволюцією. Ці алгоритми працюють за принципом кодування потенційного рішення конкретної проблеми на основі простої хромосомної структури даних, шляхом рекомбінації цих структур за умови збереження критичної інформації. Генетичні алгоритми часто розглядаються як оптимізатор функцій, хоча коло проблем їх застосування набагато ширше. Реалізація генетичного алгоритму починається із формування популяції (зазвичай випадкових) хромосом. Потім оцінюють ці структури та виділяють їх репродуктивні можливості, тобто ті хромосоми, які є кращим рішенням цільової проблеми. Таким чином, з покоління в покоління, корисні ознаки поширюються по всій популяції, а погані поступово зникають. Завдяки схрещуванню найбільш пристосованих особин, успадковуються більш перспективні ділянки простору пошуку. Зрештою, популяція буде сходитися до оптимального рішення задачі. ГА знаходить приблизні оптимальні рішення за досить короткий час, що є очевидною перевагою даного методу. Таким чином генетичний алгоритм – це евристичний алгоритм пошуку, що використовується для розв’язування задач оптимізації та моделювання шляхом випадкового підбору, комбінування і варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію. «Генетичними» алгоритми стали називатися пізніше, аж у 1975 році Холланд називав їх репродуктивними планами *(reproductiveplan)* і розглядав насамперед як алгоритми адаптації. Але зміщення акцентів у трактуванні поняття «адаптація», про яке автор говорить у передмові 1992 року, дуже точно передає стан неоднозначності, намагаючись, з одного боку, дати

досить загальне і несуперечливе поняття адаптації, а з іншого боку, розмежувати поняття адаптації і оптимізації, адаптації та еволюції, адаптації та навчання.

Генетичні алгоритми – це алгоритми пошуку, які ґрунтуються на концепціях природного відбору. У природі особи, які мають кращі риси виживання, відповідно існують довший період часу, оскільки мають кращі шанси народити потомство з подібним генетичним матеріалом. Протягом деякого часу вся популяція буде складатися з великої кількості генів вищих особин і меншої від слабкіших. Помилка старших теоретиків, наприклад Жана Батиста Ламарка, полягала в тому, що оточення впливало на індивідуальну особистість. Тобто навколишнє середовище змусить індивіда адаптуватися до нього. Молекулярне пояснення еволюції доводить, що це біологічно неможливо. Вид не пристосовується до навколишнього середовища, скоріше, виживають тільки найсильніші. Так само працює природний відбір у генетичних алгоритмах. Генетичний алгоритм відрізняється від інших методів пошуку тим, що здійснює пошук серед сукупності точок і працює з набором параметрів, а не самими значеннями параметрів. Він також використовує об'єктивну інформації про функцію без будь-якої інформації про градієнт. Тоді як традиційні методи використовують градієнтну інформацію. Через ці особливості алгоритму, вони застосовуються до різних функцій оптимізації, оцінки параметрів та програм машинного навчання [1].

На даному етапі розвитку немає певної стратегії побудови рішення, існує величезна кількість реалізованих алгоритмів, не схожих один на одного. Але все ж існує традиційна схема роботи даних алгоритмів (рисунок 1). Критеріїв зупинки пошуку рішення може бути декілька: часові рамки, кількість створених популяцій та зменшення покращення фітнес-функції у порівнянні з попередніми ітераціями.

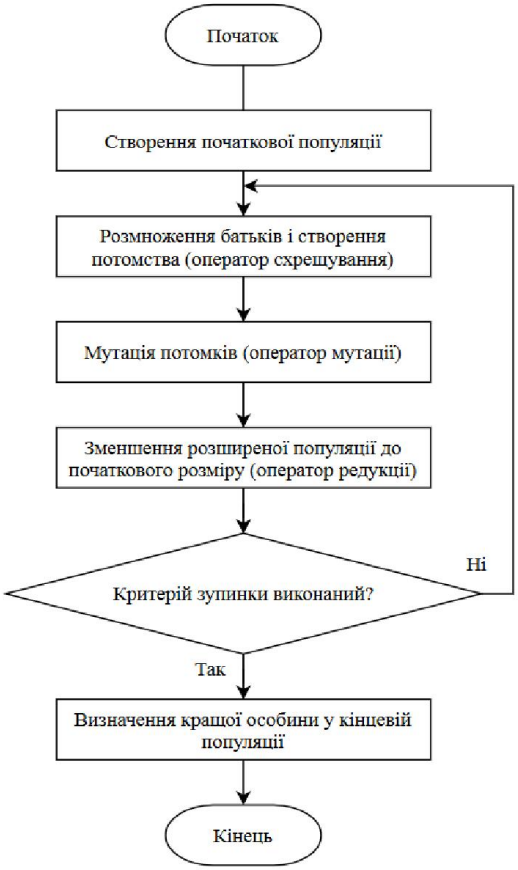


Рисунок 1.1 – Традиційна схема генетичних алгоритмів

Потенціал генетичних алгоритмів складно переоцінити. Коли знайти рішення майже неможливо звичайними методами, очевидним виходом із ситуації є саме вони. У різних формах генетичні алгоритми застосовуються до наукових і технічних проблем. Їх можна використовувати при створенні обчислювальних структур, таких як кінцеві автомати та мережі сортування. Нерідко вони використовуються при проектуванні нейронних мереж і управлінні роботами, при моделюванні розвитку процесів у різних предметних областях, в ігрових стратегіях, складанні розкладу, логістиці, задачах розкрою, у розробках штучного життя і в багатьох інших областях [2]. Але все ж найпопулярніше застосування генетичних алгоритмів – це оптимізація багатопараметричних функцій.

Так, наприклад, ведуться експерименти по створенню працюючих в команді роботів з метою розмінування території. В основі лежить багатошарова нейронна мережа, яка безпосередньо відповідає за управління

роботами і, крім того, передає і отримує сигнали іншим членам команди. Іншими словами, крім параметричної оптимізації нейромодель виконує неявне створення мови комунікації між роботами. Результати показують явну перевагу команди роботів саперів, які обмінюються інформацією між собою, перед роботами, що діють без будь-якого зв'язку. Як наслідок, дана технологія в перспективі може врятувати безліч людських життів, а професія сапера паче не буде актуальною.

В останні десять років ігрова індустрія розвивається швидкими темпами, чималу роль в цьому відіграє застосування генетичних алгоритмів. Багато процесів в іграх виконуються за правилами живої природи, що робить ігровий процес більш реалістичним і непередбачуваним. Як було сказано вище, генетичні алгоритми застосовуються при проектуванні нейронних мереж, які швидкими темпами впроваджуються в різні сфери життєдіяльності суспільства, зокрема, ігрову [3]. Так, в березні 2016 року програму AlphaGo, заснована на нейронних мережах, обіграла найсильнішого гравця в світі в грі «Го», що говорить про те, що вже зараз програма, заснована на генетичних алгоритмах, перевершує людину у логіці. Дані технології в найближчому майбутньому можуть подарувати нам ігри повного занурення, що ще недавно здавалося фантастикою.

Генетичні алгоритми також є частиною бізнесу, з їх допомогою ведуться підрахунки ризиків і загроз для підприємств, що згодом оберігає компанії від непередбачених витрат, допомагає скоротити збитки і збільшити прибуток за рахунок вибору оптимальних стратегій.

Звичайно що у генетичних алгоритмів є ряд недоліків. Генетичні алгоритми навчання складні для розуміння і програмної реалізації. Можливі випадки, де проблематично використовувати генетичний алгоритм, наприклад, коли необхідно знайти точний глобальний оптимум, коли великий об’єм часу виконання функції оцінки, коли необхідно знайти всі рішення задачі, а не одне з них, коли конфігурація є не простою і, нарешті, коли відповідь має скоріше статичний характер. Але все ж можливості

генетичних алгоритмів відкривають нам надзвичайно широкі горизонти рішень найрізноманітніших задач.

Генетичні алгоритми – це геніальна ідея, основа для них взята у самої природи, про яку багато досі не відомо, а значить, існує ще безліч невирішених завдань, у вирішенні яких належить розібратися. У використанні генетичних алгоритмів головне пам'ятати, що їх використання корисно лише тоді, коли для даного завдання немає відповідного спеціального алгоритму рішення [4]. Основні концепції та принцип роботи ГА

Найважливіші кроки у роботі ГА – це генерування сукупності рішень, пошук цільової функції та функції придатності та застосування генетичних операторів.

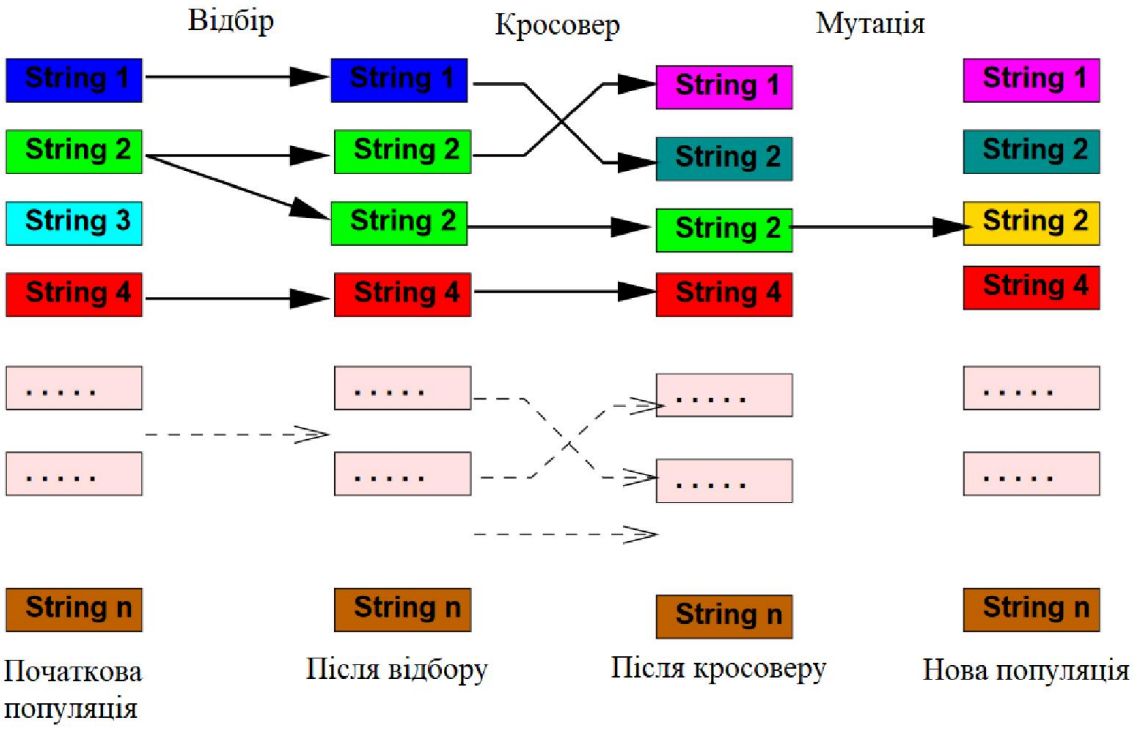


Рисунок 1.2 – Основні операції ГА

Важливим моментом є кодування змінних, що описують проблему. Найпоширеніший метод кодування – перетворення змінних у двійковий

рядок або вектор. ГА працюють найкраще, коли вектори рішення є двійковими. Якщо проблема має більше однієї змінної, кодування з кількома змінними створюється шляхом конкатенації (склеювання) стількох кодів окремих змінних, скільки і змінних у задачі. Генетичний алгоритм обробляє ряд рішень одночасно. Отже, на першому кроці популяція, що має P індивідуумів, генерується псевдо випадковими генераторами, являється особинами, які представляють можливе рішення. Це представлення вектора рішення у просторі розв'язку і називається початковим рішенням. Це забезпечує надійність і неупередженість пошуку, оскільки він починається з широкого діапазону точок у просторі рішення. На наступному кроці окремі члени сукупності оцінюються, щоб знайти цільову функцію. На цьому кроці метод зовнішньої штрафної функції використовується для перетворення обмеженої проблеми оптимізації на необмежену. На третьому етапі цільова функція перетворюється на фітнес-функцію, яка обчислює значення придатності для кожного члена сукупності. Далі йде застосування операторів GA.

Існує кілька підходів до вибору батьківської пари, так званого оператору вибору батьків. Найбільш поширеними операторами вибору батьків є наступні. ***Панміксія*** – найпростіший оператор відбору. Відповідно до нього кожному члену популяції зіставляється випадкове ціле число на відрізку *[1; n]*, де *n* – кількість особин у популяції. Будемо розглядати ці числа як номери особин, які візьмуть участь у схрещуванні. При такому виборі якісь із членів популяції не будуть брати участь в процесі розмноження, так як утворюють пару самі з собою. Якісь члени популяції візьмуть участь у процесі відтворення неодноразово з різними особинами популяції. Незважаючи на простоту, такий підхід універсальний для вирішення різних класів задач. Однак він досить критичний до чисельності популяції, оскільки ефективність алгоритму, що реалізує такий підхід, знижується з ростом чисельності популяції. ***Інбридинг*** є такий метод, коли один з батьків вибирається випадковим чином, а другий є членом популяції,

найближчим до першого. Тут «найближчий» може розумітися, наприклад, в сенсі мінімального відстані Хеммінга (для бінарних рядків) або евклідової відстані між двома векторами. Відстань Хеммінга дорівнює числу локусів (розрядів) що різняться у бінарному рядку. При ***аутбридингу*** також використовують поняття схожості особин. Однак тепер батьківські пари формують з максимально далеких особин. Останні два способи по різному впливають на поведінку генетичного алгоритму. Так, інбридинг можна охарактеризувати властивістю концентрації пошуку в локальних вузлах, що фактично призводить до розбиття популяції на окремі локальні групи на ділянках екстремуму ландшафту. Аутбридинг ж спрямований на попередження збіжності алгоритму з вже знайденим рішенням, змушуючи алгоритм переглядати нові, недосліджені області.

Селекція полягає в тому, що батьками можуть стати тільки ті особини, значення пристосованості яких не менше порогової величини, наприклад, середнього значення пристосованості по популяції. Такий підхід забезпечує більш швидку збіжність алгоритму. Однак через швидку збіжність селективний вибір батьківської пари не підходить тоді, коли ставиться завдання визначення декількох екстремумів, оскільки для таких завдань алгоритм, як правило, швидко сходиться до одного з рішень. Крім того, для деяких багатовимірних задач з складним ландшафтом цільової функції швидка збіжність може перетворитися в передчасну збіжність, не знайшовши квазіоптимальне рішення. Цей недолік може бути частково компенсований використанням відповідного механізму відбору, який «гальмував» би занадто швидку збіжність алгоритму. Порогова величина в селекції може бути обчислена різними способами. Тому в літературі по ГА виділяють різні варіації селекції. Найбільш відомі з них – це турнірний і пропорційний відбори (метод рулетки).

При турнірному відборі (*tournamentselection)* з популяції, що містить N особин, вибираються випадковим чином t особин, і найкраща з них записується в проміжний масив, ця операція повторюється *N* раз. Особи в

отриманому проміжному масиві потім використовуються для схрещування (також випадковим чином). Розмір групи рядків, що відбираються для турніру, часто дорівнює двом. Таким чином *t* називають чисельністю турніру. Перевагою даного способу є те, що він не вимагає додаткових обчислень. У методі рулетки *(roulette-wheelselection)* особини відбираються за допомогою *N* «запусків» рулетки, де *N* – розмір популяції. Колесо рулетки містить по одному сектору для кожного члена популяції. Розмір *i*-го сектора пропорційний ймовірності попадання в нову популяцію *P(i)*. Очікуване число копій *i*-ї хромосоми після оператора рулетки визначається за формулою $N_i = P(i)N$. При такому відборі члени популяції з більш високою пристосованістю мають більшу вірогідність частіше вибиратися, ніж особини з низькою пристосованістю. Інші способи відбору можна отримати на основі модифікації вищенаведених.

Кросинговер (бінарна рекомбінація або схрещування). Одноточковий кросинговер *(single-point crossover)* моделюється наступним чином: нехай є дві батьківські особини з хромосомами $X = \{x_i, i = [0, L]\}$та $Y = \{y, i \in [0, L]\}$. Випадковим чином визначається точка всередині хромосоми (точка розриву), в якій обидві хромосоми діляться на дві частини і обмінюються ними. Такий тип кросинговеру називається однокрапковим, так як батьківські хромосоми розділяються тільки в одній випадковій точці. Також застосовується дво- і *N*-точковий кросинговер.

У двоточковому кросинговері (і багатоточковому кросинговері взагалі) хромосоми розглядаються як цикли, які формуються з'єднавши решту лінійної хромосоми разом. Для заміни сегмента одного циклу сегментом іншого циклу необхідно обрати дві точки розрізу. Отже, двоточковий кросинговер вирішує ту ж саму задачу, що і одноточковий, але більш повно. Хромосома, розглянута як цикл, може містити більшу кількість стандартних блоків, так як вони можуть зробити «циклічне повернення» в кінці рядка. На даний момент багато дослідників погоджуються, що двоточковий кросинговер взагалі краще, ніж одноточковий.

Для багатоточкового кросинговеру *(multi-point crossover)* обираємо *m* точок розрізу. Точки розрізу вибираються випадково без повторень і упорядковано в порядку збільшення. При кросинговері відбувається обмін ділянками хромосом, обмеженими точками розрізу, і таким чином отримуються двоє нащадків. Ділянка особини з першим геном до першої точки розрізу в обміні не бере участь.

Однорідний кросинговер *(uniform crossover)* створює маску (схему) особини, в кожному локусі якої знаходиться потенційна точка кросинговеру. Маска кросинговеру має ту ж довжину, що і перехресні особини. Таким чином, гени маски представляють собою випадкові двійкові числа (0 або 1). Згідно з цими значенням, геном нащадку стає перша (якщо ген маски дорівнює нулю) або друга (якщо ген маски дорівнює одиниці) особина-батько. Оскільки у такому випадку рядок бітових значень довше, це гарантує, що в нащадках будуть чергуватися короткі строки особин-батьків.

Тріадний кросинговер *(triadic crossover)*. Цей різновид кросинговеру відрізняється від однорідного тим, що після відбору пари батьків з інших членів популяції випадковим чином вибирається особина, яка в подальшому використовується в якості маски. Далі 10% генів маски мутують. Потім гени першого батька порівнюються з генами маски: якщо гени однакові, то вони передаються першого нащадку, в іншому випадку на відповідні позиції хромосоми нащадка переходять гени другого з батьків. Генотип другого нащадка відрізняється від генотипу першого тим, що на тих позиціях, де у першого нащадка стоять гени першого батька, у другого нащадка стоять гени другого з батьків і навпаки.

Перетасовочний кросинговер *(shuffler crossover)*. В даному алгоритмі особини, відібрані для кросинговеру, випадковим чином обмінюються генами. Потім вибирають точку для одноточкового кросинговеру і проводять обмін частинами хромосом. Після схрещування створені нащадки знову тасуються. Таким чином, при кожному кросинговері створюються не тільки нові нащадки, а й модифікуються батьки (старі батьки видаляються), що

дозволяє скоротити число операцій в порівнянні з однорідним кросинговером.

Кросинговер зі зменшенням заміни *(crossover with reduced surrogate)*. Оператор зменшення заміни обмежує кросинговер, щоб завжди, коли це можливо, створювати нові особини. Це здійснюється за рахунок обмеження на вибір точки розрізу: точки розрізу повинні з'являтися тільки там, де гени розрізняються. Як було показано вище, кросинговер генерує нове рішення (у вигляді особини-нащадка) на основі двох наявних, комбінуючи їх частини. Тому число різних рішень, які можуть бути отримані кросинговером при використанні однієї і тієї ж пари готових рішень, обмежена. Відповідно, простір, яке ГА може покрити, використовуючи лише кросинговер, жорстко залежить від генофонду популяції. Чим різноманітніше генотипи рішень популяції, тим більше простір покриття. При виявленні локального оптимуму відповідний йому генотип буде прагнути зайняти всі позиції в популяції, і алгоритм може зійтися до помилкового оптимуму. Тому в генетичному алгоритмі важливу роль грають мутації. Існує кілька способів мутації генів.

Після процесу відтворення відбуваються ***мутації****(mutation)*. Даний оператор необхідний для «вибивання» популяції з локального екстремуму і перешкоджає передчасної збіжності. Це досягається за рахунок того, що змінюється випадково обраний ген в хромосомі. Так само як і кросинговер, мутації можуть проводитися не тільки по одній випадковій точці. Можна вибирати для зміни кілька точок в хромосомі, при чому їх число також може бути випадковим. Використовують і мутації зі зміною відразу деякої групи точок поспіль.

Оператори відбору особин в нову популяцію. Для створення нової популяції можна використовувати різні методи відбору особин.

Відбір урізанням *(truncation selection)*. При відборі «урізанням» використовують популяцію, що складається як з особин-батьків, так і особин нащадків, відсортовану по зростанню значень функції придатності особин. Число особин для схрещування вибирається відповідно до порогу $T \in [0;1]$.

Поріг визначає, яка частка особин, починаючи з найпершої (найпридатнішої), братиме участь у відборі. В принципі, поріг можна задати і числом більше 1, тоді він буде просто дорівнювати числу особин з поточної популяції, допущених до відбору. Серед особин, які потрапили «під поріг», випадковим чином вибирається одна і записується в нову популяцію. Процес повторюється *N* разів, поки розмір нової популяції не стане дорівнює розміру вихідної популяції. Нова популяція складається тільки з особин з високою придатністю, причому одна і та ж особина може зустрічатися кілька разів, а деякі особини, які мають придатність вище порогової, можуть не потрапити в нову популяцію. Через те, що в цій стратегії використовується відсортована популяція, час її роботи може бути великим для популяції великого розміру і залежатиме також від алгоритму сортування.

Елітарний добір *(elite selection)*. Створюється проміжна популяція, яка включає в себе як батьків, так і їх нащадків. Члени цієї популяції оцінюються, а потім з них вибираються *N* найкращих (придатних), які і увійдуть в наступне покоління. Найчастіше даний метод комбінують з іншими – вибирають в нову популяцію, наприклад, 10% «елітних» особин, а решта 90% – одним з традиційних методів селекції. Іноді ці 90% особин отримуються випадково, як при створенні початкової популяції перед запуском роботи ГА. Використання стратегії елітизму є корисним для ефективності ГА, так як не допускає втрату кращих рішень. Наприклад, якщо популяція зійшлася в локальному максимумі, а мутація вивела одну з рядків в область глобального, то при попередній стратегії досить імовірно, що ця особина в результаті схрещування буде втрачена, і рішення задачі не буде отримано. Якщо ж використовується елітизм, то отримане гарне рішення буде залишатися в популяції до тих пір, поки не буде знайдено ще краще.

Відбір витісненням *(exclusion selection)*. В даному відборі вибір особини в нову популяцію залежить не тільки від величини її придатності, а й від того, чи є вже в формованої популяції особина з аналогічним хромосомним набором. Відбір проводиться з числа батьків та їх нащадків. З

усіх особин з однаковою пристосованістю перевага спочатку віддається особинам з різними генотипами. Таким чином, досягаються дві мети: по-перше, не втрачаються кращі знайдені рішення, які мають різними хромосомними наборами, по-друге, в популяції постійно підтримується генетична різноманітність. Витіснення в даному випадку формує нову популяцію скоріше з віддалених особин, замість особин, що групуються близько поточного знайденого рішення. Даний метод найбільш придатний для багатоекстремальних завдань, при цьому крім визначення глобальних екстремумів з'являється можливість виділити і ті локальні максимуми, значення яких близькі до глобальних [5].

1.2 Огляд квантового генетичного алгоритму

GQA базується на концепції та принципах квантових обчислень, таких як кубіти та суперпозиція станів. Замість бінарного, числового або символічного представлення, прийнявши кубітову хромосому як представника популяції, GQA може представляти лінійну суперпозицію рішень через її ймовірнісне представлення. Як генетичні оператори, квантові використовуються для пошуку найкращого рішення. Швидка конвергенція та хороші можливості глобального пошуку характеризують продуктивність GQA.

Багато зусиль приклали вчені для просування ідеї квантового комп’ютера з початку 1990-х років, тому що ці комп’ютери були більш потужними, ніж класичні, з різних спеціалізованих проблем. Але якщо немає квантового алгоритму, який би вирішував практичні проблеми, апаратне забезпечення квантового комп’ютера може бути непотрібним. Його можна розглядати як комп’ютер без операційної системи. Хоча нові квантові алгоритми мали б значну користь, яка могла б вирішувати обчислювальні проблеми швидше, ніж класичні алгоритми, на сьогоднішній день відомо лише декілька квантових алгоритмів. Тим не менш, квантові обчислення

привертають серйозну увагу, оскільки їх перевагу було продемонстровано кількома квантовими алгоритмами, такими як алгоритм квантового факторингу Шора та алгоритм пошуку бази даних Гровера. Алгоритм Шора знаходить прості множники n-цифри число в поліноміальному часі, тоді як найбільш відомі класичні алгоритми факторингу вимагають часу $O(2^{n1/3\log(n)1/3})$. Алгоритм пошуку може знайти елемент у несортованому списку з n елементів за $O(\sqrt{n})$ кроків, тоді як класичні алгоритми вимагають $O(n)$ [6].

GQA ґрунтується на концепціях кубітів та суперпозиції станів квантової механіки. Найменша одиниця інформації, що зберігається у квантовому комп'ютері з двома станами, називається квантовим бітом або кубітом. Кубіт може бути в стані «1», в стані «0» або в будь -якій суперпозиції з двох. Стан кубіта можна представити як:

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1.1}$$

Де α і β – комплексні числа, які визначають амплітуди ймовірностей відповідних станів. $|\alpha|^2$ дає ймовірність того, що кубіт буде знайдений у стані «0», а $|\beta|^2$ - ймовірність того, що кубіт буде знайдений у стані «1». Нормалізація стану до гарантій єдності:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{1.2}$$

Якщо існує система m-кубітів, система може представляти 2^m станів одночасно. Однак під час спостереження за квантовим станом він руйнується до єдиного стану.

Теза Сильної Церкви-Тьюринга стверджує, що ймовірнісна машина Тьюринга може ефективно імітувати будь-яку реалістичну модель обчислень. Під «ефективним» ми маємо на увазі, що існує поліном p такий, що кількість ресурсів, що використовуються моделюванням машини Тьюринга, не більше

p *(M)*, де *M* – кількість ресурсів, що використовуються даною реалістичною моделлю обчислень. Оскільки комп'ютер є фізичним пристроєм, будь -яка розумна модель обчислень повинна бути передана в реалістичні фізичні рамки, отже, умова, що модель є "реалістичною", є цілком природною. Імовірнісна машина Тьюрінга неявно введена в класичні рамки фізики, і, здається, вона діє до тих пір, поки конкуруюча модель обчислень також відлита в класичній структурі. Однак приблизно століття тому була розроблена нова основа фізики - квантова механіка. Вплив цієї нової основи на теорію обчислень був сприйнятий не дуже серйозно до початку 1970-х років Стівеном Віснером для криптографічних цілей (а пізніше Беннетом і Брассардом).

Беніоф запропонував використовувати квантові пристрої для здійснення зворотних обчислень. Фейнман зауважив, що класичний комп'ютер, здається, не здатний ефективно моделювати динаміку досить простих квантово -механічних систем, і запропонував, щоб "квантовий" комп'ютер з компонентами, що розвиваються відповідно до квантово - механічних правил, міг би ефективно виконувати таке моделювання. Манін зробив подібне спостереження самостійно. Дойч працював над доведенням оригінальної тези Черч-Тьюрінга (яка стосувалася лише ефективних обчислень, а не ефективних обчислювальних можливостей) у квантово-механічній структурі та визначив дві моделі квантових обчислень; він також дав перший квантовий алгоритм. Одна з ідей Дойча полягає в тому, що квантові комп'ютери можуть використовувати переваги обчислювальної потужності, наявної у багатьох «паралельних всесвітах», і таким чином перевершувати звичайні класичні алгоритми. Хоча думка про паралельні всесвіти іноді є зручним способом для дослідників винаходити квантові алгоритми, алгоритми та їх успішна реалізація не залежать від будь -якої конкретної інтерпретації стандартної квантової механіки.

Квантові алгоритми – це алгоритми, які працюють на будь-якій реалістичній моделі квантових обчислень. Найбільш часто

використовуваною моделлю квантових обчислень є ланцюгова модель (строгіше, модель однорідних сімейств ациклічних квантових ланцюгів), а в квантовій сильній Черч-Тьюрингській тезі стверджується, що квантова схема може ефективно імітувати будь-яку реалістичну модель обчислень. Було розроблено кілька інших моделей квантових обчислень, які дійсно можна ефективно моделювати за допомогою квантових схем. Квантові схеми дуже нагадують більшість підходів, що застосовуються в даний час для спроб побудови масштабованих квантових комп'ютерів. Вивчення квантових алгоритмів дуже важливо з кількох причин. Обчислювально безпечна криптографія широко використовується в суспільстві сьогодні і спирається на труднощі невеликої кількості обчислювальних проблем. Схоже, що квантові обчислення переосмислюють те, що є відстежуваною чи нерозв'язною задачею, і одним із перших проривів у розвитку квантових алгоритмів стало відкриття Шором ефективних алгоритмів для факторингу та пошуку дискретних логарифмів.

Труднощі факторингу та пошуку дискретних логарифмів були і залишаються в основі використовуваної в даний час криптографії з відкритим ключем, і його результати показали, що якщо і коли побудований квантовий комп’ютер, будь-які особи, які записали зашифрований текст та відкриті ключі, можуть бути скомпрометовані будь-якими повідомленнями, які раніше були зашифровані за допомогою наших сучасних криптографічних інструментів з відкритим ключем. Крім того, величезна існуюча інфраструктура відкритих ключів була б скомпрометована без чіткої альтернативи замінити її; також для розгортання будь-якої альтернативи знадобиться багато років. Несподівано поспішили відповісти на два фундаментальних питання. По-перше, чи можна насправді побудувати досить великий квантовий комп’ютер? Можливо, це не розумна модель обчислення. Подальша робота над квантовою виправленням помилок свідчить про те, що відповідь «так», а прогрес експерименту неухильно зростав. По -друге, які ще цікаві та важливі проблеми можуть вирішити

квантові комп'ютери ефективніше, ніж найвідоміші класичні алгоритми? Наступні розділи аналізують стан техніки щодо другого питання.

Структура CGA проілюстрована на рисунку 3. Найпростіший спосіб кодування хромосом - це представляти їх двійковими рядками. Початковий Популяція повинна починатися з випадкових хромосом рівномірно розподілений по всьому простору пошуку. Наступний крок - операція оцінки. Його роль – позначити особи населення. Після цього, особи будуть сортуватися відповідно до їхніх позначок. Операція вибору має за мету обрати деяку кількість особи, щоб забезпечити відтворення. Перехресне операцію можна виконати шляхом обміну деякими частинами вибраних осіб у випадкових позиціях, що веде створити новий набір хромосом, що замінює старий одиниці. Перед повторенням процесу рекомендується провести мутацію, щоб виправити стохастичні помилки, щоб уникнути генетичного дрейфу та забезпечити генетичне різноманіття в популяції. Він складається з зміни деяких випадкових позицій індивідів з невеликою ймовірністю (зазвичай від 1% до 0,1%).

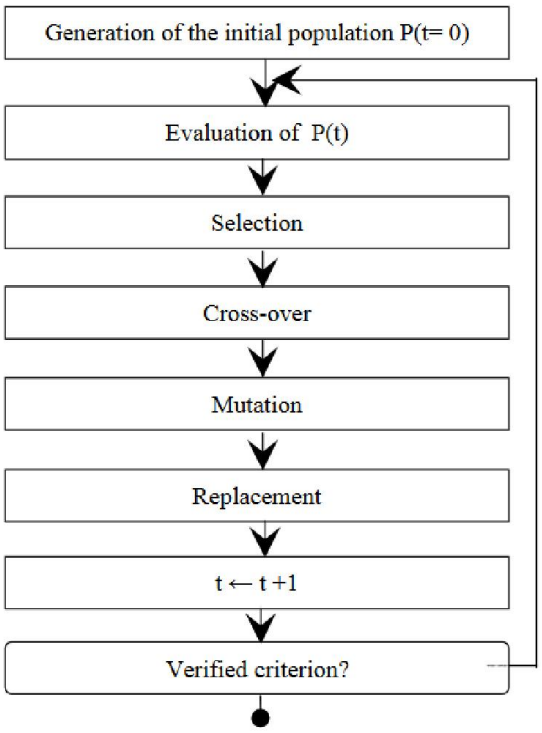


Рисунок 1.3 – Структура QGA.

1.2.1 Механізми квантових обчислень. Квантові обчислення характеризуються:

1. використанням квантового обчислювального методу, який ґрунтується на певних принципах квантової механіки, таких як інтерференція, когерентність та стоячі хвилі;

2. використання класичного алгоритму для перевірки того, що «кандидатні» рішення, створені квантовим методом обчислень, насправді є вірними.

Оскільки поняття вказані вище є більш фізичними, я зупинюсь задля їх пояснення. Гарним прикладом для пояснення інтерференції у квантовій механіці є експеримент зі щілинами або, інтерференційний дослід Юнга. Розглянемо непрозорий для світла екран із двома щілинами. Висвітлимо його від монохроматичного джерела. На фотопластинці позаду екрана з'явиться дифракційна картина, що узгоджується з уявленням про світло, як хвилю, викликана інтерференцією хвиль, що пройшли через дві щілини. Тепер розглянемо світло як потік частинок фотонів. З погляду класичної механіки, кожен фотон потрапляє на платівку через першу, або через другу щілину. Знайдемо на фотопластинці крапку з інтерференційним мінімумом освітленості. Закриємо одну щілину. З погляду уявлень класичної механіки, на жоден з фотонів, які проходять через іншу щілину, закриття цієї щілини не вплине. Тим не менш, ми побачимо, що інтерференційний мінімум освітленості зникне, на нього почнуть падати фотони з іншої щілини. Кожен окремий фотон починає поводитися як хвиля.

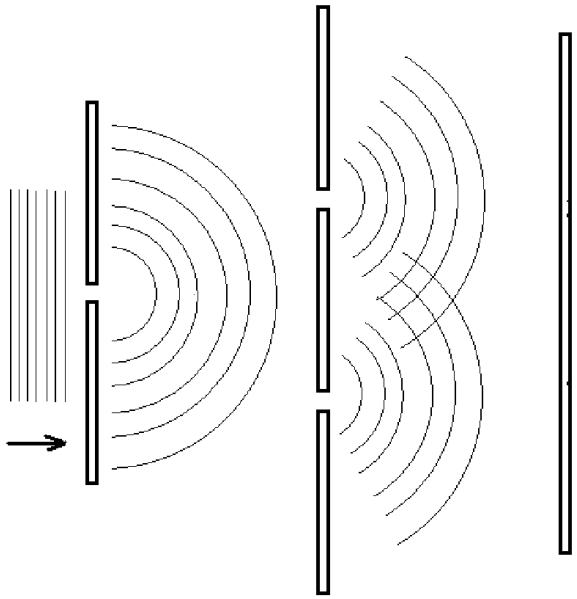


Рисунок 1.4 – Парадокс двох щілин у квантовій механіці

Щодо когерентності, у квантовій оптиці когерентним станом називають стан квантового електромагнітного поля, який найкраще описує когерентність та має поведінку схожу на класичну.Когерентність – властивість хвилі зберігати свої частотні, поляризаційні й фазові характеристики. Інтерференція та когерентність дуже тісно пов’язані між собою. Таким чином, когерентність світла пояснюють постійним у часі співвідношенням між фазами світлових хвиль, що створює можливість отримання інтерференції. Когерентні промені одержують від того самого джерела. Розрізняють повну і часткову когерентність світла. Повна когерентність настає тоді, коли контраст інтерференційної картини ідеальний, тобто мінімальна інтенсивність світла в області тіні дорівнює нулю; часткова – якщо контраст не ідеальний. Якщо контраст відсутній, то світло цілком некогерентне.

Квантова когерентність – це одна з властивостей, які мають квантові частинки. Формально його можна запровадити і для класичних макроскопічних об'єктів, але тоді виникнуть парадокси, такі як кіт Шредінгера. Всім відомий кіт Шредінгера знаходиться в суперпозиції двох станів: він і живий, і мертвий. Бути живим чи мертвим – це два

взаємовиключні стани, однак у квантовій механіці частинки можуть бути не тільки в одному з них, а й у кількох відразу.

Поняття квантової когерентності з'явилося на зорі квантової механіки – у той час, коли люди зрозуміли математичний апарат і отримали уявлення про лінійність простору, що описує квантові стани. Елементи цього простору описуються векторами. Якщо в цьому просторі є два вектори, наприклад *a* і *b*, їх можна скласти і отримати третій, *a+b*. Взаємовиключним подіям відповідають ортогональні вектори. Коли у вас є два взаємовиключні стани – наприклад, живий (вектор *a*) і мертвий (вектор *b*), — існує ще й стан виду: *√(½)* живий плюс *√(½)* мертвий, тобто *√(½a+½b)*. Коефіцієнти можуть бути іншими чи комплексними. Таким чином, самою теорією закладено існування суперпозиційних станів, оскільки гільбертовий простір (узагальнення евклідового простору, що допускає нескінченну розмірність і повноту за метрикою, породженою скалярним твором) лінійний.

Це дуже цікаво, тому що ми говоримо про здатність об'єкта мати одночасно дві властивості. Наприклад, у вас є яблуко, яке може бути червоним і зеленим. Або струм у надпровідниковому кільці може текти і вправо, і вліво. Це і є принцип суперпозиції. А коли суперпозиція руйнується, у термінах української мови треба говорити «або-або». Тобто яблуко або червоне, або зелене. Струм у кільці тече або в один бік, або в інший. Союз «або» описує наше незнання. Якщо відчути різницю на мовному рівні, то стане очевидно, що мати обидві властивості одночасно і мати тільки одну з двох властивостей – це дві абсолютно різні ситуації. Перша відповідає когерентності, а друга – її відсутності.

Розглянемо практичну значимість квантової суперпозиції. Допустимо, існує завдання пошуку якоїсь інформації в неструктурованій базі даних. Якщо у вас є телефонний довідник, то по прізвищу людини легко знайти його телефон. Інша справа – зворотне завдання, коли необхідно знайти прізвище людини у телефонному довіднику, знаючи його номер. Це досить складно зробити, якщо довідник великий, оскільки в класичному випадку це

вирішується лише методом перебору та вимагає в середньому *N/2* звернень до довідника, де *N* – число номерів. Однак у квантовому алгоритмі Гровера вона вирішується швидше. У квантовому випадку всі номери телефонів – це і є всілякі ортогональні вектори в просторі гільберта *(a, b, c, ...)*. Принцип квантової суперпозиції означає, що у цьому просторі може існувати і когерентний квантовий стан виду $(a + b + c + ...)/\sqrt{N}$, де *N* – число номерів у довіднику. Квантовий алгоритм Гровера починає свою роботу саме з суперпозиції всіх цих станів. Тобто ваш початковий стан – це когерентна суперпозиція всіляких телефонних номерів, які є. У класичній фізиці це в принципі неможливо, оскільки у вас є лише якийсь конкретний номер (перший або другий, або двадцять п’ятий, або тридцять шостий). А в квантовій механіці у вас суперпозиція і першого, і другого, і двадцять п'ятого, і тридцять шостого номера всіх відразу.

Звісно, амплітуда $(1/\sqrt{N})$ цих станів мала. Але алгоритм побудований таким чином, щоб він зростав у того номера, який ви шукаєте. Тобто в результаті виходить стан виду $0{,}001\ a + 0{,}001\ b + 0{,}001\ c + ... + 0{,}995\ x + 0{,}001\ y + ...$ де *x* –номер телефону. Це дає змогу прискорити пошук квадратично. Наприклад, якщо у вас десять тисяч телефонних номерів, то класичний пошук зажадав близько п'яти тисяч звернень до довідника, оскільки потрібно перевірити приблизно половину. А в квантовому випадку вам знадобиться близько ста звернень. Виграш становитиме $\sqrt{N}$, де *N* — розмір бази даних. Це означає, що квантова суперпозиція (вона когерентність) потрібна саме для реалізації квантових алгоритмів.

Стоячі хвилі також мають пряме відношення до інтерференції та когерентності. Оскільки, стоячі хвилі – це результат накладання двох біжучих когерентних хвиль з однаковими амплітудами, які поширюються назустріч одна одній. Також, стояча хвиля – явище інтерференції хвиль, що розповсюджуються в протилежних напрямках, при якому перенос енергії ослаблений або відсутній. Типові приклади таких хвиль коливання електромагнітні коливання НВЧ-діапазону всередині мікрохвильової печі.

Стоячі хвилі виникають при складанні падаючої і відбитої хвиль, що біжать, у сприятливих умовах. Ці умови, як правило, у тому, щоб у області простору, відведеної для хвилі (в резонаторі), укладалося ціле число довжин півхвиль. Якщо в резонаторі утворюються хвилі різних довжин, то посиляться і накопичаться саме ті, для яких виконується ця умова. Лазерне випромінювання до того, як воно вийшло з лазера, накопичується там саме у вигляді стоячих хвиль. На рисунку нижче приведено приклад стоячої хвилі.

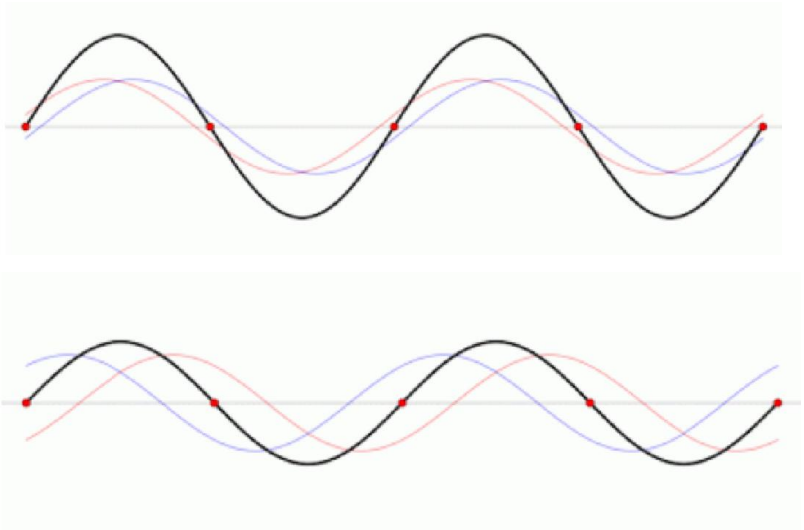


Рисунок 1.5 – Приклад стоячої хвилі як результат накладання падаючої на відбивної

Тепер, розібравшись з поняттями квантових обчислень, розберемо структуру самого атома. Ядро містить частинки, які називаються протонами (позитивно заряджені) і нейтрони (електрично нейтральні). Ядро оточене електронами (негативно зарядженими). Орбіти електронів, згідно з квантовою механікою, не є площинними, як орбіта нашої планети Землі. Орбіту електронів краще описати як хвилю, як на рис. 7. Існує кілька типів орбіт залежно від моменту та рівня енергії. Електрон «перескакує» з орбіти з

низьким енергетичним рівнем на орбіту з вищим рівнем енергії, поглинаючи енергію (фотон). За таким же принципом і працюють лазери.

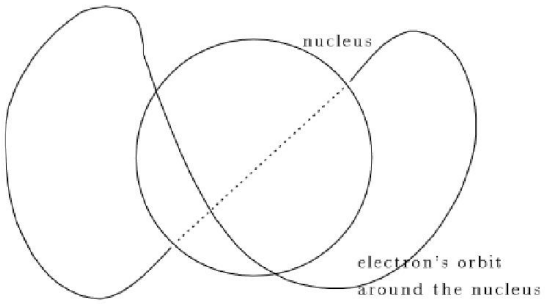


Рисунок 1.6 – Електрон проходить повний цикл навколо ядра, починаючи з деякої точки перед ядром, обходячи його задню частину, що показано пунктирною лінією, перш ніж завершити свій цикл

Термін «квант» означає, що не може бути проміжних станів чи орбіт. Стрибки є дискретними. Наприклад у атома водню таких станів-орбіт існує всього 6, їх називають енергетичними рівнями, які можна приблизно охарактеризувати як кількість циклі, які електрон повинен пройти, обертаючись навколо ядра на цьому енергетичному рівні. Іншими словами, електрон з першого рівня обертається навколо ядра з одним циклом, на другому енергетичному рівні з двома і так далі. Електрон, знов таки, може стрибнути дискретно, назад на інший рівень, вивільнивши енергію (фотон). Отже, електрон навколо ядра переходить у стани в дискретних квантах, поглинаючи енергії або вивільнюючи її. Розташування квантових частинок найкраще можна описати вектором квантового стану $|\Psi\rangle$, а зважена сума двох можливих місць розташування дорівнює $w|A\rangle + x\ |B\rangle$, де w та x є ваговими коефіцієнтами комплексного числа частинки, що знаходиться в місцях A та B, відповідно, і де $w|A\rangle$ та $x\ |B\rangle$ є самими векторами стану. Для n можливих станів буде n різних вагових коефіцієнтів комплексного числа, кожен з яких описує зважену ймовірність перебування частинки у цьому місці. Вектор$|\Psi\rangle$ представляє собою лінійну суперпозицію частинки, заданої

окремими квантовими векторами стану: якщо існує *n* місць, заданих *n* векторами стану, кажуть що частинка знаходиться у всіх *n* місцях одночасно (багато всесвітів). Однак під час спостереження за квантовим станом, або хвильовою функцією, він руйнується до єдиного стану. Колапс також може відбутися через «інтерактивну декогеренцію», коли нероздільні кореляції між розташуванням векторів стану об'єкта та його оточення розділять можливі стани на бажані (колапс хвильової функції). Річард Фейнман вже показав у своїх дослідах, як квантова система може використовуватись для виконання обчислень і діяти як симулятор квантових процесів, де таке моделювання неможливо здійснити на звичайному автоматі [31].

1.2.2 Квантовий факторинг. Інтерпретація «багато всесвітів» була використана Шором у його описі методу квантових обчислень для вилучення простого коефіцієнту дуже великих цілих чисел, де регістр пам'яті поміщається в суперпозицію всіх можливих цілих чисел, які він може містити, після чого в кожному «всесвіті» виконується різне обчислення. Обчислення припиняється, коли різні всесвіти заважають один одному, оскільки повторювані послідовності цілих чисел (стоячі хвилі) зустрічаються в кожному всесвіті та між всесвітами (форма інтерактивної декогеренції, яка не залежить від спостереження). Хоча немає гарантії, що результати вірні, на цьому етапі можна провести наступну перевірку, щоб визначити, чи дійсно отримані числа є простими множниками даного великого цілого числа.

Криптографія використовується для кодування секретних операцій банків, урядів і військових. Системи відкритих ключів широко використовують ключові методи виробництва на основі RSA. Це працює таким чином, що два різні великі простих числа вибираються випадковим чином, p і q. Обчислюється їх добуток $r = p \times q$. Вибирається велике ціле число e, яке є відносно простим з $(p - 1) \times (q - 1)$; e ключ шифрування. Ключ дешифрування d є унікальним, оберненим до e $mod\ (p - 1) \times (q - 1)$. r і e

зберігається у загальнодоступних домену файлах, але *d* залишається приватним. Для секретного зв'язку від X до Y: X шифрує повідомлення за допомогою відкритого ключа Y; тільки закритий ключ Y може розшифрувати повідомлення. Для дозволеного зв'язку від X до Y: X шифрує повідомлення з використанням приватного ключа X; Y використовуватиме відкритий ключ X для розшифрування повідомлення. Для секретного та дозволеного повідомлення від X до Y, X використовує приватний ключ X, а потім відкритий ключ Y, має використовувати закритий ключ Y, а потім відкритий ключ X відновити повідомлення.

Успіх RSA залежить від уявної нерозв'язності проблеми пошуку простих множників дуже великих цілих чисел. Коли система була винайдена ще в 1977 році, її винахідники кидали виклик будь-кому розкласти на множники конкретне 129-значне число, тепер відоме як RSA 129. Ця проблема трималась до 1994 року, коли 1600 комп’ютерів, пов’язаних через Інтернет, змогли визначити фактори трохи більше ніж за вісім місяців. Незважаючи на цей прорив, криптографи заспокоїли себе думкою про додавання більше цифр до своїх кодів. Проблема розкладання чисел на множники стає експоненціально складнішою, від збільшення числа цих самих множників. Проте Шор [32] визначив квантовий алгоритм, який би розбив RSA-129 на множники за кілька секунд, якби можна було створити квантовий комп’ютер, який працював би так само швидко, як сучасний ПК. Щоб розкласти число *n* (добуток двох простих чисел), вкажіть довільну кількість паралельних всесвітів *p (0, 1, 2...)* і випадковим чином виберіть ціле число *x* а від 0 до *n*. Потім у кожному всесвіті підвищіть *x* у степені числа всесвіту. Розділіть на *n* і збережіть залишок у всесвіті. Для наступного числа в послідовності для всесвіту, *r* підвищується до степеня останнього збереженого числа, поділеного на *n* і збереженого залишку. Це продовжується в кожному всесвіті, утворюючи повторювану послідовність. Для пояснення нехай *n* (число, яке потрібно розкласти на прості множники)

буде 33. Нехай $x = 7, (0 < x < n), p = 17$. У таблиці 1 наведено детальний опис того, що відбувається в кожному всесвіті u_oдо u_{16}.

Таблиця 1.1 – Наведено 16 «всесвітів» та їх перетворення для обчислення простого числа 33, де $x = 7$

u_0:	1	7	28	31	7	28	31	7
u_1:	7	28	31	7	28	31	7	28
u_2:	16	4	25	10	1	7	28	31
u_3:	13	13	13	13	13	13	13	13
u_4:	25	10	1	7	28	31	7	28
u_5:	10	1	7	28	31	7	28	31
u_6:	4	25	10	1	7	28	31	7
u_7:	28	31	7	28	31	7	28	31
u_8:	31	7	28	31	7	28	31	7
u_9:	19	19	19	19	19	19	19	19
u_{10}:	1	7	28	31	7	28	31	7
u_{11}:	7	28	31	7	28	31	7	28
u_{12}:	16	4	25	10	1	7	28	31
u_{13}:	13	13	13	13	13	13	13	13
u_{14}:	25	10	1	7	28	31	7	28
u_{15}:	10	1	7	28	31	7	28	31
u_{16}:	4	25	10	1	7	28	31	7

Розглянемо u_2 таблиці 1: $7^2 \bmod 33 = 16$, $7^{16} \bmod 33 = 4$, $7^4 \bmod 33 = 25$, $7^{25} \bmod 33 = 10$, $7^{10} \bmod 33 = 1$. тепер розглянемо u_5: $7 \bmod 33 = 10$, 7^{10} *мод* $33 = 1$, $7^1 \bmod 33 = 7$, $7^7 \bmod 33 = 28$, $7^{28} \bmod 33 = 31$ і т. д. Можна помітити, що строки повторюються з частотою 10, тобто $u_1 = u_{10}$, $u_2 = u_{12}$, $u_3 = u_{13}$, і так далі. У кожному з цих всесвітів повторювана частота починається в одній і тій же точці і повторює ті самі числа. У той час як інші всесвіти поділяють повторюваний шаблон, вони не мають спільної вихідної точки. Це квантовий обчислювальний еквівалент стоячих хвиль у кожному всесвіті: повторення числових значень. Розрахунок $x^{u(f)/2} - 1$, де x — довільно вибране число від 0 до n, та $u(f)$ — частота звичайних повторюваних моделей всесвітів. Отже, розрахувавши за цією формулою ми отримаємо $7^{10/2} - 1 \pmod{33}$, маємо 9. Тепер знайдемо найбільший спільний дільник чисел 9 та 33, що дасть нам один з факторів, тобто 3.

Не гарантується, що метод Шора буде працювати завжди, і якщо похідне число виявиться не простим множником *n*, процедура повторюється з використанням іншого *x*. Стверджується, що в середньому буде потрібно лише кілька спроб для розкладання *n* на множники, навіть якщо *n* дуже велике. Немає відомих швидких класичних алгоритмів для розкладання великих чисел на прості, але у методі Шора використовуються відомі швидкі алгоритми для визначення кандидата простого множника *n* і визначення того, чи він насправді такий. Метод Шора був розширений, щоб мати змогу працювати з розкладанням великих парних чисел, сортуванням та нейронними мережами.

1.2.3 Квантові генетичні операції. Квантові генетичні операції – це операції, які використовують при генетичному алгоритмі у комбінації з квантовим. В основному вони базуються на кубітах і суперпозиції станів квантової механіки. Наприклад:

1. Вимірювання хромосом: Для того, щоб використовувати ефективно накладені стани кубітів, ми повинні спостерігати за кожним кубітом. Це призводить нас до вилучення класичної хромосоми, як показано на рисунку 4. Мета – дати можливість оцінити кожну квантову хромосому.

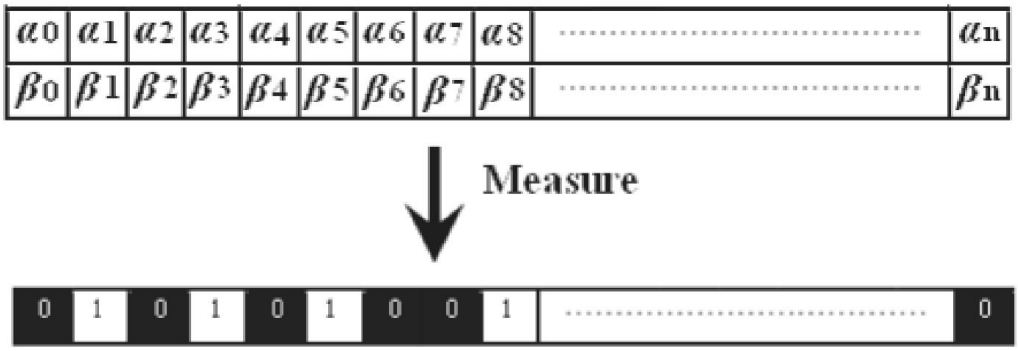


Рисунок 1.7 – Подання форми хромосоми у квантовому генетичному алгоритмі

2. Перешкоди: Ця операція дозволяє змінювати амплітуди популяції з метою покращення продуктивності. В основному він складається з переміщення стану кожного кубіта у сенсі вартості найкращого рішення. Це корисно для активізації пошуку найкращого рішення. Його можна виконати за допомогою одиничного перетворення, що дозволяє поворот, кут якого є функцією амплітуд *(a_i, bi)* та значення відповідного біта у еталонному розв'язку. Значення кута повороту $\delta\theta$ слід вибирати так, щоб уникнути передчасного зближення. Часто це визначається емпірично, а його напрямок визначається як функція значень a_i, b_i та значення кубіту, розташованого у положенні і у особи, що змінюється.

3. Стратегія ротаційних воріт *Qubit*: обертання амплітуди людини виконуються квантовими затворами. Квантові ворота також можуть бути спроектовані в відповідно до поточної проблеми. Файл популяція Q (t) оновлюється обертанням квантових воріт кубітів, що складають окремих осіб. Прийнята стратегія обертання визначається таким рівнянням:

$$\begin{bmatrix} \alpha_i^{t+1} \\ \beta_i^{t+1} \end{bmatrix} = \begin{bmatrix} cos(\Delta\theta_i) & -sin(\Delta\theta_i) \\ sin(\Delta\theta_i) & cos(\Delta\theta_i) \end{bmatrix} \begin{bmatrix} \alpha_i^{t} \\ \beta_i^{t} \end{bmatrix} \quad (1.3)$$

QGA - це комбінація між GA та квантовими обчисленнями.На відміну від класичного представлення хромосом (наприклад, двійковий рядок), тут вони представлені векторами кубітів (квантовий регістр).Таким чином, хромосома може представляти суперпозицію всіх можливих станів.Структура QGA проілюстрована на рис. 1.8.

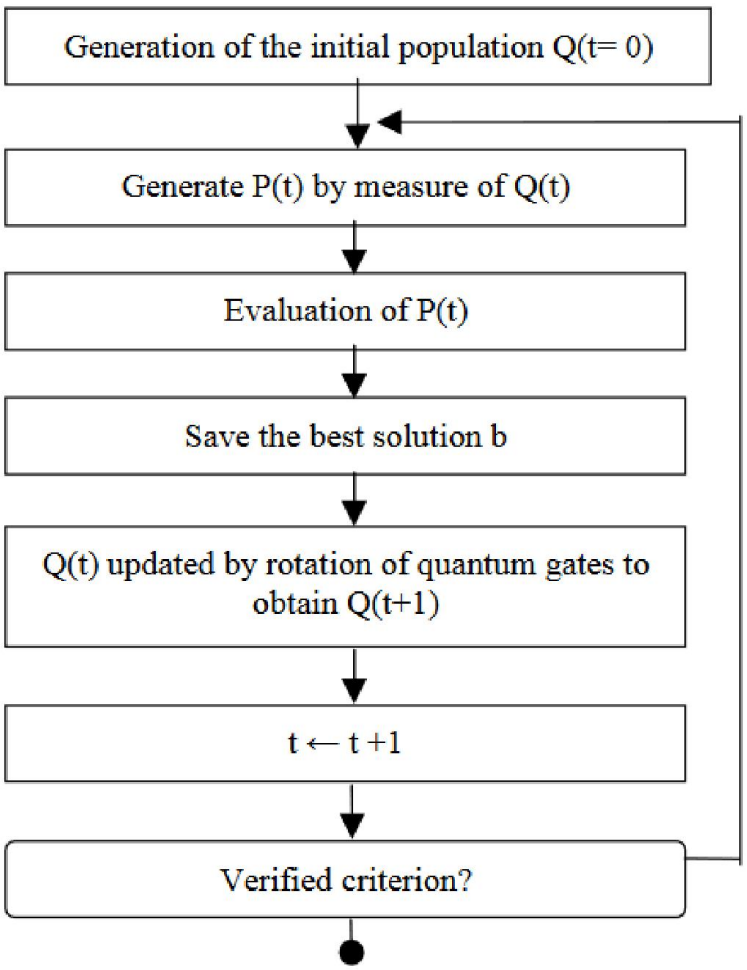


Рисунок 1.8 – Реалізація QGA

1.3 Огляд квантових генетичних алгоритмів у задачах оптимізації

Вище було розглянуто алгоритм та методи його застосування у вирішенні задачі комівояжера. Ця проблема стосується NP-повних задач. Для прикладу наведу інші оптимізаційні задачі, які може вирішувати квантовий генетичний алгоритм і чудово з ними справляється. Принципова відмінність полягає у особливостях реалізації квантових операторів з урахуванням подання хромосоми у вигляді системи кубітів та принципу суперпозиції. І хоча такі алгоритми мають широкий спектр застосування, головний алгоритм можна подати у вигляді:

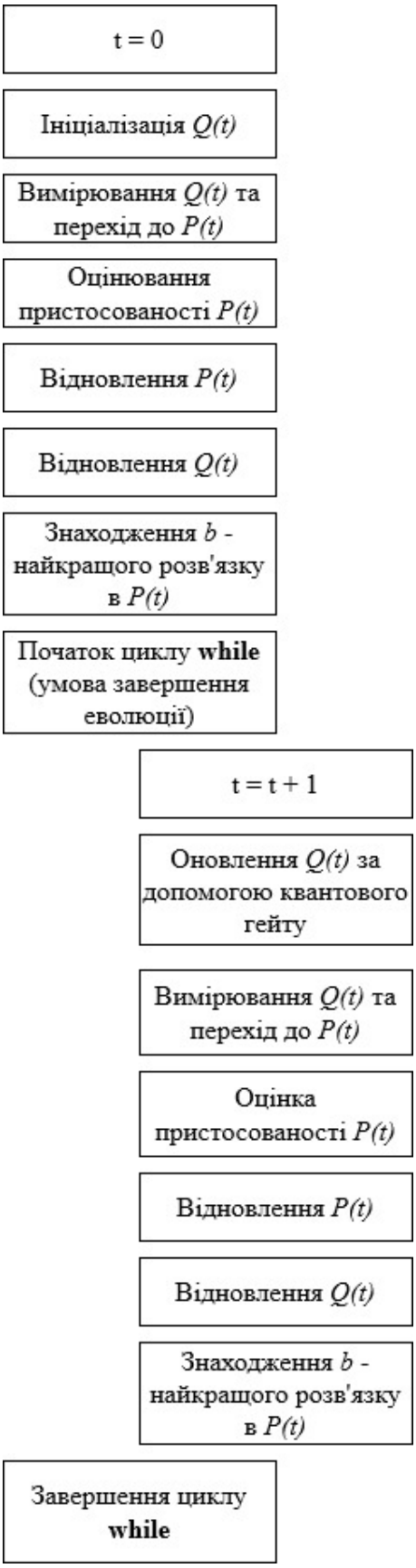


Рисунок 1.9 – Загальна схема алгоритму

На даній схемі, *Q(t)* – квантова популяція розв'язків на момент часу *t*; *P(t)* – класичне (бінарне) подання популяції, отримане в результаті квантового вимірювання. Операцію квантового вимірювання (етапи 3 та 11) детально розглянуто у працях [34, 35]. Далі розглянемо операції відновлення та роботу їх алгоритму. Операція квантового гейту до певної міри виконує роль операції схрещування в її класичному розумінні. Алгоритм роботи оператора квантової хромосоми (квантового гейту), що складається з *N* кубітів, можна реалізувати таким чином:

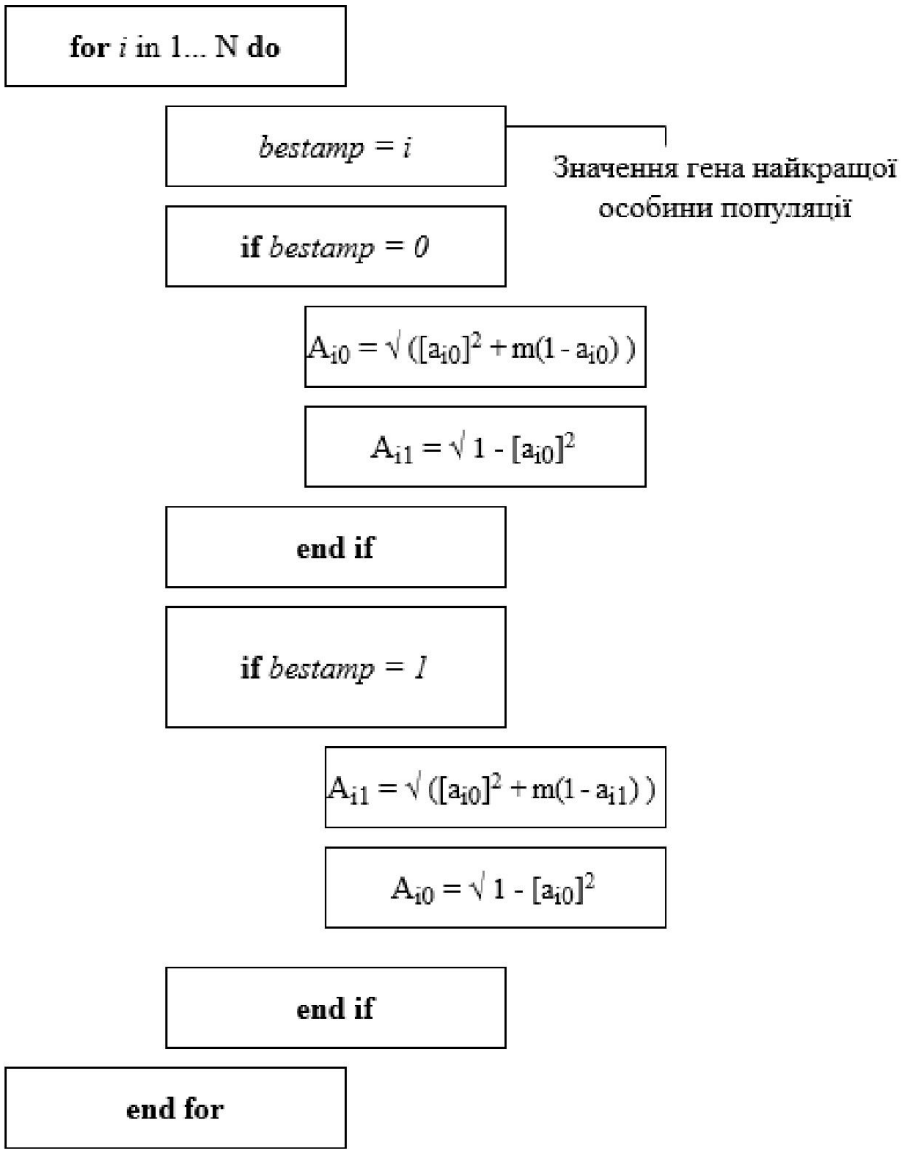


Рисунок 1.10 – Алгоритм оператора квантового гейту

На етапі 4 (8) алгоритму відбувається адаптивний поворот вектора квантового стану, що відповідає найкращій особині популяції, а на етапі 5 (9) перерахунок амплітуди іншої імовірності для забезпечення виконання умови нормування. Таким чином, у кожному новому поколінні забезпечується збільшення ймовірності того, що в результаті спостереження генеруються класичні особини, більш схожі на найкращу.

1.3.1 Вирішення задачі комівояжера за допомогою квантових механізмів. У розділі 1.2.2 було розглянуто факторинг за допомогою якого вирішимо задачу комівояжера. Звучить вона таким чином: ми маємо *n* точок та дистанцію *D,* задача у тому, чи існує такий тур, який відвідає кожне з *n* місць рівно один раз, повертаючись до початкової точки та матиме довжину шляху ≤ *D?* Опис мережі *D* наведено у таблиці 1.2.

Таблиця 1.2 – Дистанції між містами.

I	4	27	16	12	18	29	22	12	∞
H	16	21	29	10	9	23	13	∞	12
G	8	19	24	15	5	16	∞	13	22
F	12	13	5	20	32	∞	16	23	29
E	13	6	10	30	∞	32	5	9	18
D	19	2	25	∞	30	20	15	10	12
C	7	11	∞	25	10	5	24	29	16
B	13	∞	11	2	6	13	19	21	27
A	∞	13	7	19	13	12	8	16	4
	A	B	C	D	E	F	G	H	I

Числа у таблиці означають відстань між парами міст (вузлів), саме тому чисел між однаковими назвами міст немає. З таблиці ми бачимо, що наприклад, відстань від *A* до *G* дорівнює 8. Якщо вирішувати таку задачу методом перебору, тоді за допомогою комбінаторики ми отримаємо 9! комбінацій для перебору, тобто 362880 комбінацій, такий метод більше

схожий на грубу силу та є досить непрактичним, тим паче, при збільшені кількості міст (вузлів) хоча б на 1, ми отримаємо 10 мільярдів комбінацій. Генетичні алгоритми допомагають вирішити задачу оптимальніше. Класичний маршрут через мережу можна представити у вигляді хромосоми, що складається з 10 генів, кожен з яких є літерою між A – I. Перший ген у хромосомі ідентичний десятому гену в кожному випадку, оскільки кожен маршрут через мережу має бути побудований так, щоб повернутися до свого початкового вузла. Для наочності розглянемо таку хромосому:

B	C	E	I	G	A	H	D	F	B

Це являє собою такий тур: B → C→ E → I → G → A→ H → D → F → B. Цей маршрут через мережу має загальну відстань: 11 + 10 + 18 + 22 + 8 + 16 + 10 + 20 + 13 = 128 (табл. 2). Для мутації два з перших дев’яти генетичних значень (алелів) хромосоми вибираються випадковим чином і обмінюються положеннями, гарантуючи, що перший і десятий алелі залишаються ідентичними. Наприклад, вищенаведена хромосома може мутувати в:

G	C	E	I	B	A	H	D	F	G

B у першій позиції замінюється на *G* у п’ятій позиції, а останній *B* змінюється на *G* для збереження циклу. Імовірність мутації будь-якої хромосоми в популяції вибирається довільно і становить 0,667. Дві хромосоми можуть піддаватися кросинговеру. Цей кросовер є квантовою формою фіксованої точки: візьміть 1-й елемент першої хромосоми, 2-й елемент другої хромосоми, 3-й елемент першої хромосоми, 4-й елемент другої хромосоми і таким чином, якщо елемент вже присутній у хромосомі, виберіть наступний алфавітний алель, який ще не міститься в хромосомі. Наприклад, схрещування:

A	C	D	I	B	G	H	F	E	A
G	C	E	H	B	A	I	D	F	G

Створює нові хромосоми:

A	C	D	H	B	E	I	F	G	A

G	C	E	I	B	H	A	F	D	G

Тут перша хромосома складається з *A* в першому положенні у верхній хромосомі *(uc)*, *C* у другій позиції нижньої хромосоми *(lc)*, *D* у третій позиції *uc*, *H* у четвертій позиції *le* і *B* у п'ятій позиції *uc*. У цей момент A в шостому положенні *le* не може бути включено, оскільки *A* вже зустрічається в першій отриманій хромосомі, тому вибирається наступний алфавітний алель з урахуванням тих, які вже є в отриманій хромосомі. Оскільки перша результуюча хромосома вже містить *A, B, C* і *D*, замість *A* обирається *E*. Імовірність вибору хромосоми для кросинговеру вибирається довільно і становить 0,667. Не допускається дублювання хромосом. Розмір популяції був обраний так, щоб дорівнювати кількості вузлів у мережі, тобто 9. Якщо після мутації та кросинговеру залишилося більше 9 хромосом, лише 9 можуть продовжитися в наступному поколінні. Нове покоління створюється шляхом збереження хромосом з найкоротшим мережевим туром, тобто з найменшим показником *D*, а решта 8 вцілілих вибираються випадковим чином з кандидатів на відновлення відповідно до методу рангового простору, методу придатності, що враховує комбінований рейтинг дистанційного рангу та рангу різноманітності [33]. Щоб отримати дистанційний рейтинг, решта кандидатів сортуються в порядку дистанцій, тоді кандидату з найкоротшою відстанню дається рейтинг відстані 1, наступному найкращому кандидату дається рейтинг відстані 2 тощо. Ранг різноманітності хромосоми визначається шляхом обчислення суми обернених квадратів різниць між цією

хромосомою та інших, вже відібраних, хромосом. Це робиться шляхом присвоєння кожному алелю цілого числа, напр. *A =1, B = 2, C = 3, D = 4* і т. д. Ранг різноманітності розраховується наступним чином: $\sum_i 1/d^2_i$. Кандидату з найменшим значенням різноманітності присвоюється рейтинг різноманітності 1, наступному найкращому кандидату надається рейтинг різноманітності 2 і т. д. Далі оцінки за відстань та різноманітність додаються разом, щоб отримати повний комбінований бал.

Тепер застосовується метод придатності рангу. Придатність найкращого кандидата за комбінованим рангом є деякою фіксованою константою, у цьому випадку *p = 0,667*. Це значення *p* є ймовірністю. Якщо найкращого кандидата, того, хто займає 1-е місце, не обрано, то наступний найкращий кандидат, той, хто займе 2-е місце, обирається з відповідністю *p*. Цей процес відбору триває до тих пір, поки не буде обраний кандидат для наступного покоління або не залишиться лише один кандидат, і в цьому випадку буде обраний останній кандидат. Наприклад, за наведеною мережею в таблиці 2 ми обрали таку хромосому:

E	H	G	I	A	F	C	B	D	E

Представимо хромосому у вигляді відповідних числових зачень: 5 – 8 – 7 – 9 – 1 – 6 – 3 – 2 – 4 – 5.

У таблиці 3 більш детально описана процедура відбору. Ранг різноманітності показує, наскільки хромосома відрізняється від хромосоми-переможця, а хромосоми з високою різноманітністю отримують високий рейтинг. Такий принцип підтримується щоб показати принцип, що бути відмінним від інших хромосом так само добре, як і триматися позиції максимуму в просторі пошуку, задля подальшого визначення потенційних глобальних максимумів. Ранг відстані базується на фактичній відстані маршруту, представленого хромосомою, при цьому коротші маршрути мають вищий рейтинг. Сума рангів цих двох, призводить до загального комбінованого рангу, з ймовірністю того, що перша хромосома матиме шанс вижити 0,667, друга 0,222 тощо. Якщо 8 хромосом не пройдуть цей процес

відбору, щоб приєднатися до хромомсоми-переможця, для складання чисел використовується метод рандомних чисел. Наприклад, 1 – 3 – 2 – 5 – 6 – 4 – 7 – 8 – 9 – 1 обрано для наступного покоління з вірогідністю 0,667. Підсумовуючи цей «класичний генетичний алгоритм»:

1. Пошук починається з дев’яти хромосом, які вибираються випадковим чином;
2. Жодна хромосома не може з’являтися більше одного разу в кожному поколінні;
3. Дозволяється не більше дев’яти хромосомам перейти до наступного покоління;
4. Хромосома що вижила мутує з визначеною користувачем ймовірністю 0,667;
5. Два гени вибираються випадковим чином у кожного з цих вижившх, і вони обмінюються положеннями в хромосомі;
6. Хромосоми вибираються для кросинговеру з визначеною користувачем імовірністю 0,667;
7. Хромосома з найменшою відстанню подорожі доживає до наступного покоління;
8. Решта, що вижили вибираються з кандидатів відповідно до методу рангового простору і подальшого випадкового відбору, якщо потрібно;
9. Еволюція припиняється, як тільки «оптимальна» відстань не покращується для кількості поколінь, що дорівнює половині числа поколінь, необхідного для пошуку «оптимальної» хромосоми (таблиця 1.3).

Таблиця 1.3 – Опис процесу обрання хромосоми-переможця
5 – 8 – 7 – 9 – 1 – 6 – 3 – 2 – 5 з 15 ітерацій

Chromosome	$\sum_i \frac{1}{d_i^2}$	Diversity Rank	Distance Rank	Rank Sum	Combined Rank	Fitness	Distance
3 7 2 9 4 5 6 8 1 3	0.011	11	15	26	14	0.000	190
1 2 5 4 6 3 7 9 8 1	0.005	01	10	11	04	0.025	148
1 3 8 5 9 2 4 6 7 1	0.006	05	07	12	07	0.001	136
1 9 2 8 6 7 3 4 5 1	0.014	13	14	27	15	0.000	183
5 6 7 8 4 9 1 2 3 5	0.036	15	04	19	10	0.000	121
7 8 9 3 1 2 6 4 5 7	0.014	14	05	19	09	0.000	129
9 4 8 5 7 3 1 6 2 9	0.008	08	01	09	03	0.074	119
2 8 1 7 4 5 6 9 3 2	0.008	07	13	20	11	0.000	178
2 6 3 8 4 5 7 9 1 2	0.009	10	06	16	08	0.000	131
5 2 7 4 1 3 6 9 8 5	0.007	06	03	09	02	0.222	121
1 6 5 8 7 2 9 3 4 1	0.009	09	12	21	12	0.000	172
1 3 5 6 7 2 8 9 4 1	0.006	03	09	12	06	0.003	148
1 2 8 4 9 3 5 6 7 1	0.006	04	08	12	05	0.008	138
1 9 8 2 3 7 4 5 6 1	0.012	12	11	23	13	0.000	161
1 3 2 5 6 4 7 8 9 1	0.005	02	02	04	01	0.667	120

1.3.2 Перетворення задачі комівояжера за допомогою квантового генетичного алгоритму. Вище вже була розглянута задача комівояжера та приклад роботи генетичного алгоритму у вирішенні цієї задачі. Квантовий генетичний алгоритм вирішує її ще оптимальніше.

Кількість «всесвітів», необхідних для алгоритму, *u*, дорівнює кількості вузлів (міст) у мережі, у нашому випадку 9. Кожен всесвіт містить свою власну популяцію хромосом. Популяції в кожному всесвіті підкоряються правилам, ідентичним класичному випадку, і розвиваються паралельно (тобто кількість поколінь завжди однакова). Проте є одна відмінність – всесвіти можуть втручатися в роботу один одного в кожному поколінні. Ця інтерференція виникає як тип кросинговеру, що включає 9 хромосом, і відбувається відразу після класичного кросинговеру в кожному всесвіті. У кожному всесвіті є список хромосом, що становлять популяцію, і кожна хромосома в кожному всесвіті має позицію *i* індексу списку у своєму всесвіті. Ймовірність спарювання хромосом у *i*-й позиції у всіх всесвітах вибирається довільно рівною 0,667. Тільки генам в окремих всесвітах з однаковою позицією індексу в списку надається можливість кросинговеру. У результаті, отримані дев’ять хромосом розміщуються так, що кожна

призначається окремому всесвіту. Як і в класичному алгоритмі, дублікати не допускаються в межах одного всесвіту, або ж простору. Інтерференційний кросовер відбувається так: беремо 1-й елемент першої хромосоми, 2-й елемент другої хромосоми, 3-й елемент третьої хромосоми, 4-й елемент четвертої хромосоми і так далі.

Якщо елемент вже присутній у хромосомі, виберіть наступний за алфавітом елемент, який ще не міститься в хромосомі, у таблиці 1.4 описані всесвіти після інтерференційного кросинговеру, для однієї хромосоми близько чотирьох з кожного всесвіту. У таблиці 1.5, *G* (перше положення в першій хромосомі простору u_0) поєднується з *C* (друге положення в другій хромосомі всесвіту u_1) і з *H* (третє положення в третій хромосомі в u_3) тощо. Якщо значення гена вже з'являється, то обирається наступний алфавітний алель, як і раніше. Зверніть увагу на обернене перетворення: *A* (перша позиція першої хромосоми в u_8) слідуючи за *A* (друга позиція другої хромосоми в u_0) змінюється на *B* як алфавітна заміна для дублювання. Зауважте також, що кінцевий алель (кінцевий стовпець, не заштрихований) має бути таким же, як і перший, щоб повернутись до початкової точки.

Таблиця 1.4 – Опис результатів інтерференційного кросовера.

Universe	Generated chromosomes									
u0:	G	C	H	I	A	B	D	E	F	G
u1:	E	A	C	H	D	I	F	G	H	E
u2:	I	D	A	H	E	B	G	C	F	I
u3:	A	F	G	H	I	B	E	C	D	A
u4:	D	A	I	E	B	F	H	C	G	D
u5:	C	F	G	H	I	A	E	B	D	C
u6:	B	H	D	E	F	C	G	I	A	B
u7:	I	C	B	D	E	F	G	H	A	I
u8:	A	B	C	G	E	D	F	H	I	A

Таблиця 1.5 – Опис результатів інтерференційного кросовера між 9 хромосомами у 9 всесвітах.

U	Chromosomes									
u0	G	A	B	D	C	F	E	I	H	G
u1	E	C	A	B	D	G	H	F	I	E
u2	I	A	H	G	B	C	E	F	D	I
u3	A	D	C	I	E	B	G	H	F	A
u4	D	F	I	H	G	A	C	B	E	D
u5	C	A	G	H	D	B	F	I	E	C
u6	B	F	I	G	E	H	A	C	D	B
u7	I	H	F	E	G	B	C	D	A	I
u8	A	C	D	H	I	F	G	E	B	A

Квантовий генетичний алгоритм у цьому випадку перевершує класичну версію за допомогою інтерференційного кросинговеру, який забезпечує більшу кількість хромосом на вибір при виборі тих, хто вижив для наступного покоління. Подальше моделювання буде проводитися з багатьма різними мережами більшого розміру. Цікавою є можливість відслідковувати ефекти зміни розміру популяції, частоти мутацій, частоти кросинговерів та імовірності придатності рангу для обох алгоритмів. Подальша робота також необхідна для методів класифікації та перехресних перешкод.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ ДЛЯ ЗАДАЧ ОПТИМІЗАЦІЇ НА БАЗІ КВАНТОВОГО ГЕНЕТИЧНОГО АЛГОРИТМУ

2.1 Огляд алгоритмів пошуку

При вирішенні оптимізаційних завдань проектування ефективно використовуються стратегії, концепції, методи, механізми еволюційного моделювання біонічного пошуку та методів, інспірованих природними системами. Біоніка – рішення інженерних та технічних задач на основі вивчення структури та життєдіяльності живих організмів. Біонічний пошук з погляду перетворення інформації під час вирішення завдань на графах – це послідовне перетворення однієї кінцевої нечіткої множини альтернативних рішень на інше. Саме перетворення називається алгоритмом пошуку, основою якого можна використовувати генетичні, квантові, мурашині, бджолині та інші алгоритми. В основі ГА лежать алгоритми, що ефективно використовують інформацію, накопичену в процесі еволюції, для отримання квазіоптимальних та оптимальних рішень, про це ми вже дізнались з першого розділу.

Використання ідей квантової механіки дозволяє під час пошуку рішень на завданнях оптимізації використовувати підходи паралельних обчислень. Згідно з відомим принципом суперпозиції система може одночасно перебувати у всіх можливих станах. Виробляючи над одним станом довільні дії, ми робимо це одночасно над заданою безліччю станів. При вирішенні оптимізаційних завдань на графах пропонується нова технологія на основі спільного біонічного, квантового та рійового пошуку. Це дозволяє розширити область пошуку рішень без збільшення часу роботи та скоротити передчасну збіжність алгоритмів, підвищити ефективність та якість одержуваних рішень.[36]

2.1.1 Квантові алгоритми. Квантовий пошук (КП) допомагає знаходити нові підходи до вирішення *NP*-повних задач. КП працює таким чином: нехай задана функція *f(x)*, аргументи *x* – цілі числа, *x=1,2,….,N*, при чому *f(x)* приймає значення нуль у всіх випадках, крім *x=w*. Необхідно знайти значення *w*, використовуючи найменше число звернень до *f(x)*. Задачі такого типу при невеликому *x<100*вирішуються на основі повного перебору або методом спроб і помилок.

Ідею та структуру квантового алгоритму (КА) запропонував Л. Гровер. При вирішенні неструктурованої проблеми пошуку існує «оракул», який визначає чи є рішення яке ми розглядаємо шуканим. Л. Гровер розглядає *N* цілих чисел індексу *x = 1,2, ..., N* як набір ортогональних векторів *x = 1,2,..., k* у *N*-мірному просторі Гілберта. Цей крок алгоритму ставить у відповідність до кожного можливого індексу унікальний власний вектор. Спочатку готується простір, тобто квантовий регістр пам’яті, що містить певну кількість суперпозицій, що дорівнює кількості всіх *N*.[37,38].

$$\vec{S} = \frac{1}{\sqrt{N}} \sum_{x=1}^{N} \vec{x}\,, \tag{2.1}$$

Для реалізації пошуку цей квантовий простір розвивається у загальну суперпозицію, концентруючись у векторі, що визначає шлях до мети пошуку. Таким чином виконується процедура квантового кругообігу.

Для вирішення *NP*-повних задач на графах необхідно аналізувати структуру графіка, щоб «вирощувати» повні рішення, рекурсивно розширюючи послідовні рішення, що ми знаходимо при частковомузастосуванні алгоритму. Наведу модифікований алгоритм квантового пошуку:

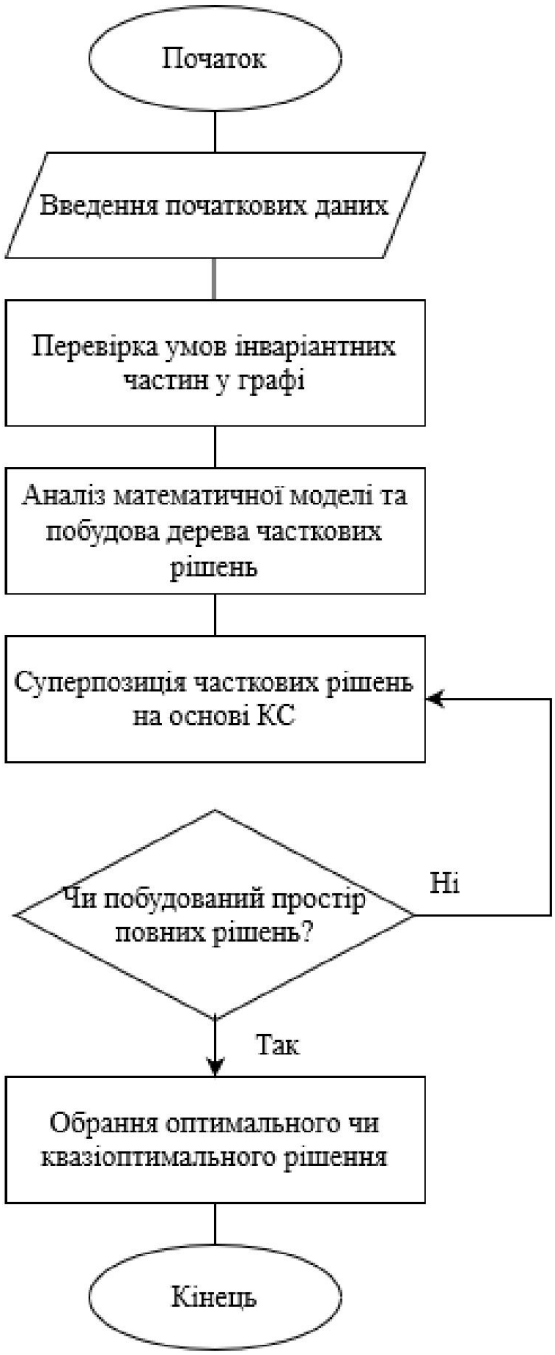


Рисунок 2.1 – Блок-схема квантового пошуку

Алгоритми квантового пошуку дуже чутливі до змін та перестановок параметрів вихідної моделі. Наприклад, для одного виду моделі об'єкта, представленого матрицею, можна отримати рішення з одним локальним оптимумом. Для цієї ж матриці з переставленими рядками та стовпцями можна отримати інше рішення з найкращим локальним оптимумом. Слід зазначити, що, змінюючи параметри, алгоритми та схему квантового пошуку,

у деяких випадках можна виходити з локальних оптимумів. Ця проблема продовжує залишатися однією з найважливіших у всіх методах оптимізації.

2.1.2 Бджолиний алгоритм. З назви можна здогадатись, що цей алгоритм моделює поведінку бджіл у природному середовищі. Ідея бджолиного алгоритму полягає в тому, що всі бджоли на кожному кроці обирають ділянки для дослідження, ці ділянки можуть бути елітними та на їх околицях. Таким чином ми урізноманітнюємо популяцію рішень на наступних ітераціях, та збільшуємо ймовірність виявлення рішень близьких до оптимальних [39].

Щоб краще розуміти базові поняття бджолиного алгоритму наведемо асоціації та терміни: джерело нектару – квітка, ділянка; фуражири – робочі бджоли; бджоли-розвідники. Джерело нектару характеризується значимістю, яка визначається різними параметрами. Фуражири закріплені за джерелами нектару. Кількість всіх бджіл у цих ділянках більша, ніж на інших. Середня кількість розвідників у рої становить 5-10%. Повернувшись у вулик, бджоли «обмінюються інформацією» та певній відзначеній для цього прощі (майданчику). Якщо розвідники знайшли найкращі джерела нектару, то за ними можуть бути закріплені фуражири. Умови зупинки алгоритму визначається користувачем і залежить від часу отримання результату.

Рисунок 2.2 – Опис бджолиного алгоритму пошуку

Таким чином, ключовою операцією алгоритму бджіл є спільне дослідження перспективних областей та їх околиць (периферій). Наприкінці роботи алгоритму популяція рішень складатиметься з двох частин: бджоли з найкращими значеннями цільової функції елітних ділянок, а також групи робочих бджіл із випадковими значеннями цільової функції.

2.1.3 Алгоритм квантового пошуку на графах. Для вирішення *NP*-повних задач на графах пропонується аналізувати структуру графа, щоб «вирощувати» повні рішення, аналогічно з іншими алгоритмами пошуку рішень, але у завданнях на графах важливим є знаходження інваріантів. Інваріант графа $G=(X,U)$, де $|X|=n$, а $|U|=m$, це число, пов'язане з G, яке приймає одне й те саме значення на будь-якому графі, ізоморфному G. Очевидно, що число вершин n та ребер m є найпростішими інваріантами графа. Наочніше показано на рисунку 2.3 визначення числа кліка з урахуванням квантового алгоритму. Клікове число$\omega(G)$ графу G – це кількість вершин в максимумі клік в G. Клік – це повний підграф, що містить найбільшу кількість ребер. Відповідно число повноти – це найбільша кількість вершин у кліку.

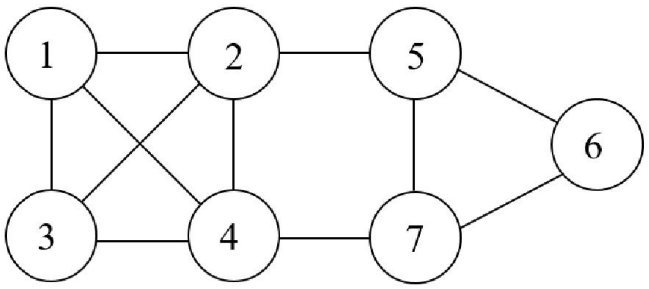


Рисунок 2.3 – Граф G

При визначенні клік графа знайдемо часткові рішення для кожної з вершин, таким чином розширимо «хороші» показники та усунимо тупікові. На першому кроці квантового пошуку для вершини 1 отримаємо такі результати:

Таблиця 2.1 – Результати першого кроку.

1 рівень	{1,2}, {1,3}, {1,4}
2 рівень	{1,2,3}, {1,2,4}, {1,3,4}

3 рівень	{1,2,3,4}

В результаті після суперпозиції часткових рішень отримаємо кліку *Q1 = {1, 2, 3, 4}, |Q1|=4*. На другому кроці для вершини два отримаємо: *{2, 5}, Q2 ={2,5}, |Q2|=2.* Для третьої вершини часткових розв'язків немає. Для четвертої вершини маємо *{4, 7}, Q3 = {4, 7}, | Q3 | = 2.*

Для п'ятої вершини отримаємо такі часткові рішення:

Таблиця 2.2 – Результати дл п’ятої вершини.

1 рівень	{5,6}, {5,7}
2 рівень	{5,6,7}

Після суперпозиції часткових рішень отримаємо кліку *Q4 = {5,6,7}, | Q4 | = 3.*

Отже, для графа *G* (рисунок 2.3) побудовано сімейство клік *Q = {Q1, Q2, Q3, Q4}*. Число повноти графа дорівнює 4.

Розглянемо рішення розмальовки графа з урахуванням квантового пошуку. Розфарбуванням графа $G_2=(X,U)$ називається розбиття графа на такі непересічні підмножини вершин $X_1 \cap X_2 \cap \ldots \cap X_e = \emptyset$, $X_1 \cup X_2 \cup \ldots \cup X_e = X$, що вершини всередині кожного підмножини несуміжні. Найменше число підмножин X_i під час розмальовки називається хроматичним числом графа.

Алгоритм базується на знаходженні часткового забарвлення для підмножини вершин. Для визначення повної розмальовки графа проводиться рекурсивне розширення часткових забарвлень із поверненням у разі тупікових рішень. Елементарний спосіб полягає у розгляді дерева пошуку часткових рішень заданої глибини. Наприклад, нехай заданий граф $G_2=(X,U)$, де $|X|=7$.

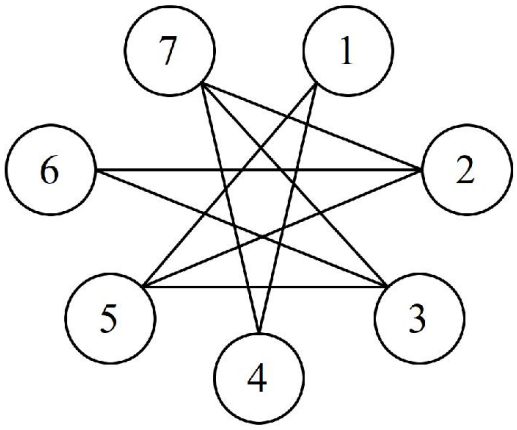


Рисунок 2.4 – Граф G_2

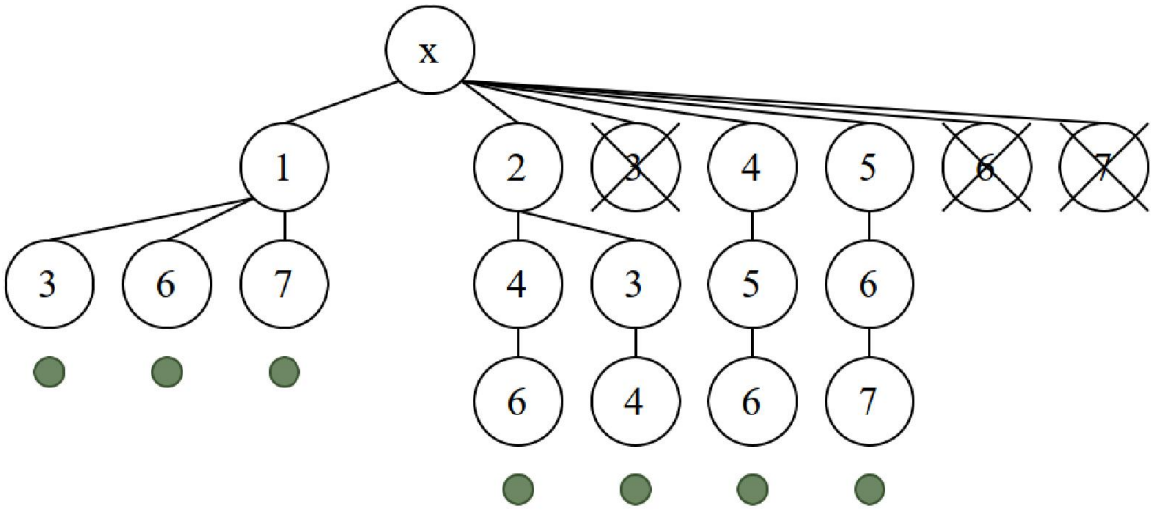


Рисунок 2.5 – Дерево забарвлення графа

Коренева вершина дерева містить усі вершини *X* графа *G*. На першому рівні дереварозташовуються всі вершини графа. Зазначимо, що можливе випадкове та впорядковане розташування вершин. Обираємо на першому рівні вершину «1» і визначаємо можливі часткові рішення з розфарбування, що включають цю вершину. Ідучи по дереву вниз, видно, що маємо три часткові рішення {1,3}, {1,6}, {1,7}. Подальше розширення цих часткових рішень неможливе. Це зазначено знаком «●». Переходимо до вершини «2» і будуємо два нових часткових рішення {2,4,6} та {2,3,4}. Вершина «3» часткових рішень не дає. Вершина «4» дає часткове рішення {4,5,6} та вершина «5» - {5,6,7}. Вершини «6» та «7» часткових рішень не дають. Для отримання альтернативних розфарбувань пропонується операція

суперпозиції. Суть операції в об'єднанні *n* часткових рішень в одне з винятком елементів, що повторюються. Це квантова властивість алгоритму пошуку на графах. Наприклад:

{1,2}**S**{3,4,5}**S**{1,3,7} = [{1,2}, {3,4,5}, {7}].

Тут **S** – знак суперпозиції. Наше завдання у тому щоб вибрати рішення з найменшою кількістю підмножин, що дозволить знайти забарвлення з найменшою кількістю кольорів.

Звідси випливає евристичне правило: за наявності кількох можливих альтернатив суперпозицію виконується у квантовому алгоритмі для підмножин, що мають найменшу кількість елементів, що збігаються.

Для вирішення оптимізаційних завдань останнім часом застосовуються жадібні алгоритми, які є різними модифікаціями алгоритмів динамічного програмування, але жадібні алгоритми простіші і швидші. Основою жадібного алгоритму є локально-оптимальний вибір на кожному кроці з прогнозом, що остаточне рішення буде оптимальним. Незважаючи на те, що у загальному випадку жадібний алгоритм може привести до локального мінімуму, для багатьох завдань ці алгоритми дають можливість отримувати оптимум шуканої цільової функції.

2.2 Архітектура суміщеного квантового та генетичного алгоритмів

Гібридизація алгоритмів пошуку рішень сприяє покрашенню їх швидкодії та ефективності, особливо коли мова йде про квантові алгоритми пошуку. Нижче наведена схема взаємодії квантових та генетичних алгоритмів. Очевидно, що блоки у даній схемі можна використовувати для побудови інших гібридизованих алгоритмів, нарощуючи їх ієрархічно. За допомогою цих блоків ми можемо побудувати схему послідовного та паралельного сумісного пошуку будь-якого рівня складності. Тут МА – мурав’їний алгоритм, КА – квантовий, ГА – генетичний, БА (ПА на схемі) – бджолиний алгоритм, РА – алгоритм роя частинок. На схемі справа видно,

що можливі різні випадки взаємодії блоків. Наприклад, спочатку виконується генетичний алгоритм, потім квантовий, мурав'їний та бджолиний.

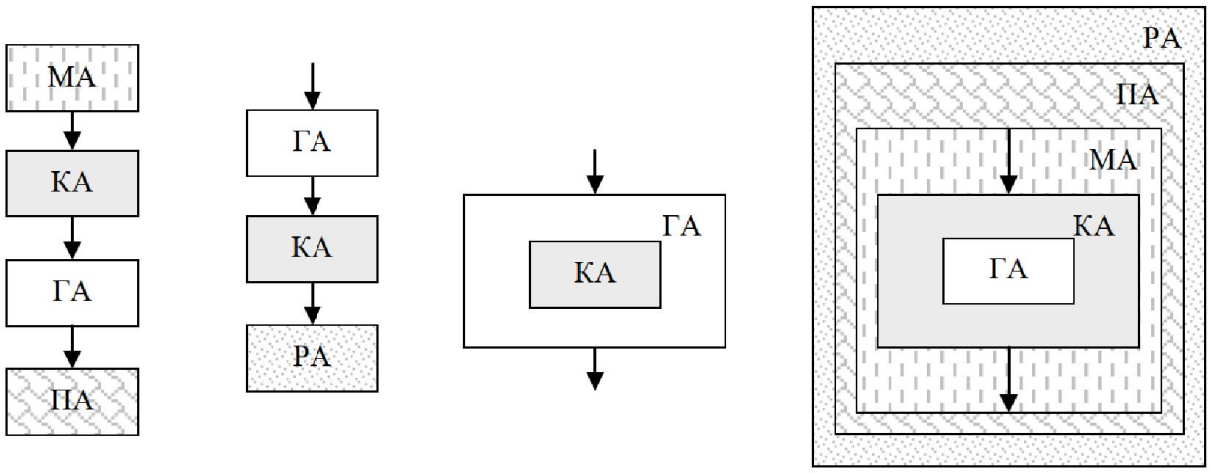


Рисунок 2.6 – Схеми взаємодії генетичних та квантових алгоритмів

Важливо зазначити, що квантовий пошук може прискорити класичний випадковий алгоритм, створюючи суперпозицію частотних рішень, збільшуючи простір пошуку шуканого рішення.

Пропонується модифікована схема спільного пошуку, що складається із трьох основних блоків (рисунок 2.7). Перший блок назвемо препроцесором. Тут виконується створення однієї або деякої множини початкових популяцій. Другий блок складається з *1,2,...,n* рівнів. Кожен рівень складається із чотирьох етапів: вибір подання рішення; розробка операторів випадкових, спрямованих та комбінованих змін; визначення законів виживання рішення; рекомбінації. Третій блок назвемо постпроцесором. Тут реалізуються принципи еволюційної адаптації до зовнішнього середовища (особі, яка приймає рішення) та самоорганізації. Зазначимо, що будівельні блоки спільного квантового та генетичного пошуку (рисунок 2.4) можуть ефективно працювати у складі процесора та постпроцесора. Перевага такої архітектури спільного пошуку полягає в тому, що в ній усі рівні пов'язані з рівнем зовнішнього середовища та можуть спілкуватися між собою.[39]

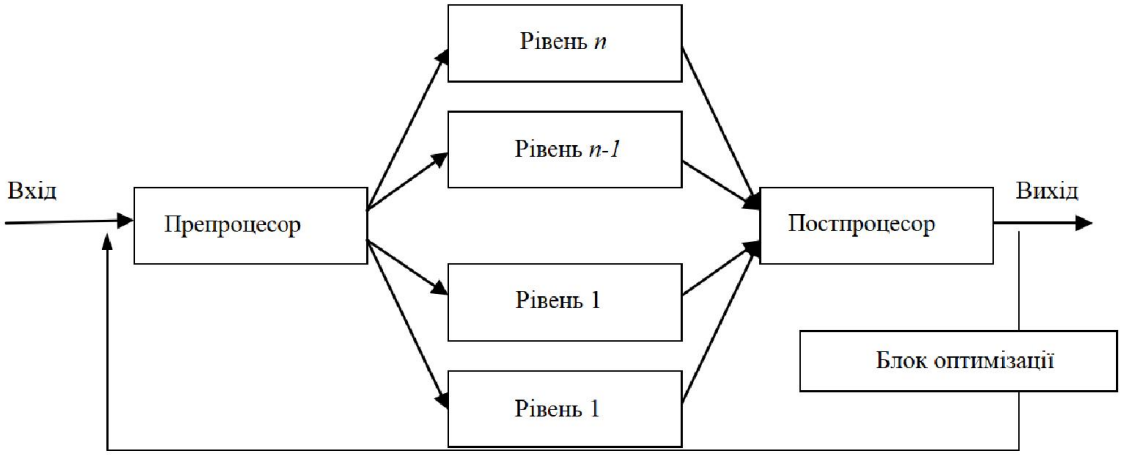


Рисунок 2.7 – Горизонтальна архітектура сумісного пошуку

Для управління та реалізації процесором сумісного пошуку використовують наступні принципи:

1. Принцип цілісності (синергики). У квантових та генетичних алгоритмах значення цільової функції альтернативного рішення не зводиться до суми цільових функцій часткових рішень;

2. Принцип чутливості до початкових умов. Результат роботи квантових та генетичних алгоритмів залежить від представлення вхідних даних досліджуваної моделі;

3. Принцип додатковості. При вирішенні оптимізаційних завдань виникає необхідність використання різних несумісних та взаємодоповнюючих моделей еволюції та вихідних об'єктів;

4. Принцип неточності. При зростанні складності аналізованого завдання зменшується можливість побудови точної моделі;

5. Принцип управління невизначеністю. Необхідно вводити різні види невизначеності в квантові та генетичні алгоритми;

6. Принцип відповідності. Мова опису вихідної задачі повинна відповідати наявності інформації, що є про неї;

7. Принцип «007». Необхідно використовувати лише ті вхідні дані, які необхідні для вирішення завдання;

8. Принцип єдності та протилежності порядку та хаосу. «Хаос як руйнівний, а й конструктивний», іншими словами у хаосі області допустимих рішень обов'язково міститься порядок, що визначає шукане рішення;

9. Принцип ієрархії. Квантові та генетичні алгоритми можуть підлаштовуватися зверху вниз та знизу вгору;

10. Принцип гомеостазу. Квантові та генетичні алгоритми конструюються таким чином, щоб будь-яке отримане альтернативне рішення не виходило з області допустимих.

2.3 Квантовий генетичний алгоритм на основі кутріту

Кутріт – це структура з кількома, більш ніж двома, станами, які також можуть використовуватися для кодування хромосоми. По суті, кутріт – це одиниця квантової інформації, яка може бути в будь-якому з *n* станів або в будь-якій їх суперпозиції. Потрійна квантова логіка є найпростішим типом багатозначної логіки. Основна одиниця пам'яті має три базові стани: $|0\rangle$, $|1\rangle$, $|2\rangle$. Їх можна представити у вигляді лінійної суперпозиції: $|q\rangle = \alpha|0\rangle + \beta|1\rangle + \gamma|2\rangle$. Матричне представлення виглядає наступним чином:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad |2\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \quad |q\rangle = \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix}. \tag{2.2}$$

Система, що містить *N* кутріт, має 3^N базових станів (на відміну від 2^N станів для двійкової логіки). При переході від двійкової до багатозначної логіки ми отримуємо експоненціальне збільшення кількості базових станів, що призводить до збільшення продуктивності алгоритму при тій же точності пошуку. В основі такого підходу лежить поняття кубіта, проте ця форма представлення інформації більш вдосконалена. Дослідження вже показали, що використання квантової логіки із кількома, більше двох, основними

станами є більш перспективним точки зору обчислювальної потужності та ефективності роботи алгоритму. Довжина квантової хромосоми визначається бажаною точністю пошуку ε, областю пошуку [x_{min}, x_{max}] і кількістю базових станів квантової системи:

$$N = \log_n \left(\frac{x_{\max} - x_{\min}}{\varepsilon} + 1 \right) \tag{2.3}$$

де n – число базових станів.

Якщо $n = 2$, точність пошуку $\varepsilon = 10^{-6}$, а область пошуку [-1,1], довжина квантової хромосоми повинна бути не менше 21 кубіт, тоді як при $n = 3$ потрібно лише 14 кубітів.

Класична представлення рішення задачі має місце у векторі кінцевого стану $|\psi\rangle$і може бути отримана в результаті дослідження квантового стану. Кутріт в одному з базових станів, отриманий в результаті квантового спостереження, є класичним уявленням про квантову хромосому. Для спостереження цього стану квантової хромосоми ми використали алгоритм приведений нижче у вигляді псевдокоду:

```
(1) for i in 1,...,N do
(2)   r ← random number in range [0, 1]
(3)   if r < ⌈a_i⌉^2 then
(4)     p ← 0
(5)   else if r < ⌈a_i⌉^2 + ⌈β_i⌉^2 then
(6)     p ← 1
(7)   else
(8)     p ← 2
(9)   end if
(10) end for
```

Рисунок 2.8– Реалізація спостереження за станом кутріту у вигляді псевдокоду

Алгоритм призводить до того, що квантова хромосома перетворюється в її класичне представлення в потрійній системі нумерації. Таким чином

стани p (0, 1 чи 2) мають ймовірності перебування кутріта у заданих станах α_i^2, β_i^2, та γ_i^2 відповідно.

Початковий стан квантової системи не несе інформації про задачу та алгоритм її розв'язку, тому на початку всі кутріти перебувають у стані, що являє собою суперпозицію базових із однаковою ймовірністю. Таким чином алгоритм призводить до появи квантової хромосоми, що трансформується в його класичне представлення в потрійній системі нумерації:

α_1	α_2	α_3	α_4	α_5	$\cdots$	$\cdots$	$\cdots$	α_N
β_1	β_2	β_3	β_4	β_5	$\cdots$	$\cdots$	$\cdots$	β_N
γ_1	γ_2	γ_3	γ_4	γ_5	$\cdots$	$\cdots$	$\cdots$	γ_N
			Спостереження					
↓	↓	↓	↓	↓	↓	↓	↓	↓
0	2	0	1	1	2	1	0	0

Рисунок 2.9 – Класичне представлення алгоритму в потрійній системі нумерації

Вся інформація про задачу та алгоритм її вирішення міститься в квантовому вентилі (гейті), тому його алгоритм є ключовим питанням побудови будь-якого квантово генетичного алгоритму. Вертикальні лінії задають амплітуди ймовірності α, β і γ. Якщо класичне представлення кращого індивіда популяції має, наприклад, 2 на позиції i, усі амплітуди, крім γ, помножуються на коефіцієнт μ. Його значення знаходиться в діапазоні від 0 до 1 і визначається емпірично. В результаті збільшиться лише амплітуда ймовірності γ, яка відповідає популяції найкращої особини на попередньому етапі еволюції. При цьому амплітуди ймовірностей α і β зменшаться.

Псевдокод для генетичного оператора, що оновлює стан кутріту (квантових воріт еволюції системи) в результаті модифікації амплітуд ймовірностей α_i, β_i і γ_i, представлений нижче.

(1) **for** $k \in 1,\ldots,N$ **do**
(2) $d = 0$
(3) **for** $j \in 1,\ldots,N$ **do**
(4) $d = \left(b_j^k - x_j^k\right)^2$
(5) $d = \sqrt{d}$
(6) **if** $d < R$ **then** *традиційний оператор квантового гейту*
(7) **end for**

Рисунок 2.10 – Псевдокод для генетичного оператора, що оновлює стан кутріту

Важливо, що описаний алгоритм квантових воріт усуває необхідність використання таблиці пошуку, яка існує в традиційному КГА. Ініціалізована сукупність *(t = 0)* включає всі можливі рішення з однаковою ймовірністю. Це означає, що процес КГА починається з випадкового пошуку. Тому ми можемо спостерігати що із розвитком еволюції в часі розподіл зазнає змін, це відображено на графіку нижче. Звичайно, процес алгоритму починається з випадкового пошуку.[40]

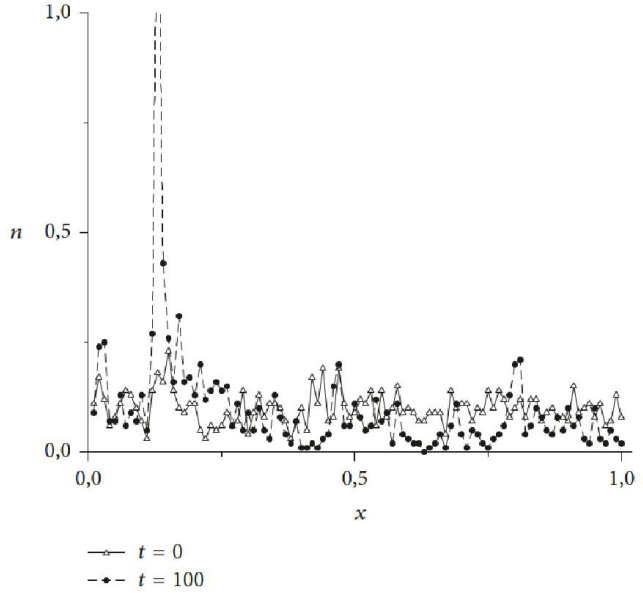


Рисунок 2.11 – Показники середнього розподілу особин популяції n по області пошуку на різних етапах еволюції на прикладі 100 спроб.

Механізми локального пошуку з t>100 проілюстровані на рисунку 2.12.На графіку продемонстрована роль параметра μ.Його фіксоване значення погано впливає на швидкість конвергенції.Тому бажано використовувати адаптивний квантовий оператор.

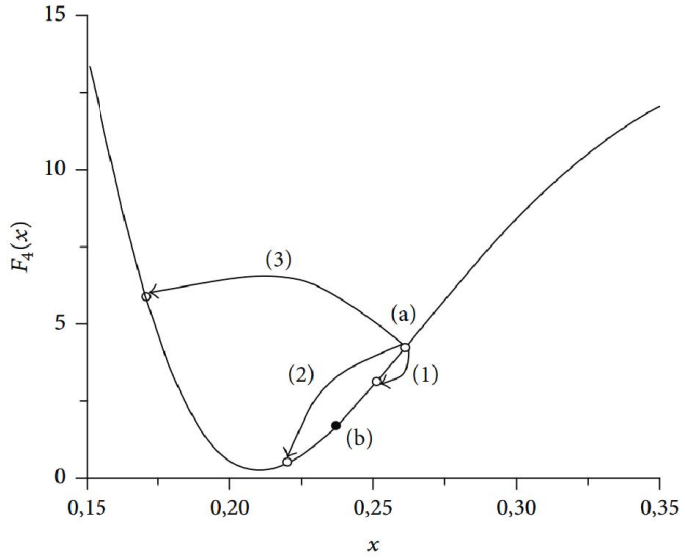


Рисунок 2.12 – Результати застосування оператора квантового гейту.

Тут (б) - найкраща особина популяції;(а) поточна особа.

Перераховані дані показують, що оператор може впливати на хромосому трьома можливими способами:

1. Фітнес функція особистості покращується, але не стає найкращою;

2. Фітнес функція особини особистості покращується, і вона стає найкращою;

3. Фітнес функція погіршується.

Таким чином, ми можемо зробити висновок, що чим більша кількість особин зазначена в області найкращих b, тим ефективніший процес локальної конвергенції. Підводячи підсумок, квантовий генетичний алгоритм починає свою роботу з глобального пошуку і автоматично перемикається на локальний через зміну структурних характеристик розподілу особин популяції. [41]

РОЗДІЛ 3
ПРАКТИЧНА ЧАСТИНА РОЗРОБКИ НА БАЗІ КВАНТОВОГО ГЕНЕТИЧНОГО АЛГОРИТМУ

3.1 Огляд програмного інструментарію*Matlab*та програмна реалізація алгоритму

Однією з новацій Matlab 7.0 є тулбокс *Genetic Algorithm and Direct Search Toolbox*, який призначений для розширення функціональних можливостей пакета, зокрема*Global Optimization Toolbox (GOT)*, генетичними алгоритмами. Працювати з генетичними алгоритмами тепер можна у двох тулбоксах.*GOT* забезпечує функції, які шукають глобальні розв'язання проблем, що містять кілька максимумів чи мінімумів. Кейси тулбоксу включають пошук шаблону, генетичний алгоритм, рій частинок, мультизапуск та глобальний пошук. Можна використовувати ці засоби для завдань оптимізації, де функція мети або обмеження є безперервною, переривчастою, стохастичною, не має похідних або включає функції чорної скриньки чи симуляції. У *Matlab* також реалізована можливість графічно відображати результати досліджень у вигляді графіків та таблиць. Тому, це середовище прекрасно підходить для реалізації нашого алгоритму оптимізації пошуку оптимального шляху, завдяки інструментам, які вбудовані у це середовище ми зможемо наочно оцінити роботу алгоритму.

Механізм роботи з генетичними алгоритмами реалізований двома способами:

1. Виклик функції генетичних алгоритмів;
2. Використання комплекту *Genetic Algorithm Tool.*

Обидва способи застосовуються черезстандартний набір функцій та модулів *Matlab*. На мій погляд, обидва способи ідентичні з тією різницею, що використовуючи панель *Genethic Algorithm Tool*, будь-які параметри

генетичного алгоритму налаштовуються з використанням графічної оболонки.

Використана бібліотека дозволяє конструювати генетичні алгоритми специфічного вигляду та виконувати обчислення у режимі кількох потоків команд. Структура бібліотеки виділяє три великих блоки класів: класи для конструювання алгоритмів, класи хромосом та класи популяції. До класу хромосоми включаються структури, що реалізують операції кросоверу і мутації, порівняльний оператор для фітнес-функцій та власне визначення вигляду хромосоми.

Клас популяції являє сукупність структур відбору, вибору особин для схрещування, формування нової популяції на наступному етапі еволюції. Клас алгоритму включає критерій виходу з еволюційного процесу та власне алгоритм. Бібліотека включає не лише інтерфейси для конструювання алгоритму. Реалізовано також часто вживані структури, такі як, наприклад, бінарна хромосома для класичного генетичного алгоритму (*Binary-codedGA*) та хромосома для генетичного алгоритму на дійсних числах (*Real-codedGA*); реалізовано ряд шаблонів для операцій кросоверу, мутації, шаблони поведінки, визначення часу, виходу з ітераційного процесу, селекції та формування популяції.

Matlab використовується дослідниками для написання генетичних алгоритмів, оскільки дає можливість імпортувати дані у файли .xls, CSV-файли тощо. Він має потужні вбудовані інструменти побудови графіків, які дозволяють легко візуалізувати дані. Це один з найкращих інструментів для генетичних алгоритмів. Якщо говорити про набори інструментів у MATLAB, одним з найпопулярніших наборів інструментів генетичного та еволюційного алгоритму є *GEATbx*. Він забезпечує глобальні можливості оптимізації для вирішення проблем, непридатних для традиційних підходів до оптимізації. Це також дозволяє вирішувати великі та складні проблеми з великою легкістю, забезпечуючи при цьому візуалізацію, багатоцільову оптимізацію, обробку обмежень тощо.

Розглянемо вбудовані функції до роботи з генетичними алгоритмами:

– *[x fval reason output population scores] = ga(@fitnessfun, nvars, options)* – функція знаходження мінімуму цільової функції;

Вхідні параметри: *@fitnessfun* – покажчик функції в *M*-файлі, за якою провадиться розрахунок функції пристосованості; *nvars* – число незалежних змінних функції пристосованості; *options* – параметри генетичного алгоритму, що налаштовуються;

Вихідні параметри: *x* – кінцева точка розрахунку; *fval* – значення функції пристосованості у точці *x*; *reason* – причина зупинення алгоритму; *output* – структура з інформацією щодо ефективності роботи алгоритму для кожного покоління; *population* – стан останнього сімейства; *scores* – кінцевий стан;

– val = gaoptimget(options, 'Name') – повертає параметри використовуваного генетичного алгоритму;

Вхідні параметри: *options* – структура із параметрами генетичного алгоритму; *'Name'*– ім'я параметра, значення якого потрібно взяти;

Вихідні параметри: *val* – значення запитуваного параметра;

– options = gaoptimset('param1', value1, 'param2', value2, ...) – встановлює параметри генетичного алгоритму;

Вхідні параметри: *'param1'* – ім'я параметра; *value1* – значення, що встановлюється;

Вихідні параметри: *options* – структура із параметрами генетичного алгоритму.

3.1.1 Реалізація роботи ГА для вирішення задачі оптимізації пошуку оптимального шляху. Основна концепція ГА призначена для імітації процесів у природній системі, особливо тих, що слідують принципам, вперше викладеним Чарльзом Дарвіном. Генетичний алгоритм відстежує набір P можливих рішень, що називається популяцією. У кожній ітерації поточна сукупність P_i замінюється на наступну P^{i+1};

1. Алгоритм керується простим циклом, як показано нижче:

1.1. [Початок] Створення випадкової сукупності *n* хромосом (доцільні рішення для проблеми);

1.2. [Фітнес функція] Оцінка придатності *f(x)* кожної хромосоми *x* у популяції;

1.3. [Нова популяція] Створення нової популяції, повторюючи наступні кроки до остаточного створення нової популяції:

1.3.1. [Вибір] Обрати дві батьківські хромосоми з популяції відповідно до їх:

1.3.1.1. фітнес функції (чим вона краща, тим більший шанс бути обраним);

1.3.2. [Кросовер] Із заданою ймовірністю провести операцію кросовера над батьківськими хромосомами, щоб сформувати нове потомство (нащадків). Якщо кросовер не проводився, потомство є точною копією батьків;

1.3.3. [Мутація] Зі вказаною ймовірністю мутуйте нове потомство в кожному локусі (положенні в хромосомі);

1.3.4. [Прийняття] Помістити нове потомство в нову популяцію;

1.4. [Заміна] Використати нову генеровану сукупність для подальшого запуску алгоритму;

1.5. [Тест] Якщо кінцева умова виконана, зупинитися та повернути найкраще рішення в поточній популяції;

1.6. [Зациклення] Перейти до кроку 2;

2. Для рішення задачі, яку ми перед собою поставили необхідно ввести деякі обмеження;

3. Обмеження 1: кожна вершина має сполучатись через пару ребер з рештою вершин, тобто, через вхідне та вихідне ребро:

$$\forall i \in V, \sum_{j \in V \setminus \{i\}} x_{ij} = 2 \qquad (3.1)$$

В сумі кожний доданок x_{ij} дорівнює або 1 (належить маршруту) або 0 (не належить). Тобто, отримана сума дорівнює кількості ребер в маршруті, таких що мають вершину i на одному з кінців. Вона дорівнює 2, оскільки кожна вершина має вхідне та вихідне ребро.

Обмеження 2:кожна множина вершин $S \subset V$, є або порожньою, або містить всі вершини, що сполучається з рештою вершин через щонайменше два ребра:

$$\sum_{i \in S,\ j \notin S} x_{ij} \geq 2 \qquad (3.2)$$

Для всіх множин вершин S де $1 \leq |S| \leq |V| - 1$. Ця сума дорівнює сумі ваг ребер маршруту між вершиною $i \in S$ та вершиною $j \notin S$. Аби усунути зайві нерівності, можна обмежитись множинами вершин S з щонайменше двома та щонайбільше $|V| - 2$ вершинами.

Обмеження 3:кожну вершину можна відвідати лише один раз.

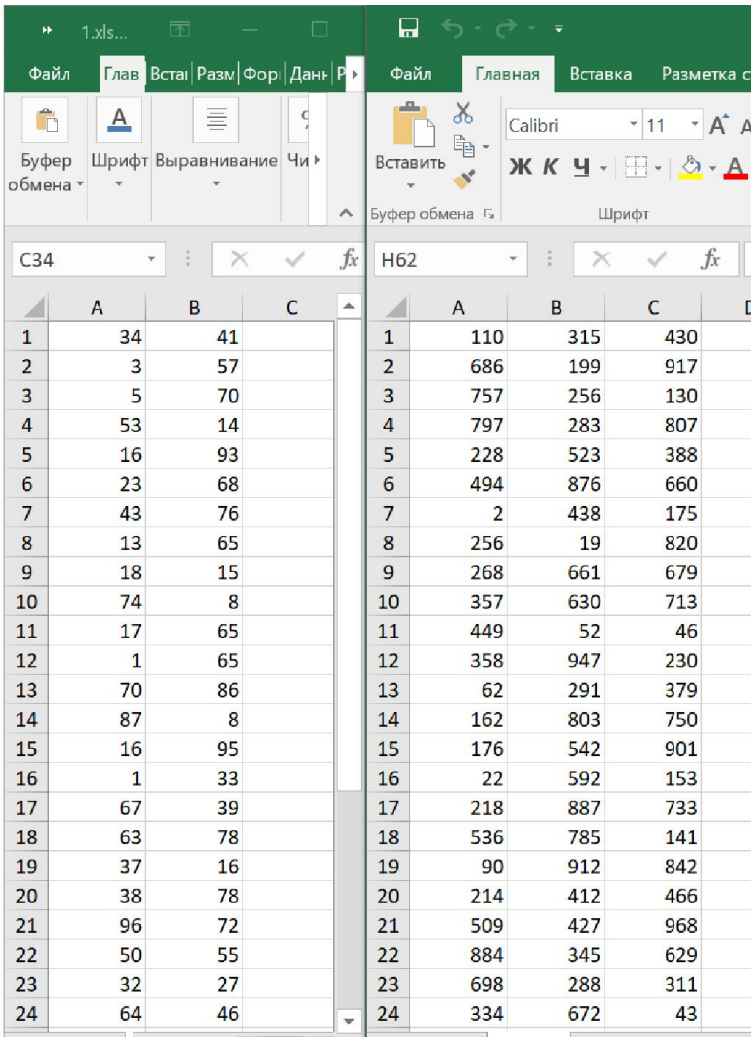
Фітнес-функція виглядає наступним чином. Агенту необхідно обійти n місць і повернутися у вихідний пункт відвідуючи кожне місто лише по одному разу та вибрати маршрут з мінімальною вартістю.

Дано: повнозв'язний граф $G = (V, U)$, де V –множина вершин графу, $|V|=n$–кількість точок; U–множина ребер (характеризує шляхи між містами); матриця суміжності $R(i, j)$, де $i, j = 1,2, ..., n$, що характеризує відстані між точками. Цільова функція для задачі має наступний вигляд:

$$Z(\varphi) = R(\varphi[n], \varphi[1]) + \sum_{i=1}^{n-1} R(\varphi[i], \varphi[i+1]) \to \min \qquad (3.3)$$

3.1.2 Опис алгоритму. Для пошуку оптимального маршруту задачі використано генетичний алгоритм з критерієм виходу за незмінною впродовж деякої кількості епох найкращою хромосомою. Критерій відбору хромосом –менша фітнес функція. Хромосома конструюється специфічним чином і являє собою маршрут, координати якого вказані у exel-файлах, прикріплених до роботи. Для того, щоб уникнути некоректних маршрутів, що виникають із застосуванням класичних одноточкових чи багатоточкових кросоверів, що може суттєво вплинути на швидкість роботи алгоритму, використовується специфічно створений кросовер. Із хромосоми одного з батьків, вибирається випадковим чином місто і далі в хромосомах батьків вибирається те, яке є найближчим серед міст, пов’язаних з поточним. Якщо таке місто вже присутнє у маршруті-нащадку, то вибирається наступне, з найменшою відстанню. Якщо всі пов’язані міста вже присутні у маршруті, то вибирається випадкове місто, із тих, що до маршруту-нащадку ще не увійшли. Алгоритм використовує оператор мутації, що міняє місцями міста у маршруті. Розмір популяції 100 осіб. Вісім найкращих у кожній епосі потрапляють до наступної без операції відбору.

У якості вхідних даних було створено *exel*-файли з координатами п’ятдесяти точок, згідно яких будується маршрут. Для візуалізації океанічної поверхні було побудовано двомірну модель координатних площин з осями *x* та *y* (рисунок 3.1), для візуалізації глибини та поверхні використано тримірну модель (*x, y, z* на рисунок 3.1).



	A	B	C
1	34	41	
2	3	57	
3	5	70	
4	53	14	
5	16	93	
6	23	68	
7	43	76	
8	13	65	
9	18	15	
10	74	8	
11	17	65	
12	1	65	
13	70	86	
14	87	8	
15	16	95	
16	1	33	
17	67	39	
18	63	78	
19	37	16	
20	38	78	
21	96	72	
22	50	55	
23	32	27	
24	64	46	

	A	B	C
1	110	315	430
2	686	199	917
3	757	256	130
4	797	283	807
5	228	523	388
6	494	876	660
7	2	438	175
8	256	19	820
9	268	661	679
10	357	630	713
11	449	52	46
12	358	947	230
13	62	291	379
14	162	803	750
15	176	542	901
16	22	592	153
17	218	887	733
18	536	785	141
19	90	912	842
20	214	412	466
21	509	427	968
22	884	345	629
23	698	288	311
24	334	672	43

Рисунок 3.1 – Вхідні дані. З ліва показана таблиця для побудови точок на осях x, y, а з права – x, y, z.

3.1.3 Результати виконання програми. Результатами роботи програми є прокладений маршрут та оцінена його вартість на графіку *Total Cost*, інші результати подаються у вигляді графіків матриці дистанції *(Distance Matrix)*, фітнес функції *(Best Solution History)*, початкового розположення на площині точок *(Location)* та побудований маршрут з прорахованою дистанцією *(Total Distance)*.

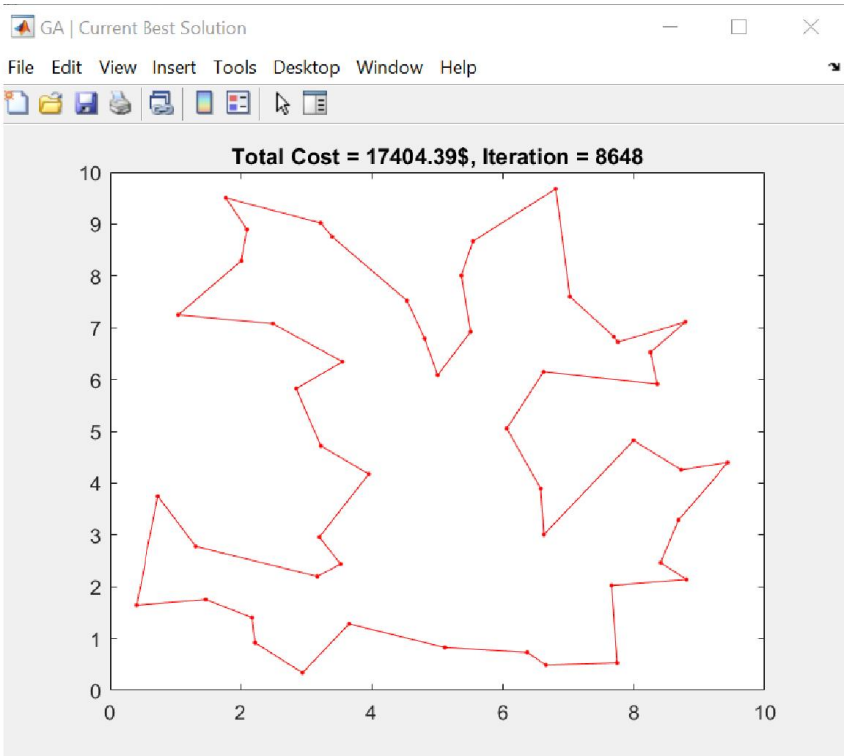


Рисунок 3.2 – Графік Total Cost

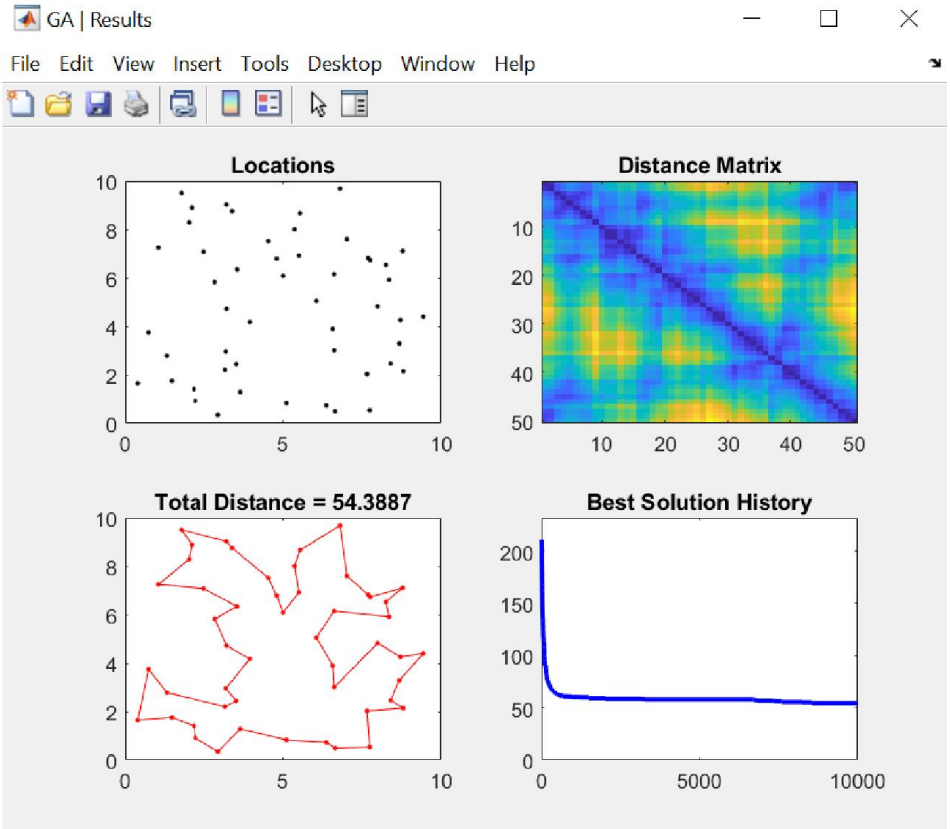


Рисунок 3.3 – Графіки Distance Matrix, Best Solution History, Location, Total Distance

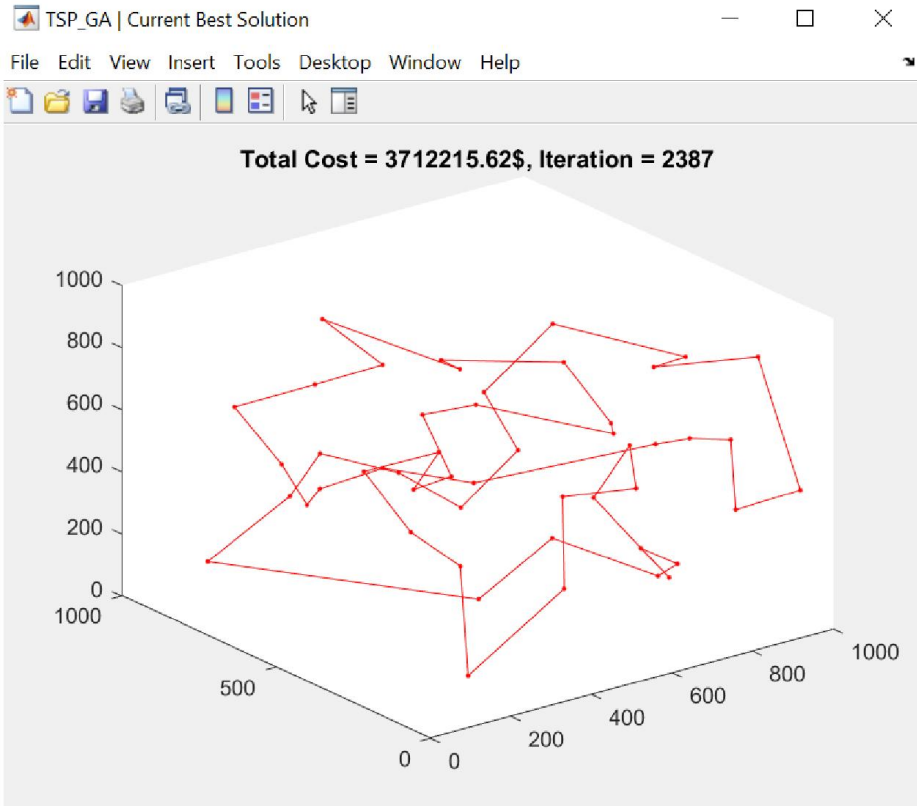


Рисунок 3.4 – Total Cost (3D)

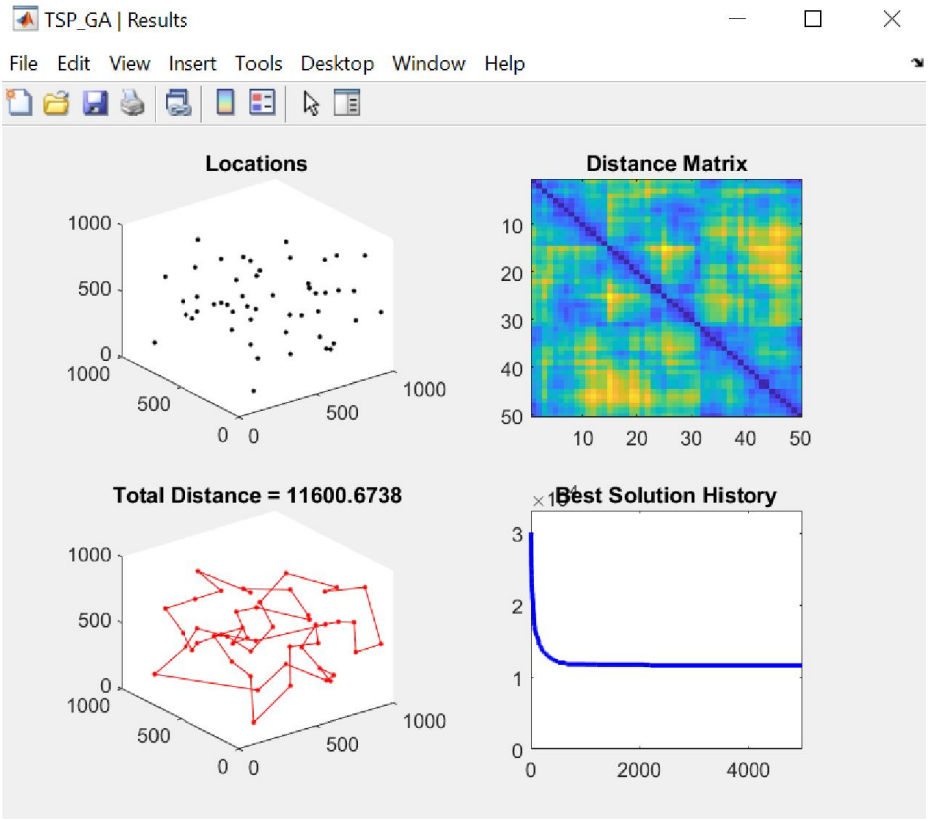


Рисунок 3.5 – Графіки Distance Matrix, Best Solution History, Location, Total Distance (3D)

3.2 Огляд програмного інструментарію*Python* та розробка ГА

Python це один з найбільш універсальних інструментів для генетичного програмування, що може похвалитися безліччю цікавих бібліотек для генетичних алгоритмів, що мають гідні можливості побудови графіків. Для роботи з генетичними алгоритмами створено цілу низку каркасів на *Python*, наприклад *GAFT, Pyevolve і PyGMO*. Але, розглянувши різні варіанти, я хочу зупинитися на каркасі *deap*, оскільки він простий у користуванні і пропонує широкий набір функцій, підтримує розширюваність та має докладну документацію. *Deap* (скорочення від *Distributed Evolutionary Algorithms in Python* – розподілені еволюційні алгоритми на *Python*) підтримує швидку розробку рішень із застосуванням генетичних алгоритмів та інших методів еволюційних обчислень. Також пропонує різні структури даних та інструменти, необхідні для реалізації різних рішень на основі генетичних алгоритмів.

Каркас *DEAP* включає кілька вбудованих еволюційних алгоритмів, що знаходяться в модулі *algorithms*. Один з них, *eaSimple*, реалізує загальну структуру генетичного алгоритму і може замінити більшу частину написаного нами коду в функції *main*. Для збору та друку статистики можна використовувати інші корисні об'єкти *DEAP, Statistics* та *logbook*.

3.2.1 Опис алгоритму та реалізація задачі логістики на *Python.* Завдання комівояжера перегукується з 1930 року і є однією з найбільш вивчених завдань оптимізації. Найчастіше воно використовується для оцінки алгоритмів оптимізації. У цієї задачі багато варіантів, але спочатку вона ставилася на прикладі комівояжера, якому потрібно об'їхати кілька міст: нехай дано список міст і відомі відстані між кожними двома містами. Знайти найкоротший шлях, що проходить через усі міста і повертається у вихідну точку.

У задачі комівояжера міста зазвичай представляються числами від 0 до *n - 1*, а можливі рішення - послідовностями таких чисел. Наприклад, у задачі з п'ятьма містами рішення мають вигляд [0, 1, 2, 3, 4], [2, 4, 3, 1, 0] і т. д. Кожне рішення можна оцінити, підсумувавши відстані між сусідніми в послідовності містами і додавши ще відстань між останнім та першим містом. Такий список буде використано для представлення хромосоми в генетичному алгоритмі. Клас *Python* зчитує вміст TSPLIB-файлу, де зберігаються дані про відстань від міст та їх кількість, і обчислює всі попарні відстані між містами. Крім того, він обчислює повну відстань у заданому рішенні.

Для інкапсуляції даних завдання комівояжера було написано на *Python* клас *TravelingSalesmanProblem*.

Цей клас надає такі закриті методи:

– *create_data()*: читає вказаний файл з інтернету, витягує координати міст, обчислює попарні відстані між містами і запам'ятовує їх у матриці відстаней (двовимірному масиві). Потім серіалізує розташування міст і обчислені відстані в файлі за допомогою вбудованої утиліти *pickle*;

– *read_data()*: читає серіалізовані дані, а якщо їх ще немає, викликає *create_data()* для їх підготовки.

Ці методи викликаються з конструктора, тому дані ініціалізуються в момент створення екземпляра. Крім того, клас надає такі відкриті методи:

– *getTotalDistance(indices):* обчислює повну довжину шляху, що проходить через міста з зазначеними індексами;

– *plotData(indices):* малює шлях, що проходить через міста з зазначеними індексами.

Метод *main* класу викликає перераховані вище методи. Спочатку він створює екземпляр завдання *bayg29* (29 міст у Баварії), потім обчислює довжину оптимального рішення.

Основні кроки:

1. Слід визначити стратегію пристосування. Ми хочемо мінімізувати стан, а це означає, що потрібно визначити мінімізуючий клас *Fitness* з однією метою, в якому задана одна негативна вага, у цьому нам допоможе оператор *creator.create*;

2. Вище вже зазначалось, що в ролі хромосоми для генетичного алгоритму виступає список цілих чисел (індексів міст) від 0 до *n – 1*, де n – кількість міст. Так, показане вище оптимальне рішення задачі *bayg29* представлене такою хромосомою: (0, 27, 5, 11, 8, 25, 2, 28, 4, 20, 1, 19, 9, 3, 14, 17, 13, 16, 21, 10,18, 24, 6, 22, 7 , 26, 15, 12, 23). Ділі цей крок можна розділити на кілька єтапів: по-перше, створення класу *Individual*, що розширює масив цілих чисел і доповнює його атрибутом типу *FitnessMin*, потім реєструється оператор *randomOrder*, який застосовує функцію *random.sample()* до діапазону, що відповідає задачі комівояжера довжині *n*. У результаті генерується випадковий список індексів від 0 до *n*-1. Наступним кроком реєструється оператор *IndividualCreator*. Він викликає оператор *randomOrder* і обходить створений ним список з метою створити хромосому, що складається з індексів міст.І насамкінець реєструється оператор *populationCreator*, який породжує список індивідуумів, викликаючи в циклі оператор *Individual-Creator*;

3. Реалізувавши хромосому ми можемо визначити функцію обчислення пристосованості. Для цього служить функція *tspDistance();*

4. Наступний крок – визначення генетичних операторів;

5. Оператори схрещування та мутації. Потрібно мати на увазі, що тепер хромосома – не просто список цілих чисел, а список індексів, що представляє порядок відвідування міст, тому ми не маємо права перемішувати частини двох списків або довільно змінювати індекс. Необхідно використовувати спеціалізовані оператори, призначені для створення допустимих списків індексів , тобто оператор упорядкованого схрещування;

6. Далі реалізовується сам генетичний алгоритм, скористувавшись вбудованим у DEAP алгоритмом *eaSimple* і його об'єктами статистики та залом слави, які дають інформацію, необхідну для подальшого відображення результатів.

Дана програма реалізує 300 популяцій, 200 поколінь, ймовірність схрещування 0.9, мутації 0.1.

3.2.2 Удосконалення алгоритму за допомогою елітизму та результати роботи програми. Спробувавши збільшити кількість поколінь у попередній версії програми, ми прийдемо до висновку, що рішення не покращується. Програма застряє в неоптимальному рішенні, досягнутому раніше покоління 200для 500 поколінь. Це стає очевидно якщо порівняти рисунки 3.5 і 3.6.

Рисунок 3.6 – Статистика для 200 поколінь

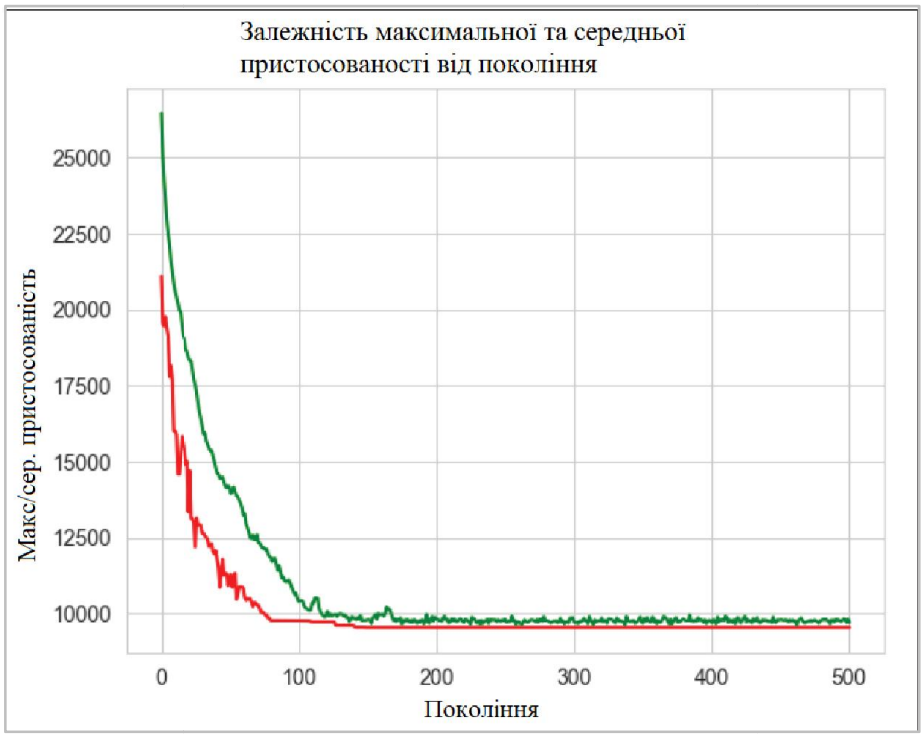


Рисунок 3.7 – Статистика для 500 поколінь

Для компенсації потрібно трохи підвищити степінь використання. Це дозволяє зробити механізм елітизму. Елітизм дозволяє зберегти кращі рішення, захистивши їх від застосування операторів відбору, схрещування та мутації. Для його реалізації доведеться модифікувати код алгоритму *DEAP algorithms*.

Хоча середня пристосованість популяції в генетичному алгоритмі зростає від покоління до покоління, будь-якої миті може статися так, що кращі індивідууми в поточному поколінні зникнуть. Це пов'язано з тим, що оператори відбору, схрещування і мутації змінюють індивідуумів в процесі створення наступного покоління. У багатьох випадках втрата тимчасова, оскільки ці (або навіть найкращі) індивіди знову з'являться в майбутньому поколінні. Але якщо необхідна гарантія, що найкращі індивідууми обов'язково перейдуть у наступне покоління, то можемо застосувати факультативнустратегію елітизму. Це означає, що *n* кращих індивідуумів (*n* – невелике, заздалегідь задане число) копіюються в наступне покоління, до того як всі місця будуть зайняті нащадками, отриманими в результаті

відбору, схрещування і мутації. Скопійовані елітні індивідууми, як і раніше, можуть використовуватися як батьки нових індивідуумів.

Іноді елітизм має помітний позитивний ефект на якість алгоритму, оскільки не потрібно витрачати час на повторне відкриття хороших рішень, втрачених у результаті еволюції.

Для реалізації елітизму використовуємо метод e*aSimpleWithElitism()* аналогічний оригінальному методу *eaSimple()*, але тепер об'єкт *halloffame* використовується для реалізації механізму елітизму. Індивідууми, що зберігаються в об'єкті *halloffame*, просто копіюються в наступне покоління, не піддаючись впливу операторів відбору, схрещування та мутації. Для цього потрібно внести такі модифікації:

– замість того щоб відбирати індивідуумів в кількості, що дорівнює розміру популяції, ми відбираємо їх менше на стільки, скільки індивідів знаходиться в залі слави:

– після застосування генетичних операторів індивідууми додаються із зали слави в популяцію.

У результаті, на рисунку 3.7 ми можемо побачити, що нам вдалося усунути шум, що спостерігався раніше, а також зберігати відстань між кращим і середнім значенням протягом набагато більш тривалого часу, ніж раніше.

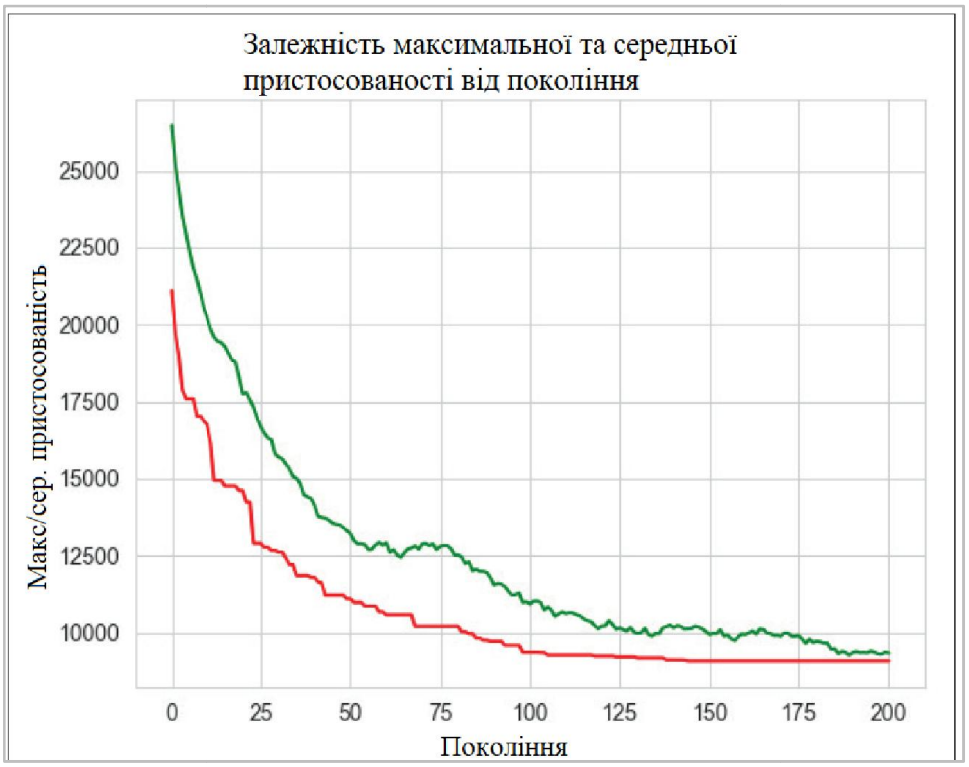


Рисунок 3.8 – Статистика після 500 поколінь з механізмом елітизму

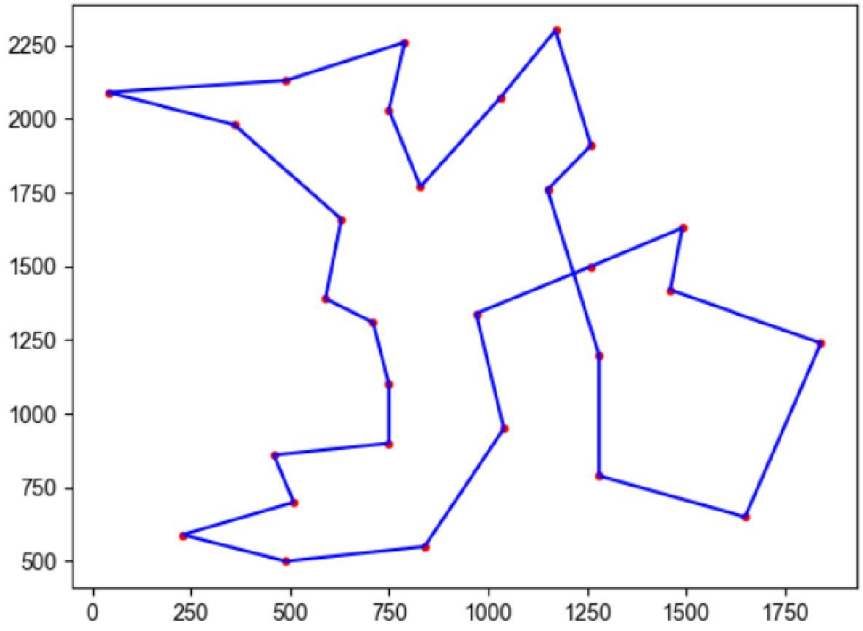


Рисунок 3.9 – Побудований маршрут для комівояжера

3.3 Розробка квантового генетичного алгоритму на Python

3.3.1 Технології для квантового програмування. Для симуляції квантових комп’ютерів написано багато програмних пакетів, зв'язки з ними влаштовані так, щоб їх можна було використовувати з *Python*. Ця мова програмування лишається так само доступною та простою у користуванні навіть для таких складних та неоднозначних задач. Для квантового програмування встановлені спеціальні модулі, такі як *qiskit, cirq, ocean*. Qiskit – це *IBM's* комплект розробки програмного забезпечення з відкритим вихідним кодом для роботи з квантовими комп'ютерами. Цей модуль спрощує розробку квантових додатків, пропонуючи ресурси, необхідні для взаємодії з квантовими системами та симуляторами, Підходить для кінцевих користувачів без досвіду у квантовій розробці. З чотирма пакетами *Qiskit* – *Aqua, Terra, Ignis,* та *Aer* можна працювати як із простими, так і зі складними алгоритмами. Користуватись цим доповненням можна у двох варіантах: або запустити його локально, або в хмарі без встановлення будь-якого ПЗ, використовуючи комп'ютерну лабораторію IBM.*Cirq* – це заснована на *Python* бібліотека програмного забезпечення *Google* для написання, керування та оптимізації квантових схем, а потім їх запуску на квантових системах або симуляторах.На початок 2021 року *Cirq* працює над пропозицією доступу до одного з реальних квантових комп'ютерів *Google*. Тим часом, все ще можна створювати алгоритми та схеми та тестувати їх на квантовому симуляторі.*Google* пропонує різні посібники, які допомагають новачкам перейти від нульового до експертного рівня квантової симуляції за допомогою *Cirq*.*Ocean*– це набір інструментів, що надаються *D-Wave* для вирішення складних завдань за допомогою квантових комп'ютерів. Програмне забезпечення виконує обчислення, необхідні перетворення випадкових завдань у форму, яку може вирішити квантовий комп'ютер.

Для розробки власного алгоритму я обрала модуль *qiskit*. Про нього детальніше. Він працює у двох режимах. Перший – як і всі, звичайний

локальний емулятор. Другий – запуск програми на справжньому квантовому комп'ютері IBM Q.

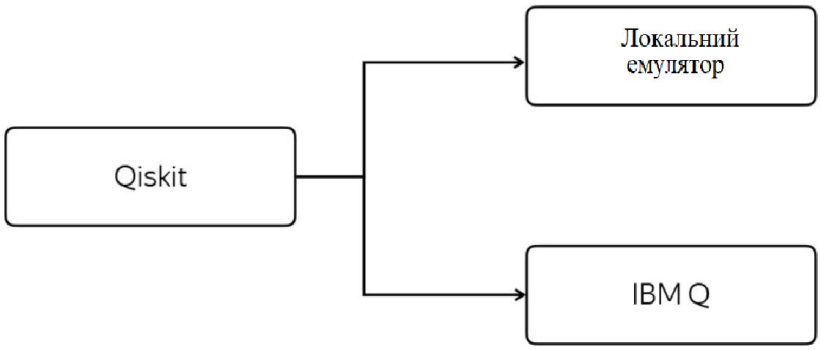


Рисунок 3.10 – Режими роботи обраного модуля

Рисунок 3.11 – Квантовий комп’ютер *IBM-Q*

Виглядає цей комп’ютер зараз як діжка, в якій підтримується екстремально низька температура – близько трьох млК.І *IBM* надає можливість за допомогою хмарних технологій підключитися та запустити ваш код прямо на цій машині.

Для загального доступу є кілька інсталяцій комп'ютера. Один із них 5-кубітний, є 15-кубітний, 16-кубітний, нам доступні навіть 20. Кубіт для цього

комп'ютера – це просто вектор у двовимірному просторі, у якого є певний базис. В одному кубіті може зберігатись нескінченно багато інформації саме через векторну форму.

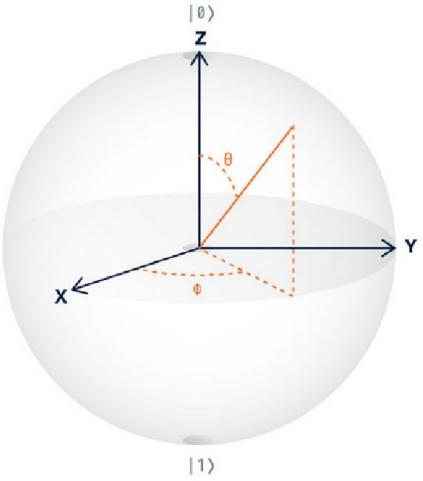


Рисунок 3.12 – Кубіт як вектор, представлений у тривимірній сфері

3.3.2 Операції на квантовому симуляторі. Перше, ло чого ми звикли у класичному імперативному програмуванні порахувати значення змінної, наприклад, у *Python*. Ми хочемо прочитати, у якому стані перебуває кубіт. Але ми ніколи не дізнаємося про точні значення α і β. Якщо ми спробуємо подивитись на кубіт, прочитати, то ми отримаємо або 0, або 1 з відповідними ймовірностями. Імовірності – це просто проекції на відповідні базисні вектори. Друге, що ми можемо робити – привласнювати змінній значенні іншої, проте у квантовому світі так робити не можна. Тобто, є кубіт, якого ми не можемо прочитати, не можемо присвоїти. Що взагалі можна робити?

Кубіт – це вектор. Вектор можна взяти та покрутити по сфері навколо. Щоб покрутити, можна вигадати матрицю, яка це обертання робить. Усі операції над кубітами – це матриці. Вони називаються унітарними.

$$UU^{\dagger} = I$$

Рисунок 3.13 – Комплексно-сполучена матриця

Цей значок означає транспоновану та комплексно-сполучену матрицю. Її властивість полягає у тому, що для будь-якої операції є зворотна. Тобто хоч би як ми покрутили вектор, ми завжди можемо його повернути в колишнє положення.

Є звичайно і ті операції, до яких ми звикли у класичному програмуванні. Наприклад, оператор заперечення (*NOT*):

$$|0\rangle \rightarrow |1\rangle \qquad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$
$$|1\rangle \rightarrow |0\rangle$$

Рисунок 3.14 – Оператор*NOT*

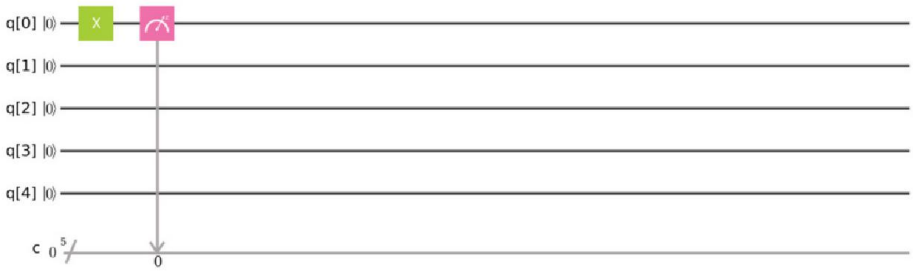


Рисунок 3.15 – Застосування оператора *NOT* на схемі симулятора

У квантовому програмуванні зазвичай користуються схемами. На візуальному симуляторі від *IBM*можна прослідкувати роботу операторів через взаємодію з кубітами. Схема зазвичай виглядає таким чином, що на п’яти горизонтальних лініях ми можемо помістити наші оператори. П’ять ліній означає п’ять кубітів. Оператори позначаються у вигляді блоків. Бібліотека Quantum Composer містить багато різних класів воріт: одиночні кубітні ворота, такі як жовта операція бездіяльності; зелений клас операторів Паулі, які представляють біт-фліпс (*X*, еквівалентний класичному НЕ); фаза-сальто (*Z*); і комбінований біт-фліп і фаза-фліп (*Y*). Також пропонуються операції Clifford, які є синім класом воріт, таких як *H, S і S†* ворота для генерації квантових суперпозицій і складних квантових фаз, а також згаданих раніше двох-кубітних воронок *C-NOT*. Червоні ворота є двофазними, що є

важливим для забезпечення квантового обчислення його потужності. Для вимірювання стану будь-якого кубіту використовуються стандартна операцію вимірювання, яка є простою *Z*-проекцією, присвоєною класичному біту в класичному бітовому регістрі. Сірий бар'єр дозволяє відокремлювати частини схеми від візуальної лінії; при використанні оптимізації, він зупиняє оптимізацію інструментів через бар'єр. Квантовий алгоритм (схема) починається з підготовки кубітів у чітко визначених станах, тут основний стан, | 0, після чого виконують серію одно- і двокубітних воріт в часі з подальшим вимірюванням кубітів.

Такі самі блоки можна написати на мові *Python,* за допомогою бібліотеки *qiskit*. Для початку необхідно ініціалізувати квантовий регістр з одного кубіту, та класичний регістр. Класичний регістр потрібен, щоб записати результат вимірювання. Тобто з квантовим регістром робляться перетворення, результат отримується класичний – 1 чи 0.

Такі оператори як *AND* чи *OR* у квантовому програмуванні не працюють, оскільки вони не зовсім оборотні. Іншими словами, отримуючи «0» при операції «і» ми ніколи не зможемо дізнатись якими були початкові значення. Проте, з’являється інший оператор *CNOT:*

```
if q[0]:
    q[1] = not q[1]
```

$$CNOT|00\rangle \rightarrow |00\rangle$$
$$CNOT|01\rangle \rightarrow |01\rangle$$
$$CNOT|10\rangle \rightarrow |11\rangle$$
$$CNOT|11\rangle \rightarrow |10\rangle$$

Рисунок 3.16 –Оператор *CNOT*

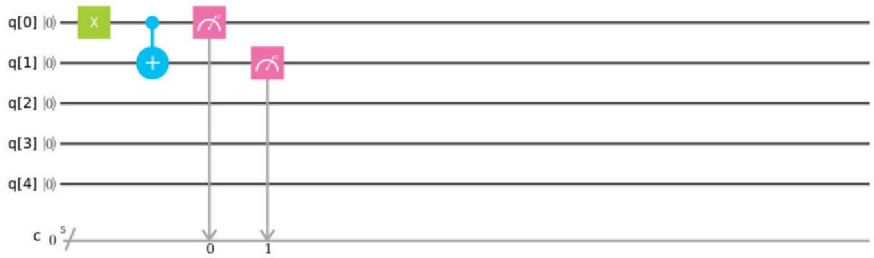


Рисунок 3.17 – Застосування оператора *CNOT* на схемі симулятора

Для опису наступного оператора хочу привести аналогію з грою «орел чи решка», у якій бере участь користувач та комп'ютер. По черзі два гравці мають підкинути уявну монету, але результат не видно одразу. Для комп'ютера це звичайно вибірка 0 чи 1. У гравців є дві спроби, по їх закінченню комп'ютер видає результат – хто виграв, при цьому переможцем є той, в кого більше випало «орлів». Якщо грати із звичайним комп'ютером, без якого ми вже не можемо уявити наше життя, то в обох гравців шанси рівні – 50 на 50. На практиці так і виходить – близько 50% виграшу. У такій самій грі брали участь 372 людини, але їх суперником був квантовий комп'ютер. Результат був цілком очікуваний – 97% виграшу комп'ютера. До речі, ті 3% небагаторазового програшу пов'язані зі збоями в операційній системі. Ось як це працює: звичайний комп'ютер приймає кожну сторону монети за один біт, тобто 0 або 1, і його мікросхема налаштована на положенні монети в даний момент часу. Квантовий комп'ютер працює зовсім по іншому. Кубітом, більш динамічний, для нього є характерним недвоїчний стан. Він може існувати у суперпозиції, або по іншому – ситуації накладання 0 та 1. Такий стан можна назвати не просто невизначеним, а скоріш стан у діапазоні від 0 до 1. По суті, з деякою ймовірністю це може бути 0, а з деякою – 1. Така ймовірність може бути 80/20 або 54/46, 70/30 і т. д. Ключова ідея в тому, що ми змушені відмовитись від точного значення 0 та 1 та допустити деяку неоднозначність. Таким чином, під час гри квантовий комп'ютер створює динамічну комбінацію орла і решки (0 та 1), тому незалежно від того, отримує гравець переможну комбінацію чи ні, суперпозиція залишається незмінною. Квантовий комп'ютер може відокремити 0 чи 1 у самий останній момент, при підрахунках результату наприклад, і з легкістю отримати 100% виграшну комбінацію, тому його опонент приречений на програш. Таку ситуацію можна змоделювати за допомогою оператора Адамара:

$$|0\rangle \rightarrow \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Рисунок 3.18 –Оператор Адамара

Цей оператор на схемі квантового симулятора позначається блоком H:

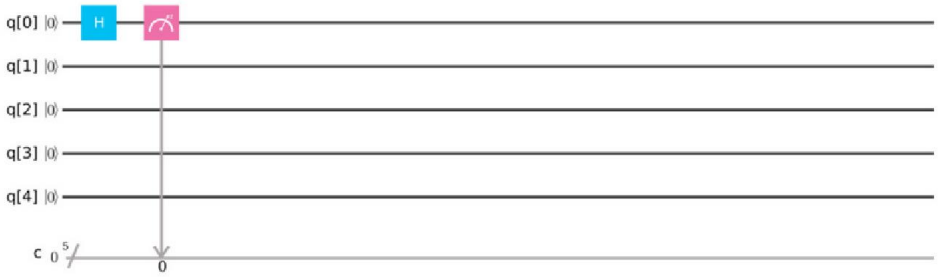


Рисунок 3.19 – Застосування оператора Адамара на схемі симулятора

Рисунок 3.20 – Результат роботи оператора Адамара

3.3.3 Опис алгоритму та реалізація на *QASM* симуляторі. Квантовий генетичний алгоритм – це ГА, який поєднує квантові оператори (обертання, вимірювання, квантові хромосоми) з класичними генетичними операторами (кросовер і мутація). Його можна використовувати в освітніх та дослідницьких цілях.

Розгляньмо задачу у термінах графів:

– міста представлені у вигляді вершин, а вартість/шлях - у вигляді ребер;

– задані відстані будуть кодуватись у вигляді фаз;

– кожне місто має певний шлях до іншого, у кожного шляху є вартість, тобто тут має місце задача мінімізації втрат;

– будуються унітарні оператори, вектори яких є обчислювальними базовими станами, а власні значення є різними комбінаціями цих фаз;

– потім ми застосовуємо алгоритм оцінки фази до певних власних станів, який дає нам можливі загальні відстані всім маршрутів;

– після отримання відстаней ми можемо здійснити пошук за цією інформацією, використовуючи алгоритм квантового пошуку для знаходження мінімуму, щоб знайти найменшу можливу відстань, а також пройдений маршрут. Це дає нам квадратичне прискорення, порівняно з класичним методом грубої сили для великої кількості міст.

Розглянемо випадок, коли кількість вузлів нашого графу дорівнює 4 (n=4). Для цього використаємо модуль *network* для та *matplotlib*. На мові Python описуємо вершини та сторони графа, додаємо позначення фаз на кожному ребрі та візуалізуємо:

```python
G = nx.DiGraph(directed=True)
G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)

G.add_edge(1, 2)
G.add_edge(1, 3)
G.add_edge(1, 4)

G.add_edge(2, 1)
G.add_edge(2, 3)
G.add_edge(2, 4)

G.add_edge(3, 1)
G.add_edge(3, 2)
G.add_edge(3, 4)

G.add_edge(4, 1)
G.add_edge(4, 2)
G.add_edge(4, 3)
```

```python
pos = {1: [0.75, 1.0],
       2: [0.75, 0.15],
       3: [0.5, -0.5],
       4: [1.0, -0.5]}

edge_labels = {(1, 2): '$\\phi_{2\\to 1}$\n $\\phi_{1\\to 2}$',
               (1, 3): '$\\phi_{1\\to 3}$\n $\\phi_{3\\to 1}$',
               (1, 4): '$\\phi_{4\\to 1}$\n $\\phi_{1\\to 4}$',
               (2, 3): '$\\phi_{2\\to 3}$\n $\\phi_{3\\to 2}$',
               (2, 4): '$\\phi_{4\\to 2}$\n $\\phi_{2\\to 4}$',
               (3, 4): '$\\phi_{4\\to 3}$\n $\\phi_{3\\to 4}$'
               }
```

Рисунок 3.21 – Код на мові Python для реалізації графа на 4 вершини

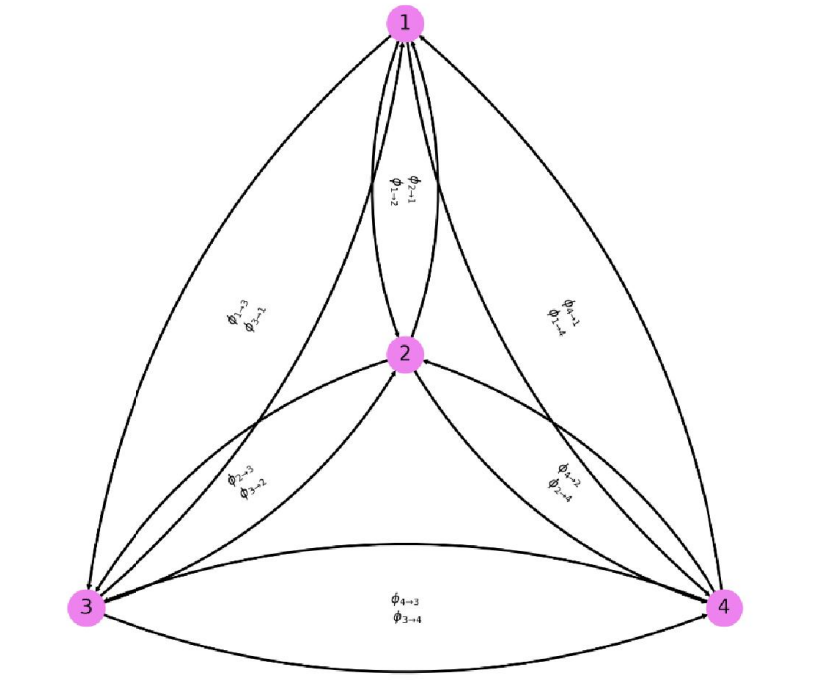


Рисунок 3.22 – Результат виконання коду: граф з чотирма вершинами та описом фаз між ними

Класичний метод, він ще називається методом грубої сили, передбачає побудову матриці відстані між кожної парі вузлів. Тут фаза і означає відстань. Наприклад: $\phi_{1\to 2}$ означає відстань/вартість від вузла «1» до «2». Класичним методом будуємо матрицю:

$$A = \begin{bmatrix} \phi_{1\to 1} & \phi_{1\to 2} & \phi_{1\to 3} & \phi_{1\to 4} \\ \phi_{2\to 1} & \phi_{2\to 2} & \phi_{2\to 3} & \phi_{2\to 4} \\ \phi_{3\to 1} & \phi_{3\to 2} & \phi_{3\to 3} & \phi_{3\to 4} \\ \phi_{4\to 1} & \phi_{4\to 2} & \phi_{4\to 3} & \phi_{4\to 4} \end{bmatrix} \quad (3.4)$$

Усі діагональні елементи матриці рівні 0, таким чином від міста *A* до міста *A* відстань рівна «0». Проблема у тому, що матриця, яка складається з даних відстаней з використанням описаної вище процедури не є унітарною. У попередньому розділі було описано квантові оператори, усі вони мають бути унітарними. Представимо матрицю як:

$$B = \begin{bmatrix} e^{i\phi_{1\to 1}} & e^{i\phi_{1\to 2}} & e^{i\phi_{1\to 3}} & e^{i\phi_{1\to 4}} \\ e^{i\phi_{2\to 1}} & e^{i\phi_{2\to 2}} & e^{i\phi_{2\to 3}} & e^{i\phi_{2\to 4}} \\ e^{i\phi_{3\to 1}} & e^{i\phi_{3\to 2}} & e^{i\phi_{3\to 3}} & e^{i\phi_{3\to 4}} \\ e^{i\phi_{4\to 1}} & e^{i\phi_{4\to 2}} & e^{i\phi_{4\to 3}} & e^{i\phi_{4\to 4}} \end{bmatrix} \quad (3.5)$$

У цьому випадку фази представленні у вигляді тензорних добутків станів з цими фазами у якості коефіцієнтів, тобто відстані будуть додані як фази, які необхідні для пошуку. Діагональні елементи рівні «1». Таким чином побудуємо унітарну матрицю (*U*) з матриці (*B*)для кожного вузла графа для оцінки фази як:

$$U_j = \left(\sum_{i=1}^{n} B[j][i] \times \text{добуток усіх можливих базисних векторів}\right) \text{ where } j, i \geq 0 \text{ and } j, i \in [1, n].$$

Отримаємо таку матрицю:

$$U_j = \begin{bmatrix} e^{ia} & 0 & 0 & 0 \\ 0 & e^{ib} & 0 & 0 \\ 0 & 0 & e^{ic} & 0 \\ 0 & 0 & 0 & e^{id} \end{bmatrix} \quad (3.6)$$

Тепер матриця є унітарною. Значення цієї матриці обчислюється за допомогою квантового алгоритму оцінки фаз. Алгоритм квантової оцінки фази (також названий квантової оцінкою власного значення) може використовувати оцінки власного значення (або фази) власного вектора унітарного оператора.

Теоретично необхідність такої модифікації пояснюється тим, що алгоритм квантової оцінки фаз використовує фазовий «відкат» для запису самої фази U у базизі Фур'є до t кубіт. Коли ми використовуємо кубіт для управління U гейтами він може обертатися (через віддачу) пропорційно фазі$e^{2i\pi\theta}$. Тоді і виявляється необхідність введення унітарної матриці, що застосовує унітарний оператор у цільовому регістрі, тільки якщо його відповідний біт управління $|1\rangle$.

Це допоможе нам для побудови шляху.

Як обчислюються власні відстані класичним методом? 1-2-3-4 → 2-3-4-1 → 3-4-1-2 → 4-1-2-3. Якщо уважно проаналізувати ці комбінації, можна побачити що, 2-3-4-1 це просто обернене 1-2-3-4 по лівій стороні, тобто якщо ми повернемо 1-2-3-4 на 1 отримаємо 2-3-4-1. Аналогічно, якщо ми повернемо 2-3-4-1 на 1, вийде 3-4-1-2, і знову повернувши на 1, отримаємо 4-1-2-3. Таким чином ми бачимо, що 2-3-4-1, 3-4-1-2, 4-1-2-3 просто версія з круговою перестановкою 1-2-3-4. Такими шляхами ми маємо знехтувати, оскільки всі 3 комбінації матимуть ту саму вартість, що й 1-2-3-4. Нам необхідно обрати з усіх можливих комбінації ту, яка при круговій перестановці дасть іншу, відмінну комбінацію

Оберемо перші 6 станів, оскільки ці стани відтворюють правило описано вище:

Таблиця 3.1 – Можливі стани

Можливі стани
1 – 2 – 3 – 4
1 – 2 – 4 – 3
1 – 4 – 2 – 3
1 – 4 – 3 – 2
1 – 3 – 2 – 4
1 – 3 – 4 – 2

На прикладі цих станів розберемо як вони обчислюються виходячи з послідовності шляхів за допомогою функції:

$$|\psi\rangle = \otimes_j |i(j)-1\rangle \quad \text{де } \mathrm{j} = [1..\mathrm{n}] \tag{3.7}$$

Для прикладу розглянемо перший з шести станів вказаних вище – 1-2-3-4 (*i(1) = 4*). Перетворимо у двійкову форму для обчислень базисних станів[42]:

$$\begin{aligned} |i(1)-1\rangle &= |4-1\rangle = |3_{10}\rangle = |11_2\rangle \\ |i(2)-1\rangle &= |1-1\rangle = |0_{10}\rangle = |00_2\rangle \\ |i(3)-1\rangle &= |2-1\rangle = |1_{10}\rangle = |01_2\rangle \\ |i(4)-1\rangle &= |3-1\rangle = |2_{10}\rangle = |10_2\rangle \end{aligned} \tag{3.8}$$

Таким чином ми закодували шлях у вигляд |11000110⟩. Продублюємо значення у таблицю з основними шести обраними станами:

Таблиця 3.2 – Обрані стани

Послідовність	Кодові стани
1 – 2 – 3 – 4	|11000110⟩
1 – 2 – 4 – 3	|10000111⟩
1 – 4 – 2 – 3	|10001101⟩
1 – 4 – 3 – 2	|01001110⟩
1 – 3 – 2 – 4	|11001001⟩
1 – 3 – 4 – 2	|01001011⟩

На мові Pythonбудуємо схему з унітарних венетлів:

```
at = 0
bt = pi/2
ct = pi/8
dt = pi/4
qt = QuantumRegister(3, 'qt')
qct = QuantumCircuit(qt)
qct.cp(ct - at, qt[0], qt[1])
qct.p(at, qt[0])
qct.cp(bt - at, qt[0], qt[2])
qct.cp((dt - ct + at - bt)/2, qt[1], qt[2])
qct.cx(qt[0], qt[1])
qct.cp(-(dt - ct + at - bt)/2, qt[1], qt[2])
qct.cx(qt[0], qt[1])
qct.cp((dt - ct + at - bt)/2, qt[0], qt[2])
qct.draw()
```

Рисунок 3.23 – Програмування унітарних гейтів на схемі квантових кубітів

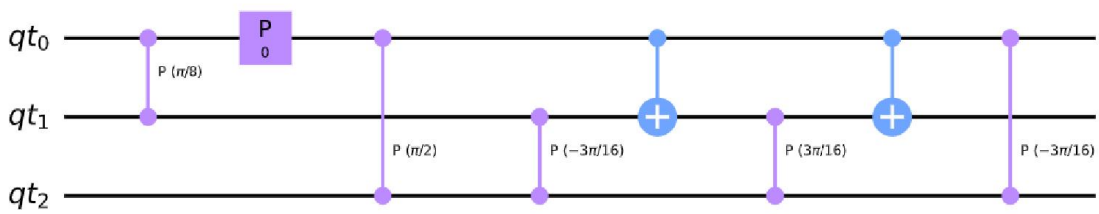


Рисунок 3.24 – Схема розташування унітарних гейтів

Тепер відобразимо загальну схему транспортної задачі за допомогою описаних вище інструментів[43].

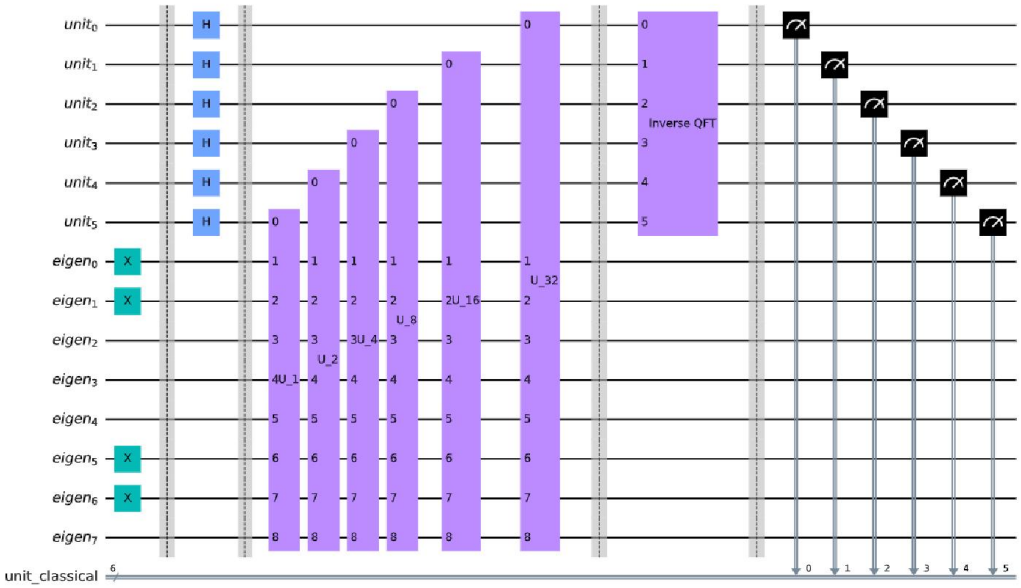


Рисунок 3.25 – Схема унітарних та власних станів, нормалізованих фаз. *(Усі використані позначення описані вище)*

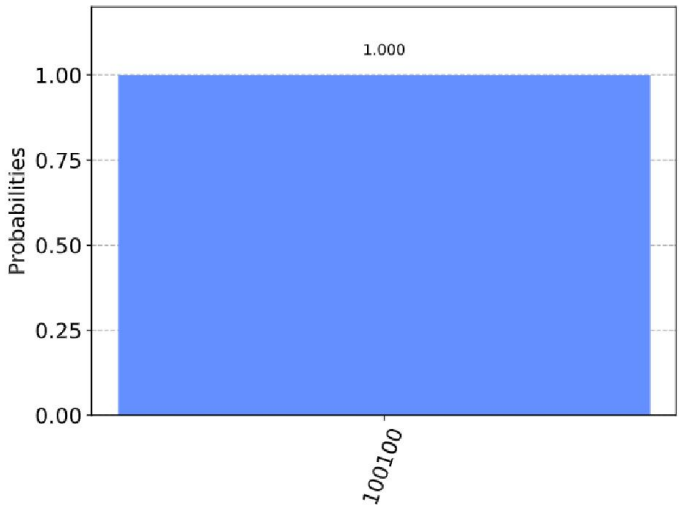


Рисунок 3.26 – Результат на симуляторі *QASM*

РОЗДІЛ 4
ТЕСТУВАННЯ ТА ВВЕДЕННЯ В ЕКСПЛУАТАЦІЮ

Коло завдань, що вирішуються за допомогою ГА постійно розширюється, ці алгоритми адаптуються до реального еволюційного процесу, створюються нові різновиди генетичних операторів, до них додаються ламарківські оператори. Розвиток методів комп'ютерної підтримки прийняття рішень потребує подальшого вдосконалення та дослідження ГА [44, 45]. Одним із напрямків досліджень у цій галузі є тестування ГА. Їх ефективність при вирішенні конкретного завдання прийнято оцінювати двома факторами: швидкістю (час досягнення заданої якості популяції або її збіжності) та стійкістю (здатність алгоритму до самостійного виходу з точки локального екстремуму та здатність постійно збільшувати якість популяції від покоління до покоління). Донедавна ефективність більшості конкретних ГА оцінювалася шляхом тестування вирішення завдання отримання бітового вектора з максимальним числом одиничних розрядів. Чим швидше ГА знаходив найкраще рішення, тим він вважався ефективнішим. Наразі це завдання вже не є об'єктивним засобом тестування. В даний час для того, щоб допомогти дослідникам ГА порівняти отримані результати з рекордними оцінками, формуються електронні бібліотеки тестових завдань. Наприклад, велика бібліотека тестових завдань у сфері дослідження операцій є у Лондонському Королівському Коледжі [46].

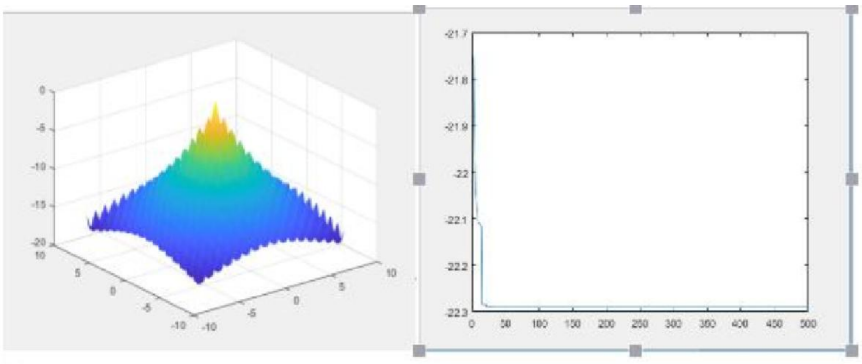
4.1 Тестування генетичного алгоритму

4.1.1 Тестова функціяЕклі. Для тестування у средовищі *Matlab*є спеціально розроблений модуль, у нових версіях він позначається як *OptimizationToolbox***.** Тут можна перевірити алгоритм за допомогою декількох функцій. Проведемо декілька дослідів за допомогою функції Еклі, вона виглядають наступним чином:

$$f(x) = -a\exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp. \quad (4.1)$$

Так як генетичні алгоритми використовують стохастичність, то для того, щоб визначити, наскільки ефективним є ваш ГА потрібно запустити його на одній і тій же тестовій функції кілька разів і лише після цього аналізувати результат.

Розмір популяції дорівнює 20, розмірність простору рішень цільової функції рівна 2. Крива, що відображена нижче – еволюційна крива оптимальної придатності, що за розрахунками становить 22,289642531756776:



```
bestv =

  22.289642531756776


optxx =

  -29.472140762463344 -29.472140762463344
```

Рисунок 4.1 – Результат перевірки ГАфункцією Еклі, *N*=20, *K=2*

Розмір популяції становить 40, розмірність простору цільової функції 2. Оптимальна придатність у цьому випадку становить 22,289642531756776:

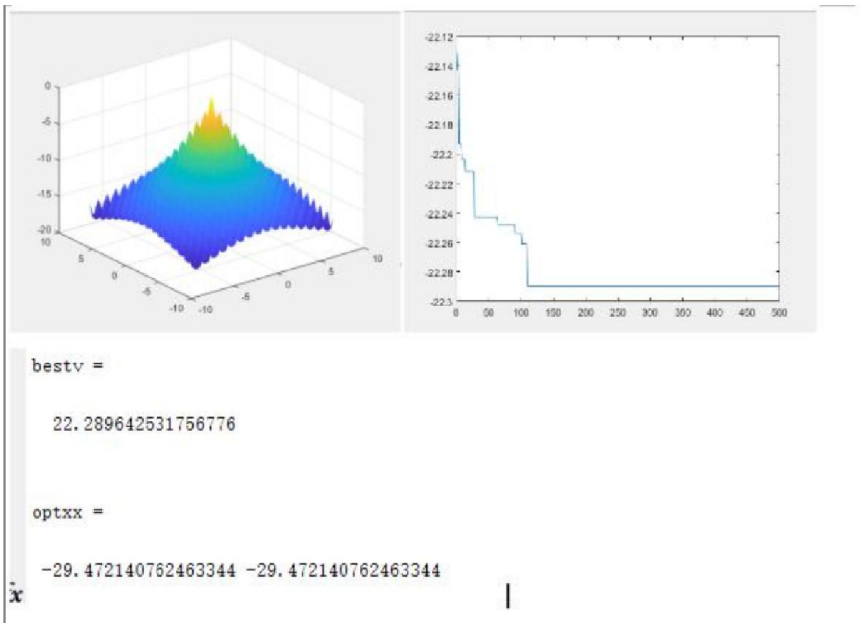


Рисунок 4.2 – Результат перевірки ГА функцією Еклі, *N*=40, *K=2*

Розглянемо випадок, коли розімр популяції становить 80, розмір простору рішень цільової функції лишаємо рівним 2. Таким чином отримаємо оптимальну придатність 22,289642531756776:

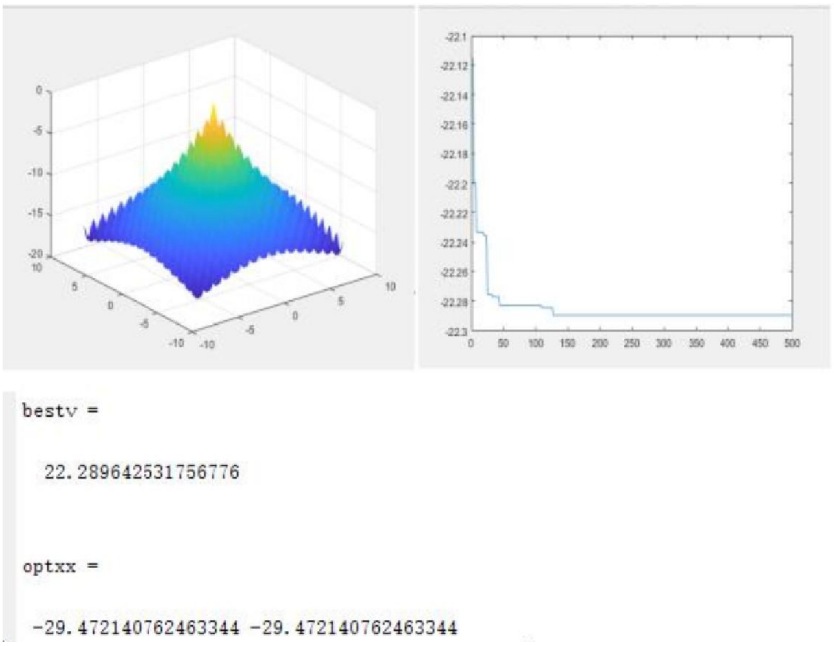


Рисунок 4.3 – Результат перевірки ГА функцією Еклі, *N*=80, *K=2*

Отже, незважаючи на розмір популяції оптимальність генетичного алгоритму лишається сталою. Тепер змінимо розмірність простору рішень

цільової фінкції. На наступному прикладі вона становитиме 5, розмір популяції 20.Оптимальна придатність змінилась і тепер становить 22.201765015798429.

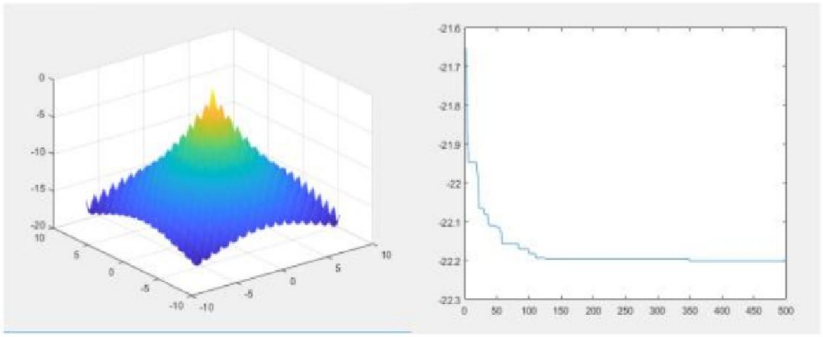


Рисунок 4.4 – Результат роботи цільової функції Екліпри *N=20, K=5*

Збільшимо розмір популяції *N*=40, розмірність простору рішенб цільової функції лишаємо сталу. У цьому випадку оптимальна придатність змінилась і становить 22.243328373427431:

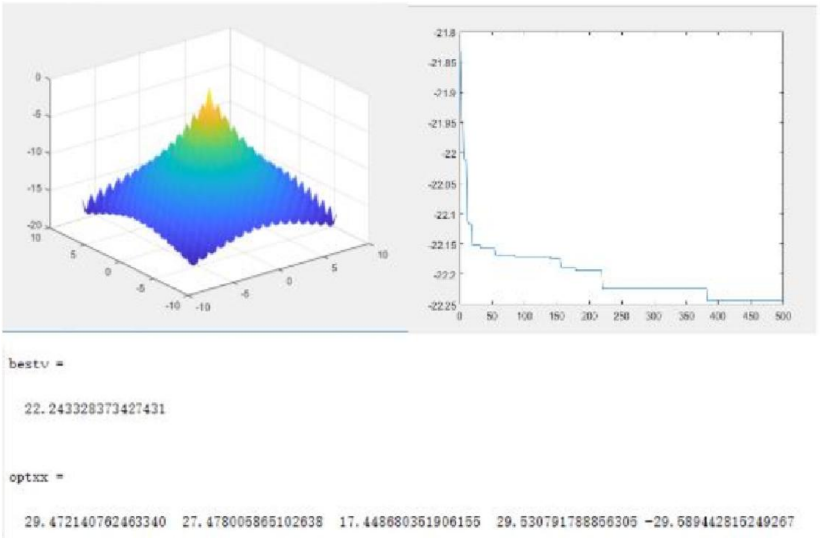
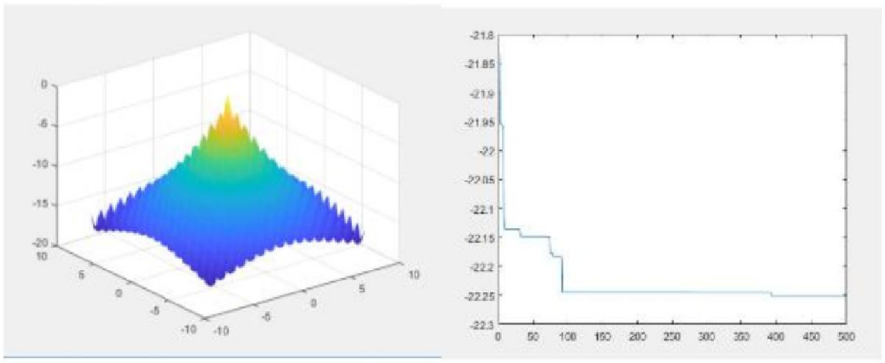


Рисунок 4.5 – Результат роботи цільової функції Екліпри *N=40, K=5*

Далі тестовий випадок виконується при розмірі популяції *N*=80, а розмірності простору рішень цільової функції 5, оптимальна придатність зростає та становить 22,251041872858458:



```
bestv =

  22.251041872858458


optxx =

  29.472140762463340  29.472140762463340 -25.425219941348974 -29.589442815249267  25.542521994134894
```

Рисунок 4.6 – Результат роботи цільової функції Екліпри *N=80, K=5*

Проведемо таке тестування для простору рішень розмірності 10. Почнемо з популяції *N*=20. Результат на рисунку відображає еволюційна крива оптимальної придатності, що становить 22.219067652164274:

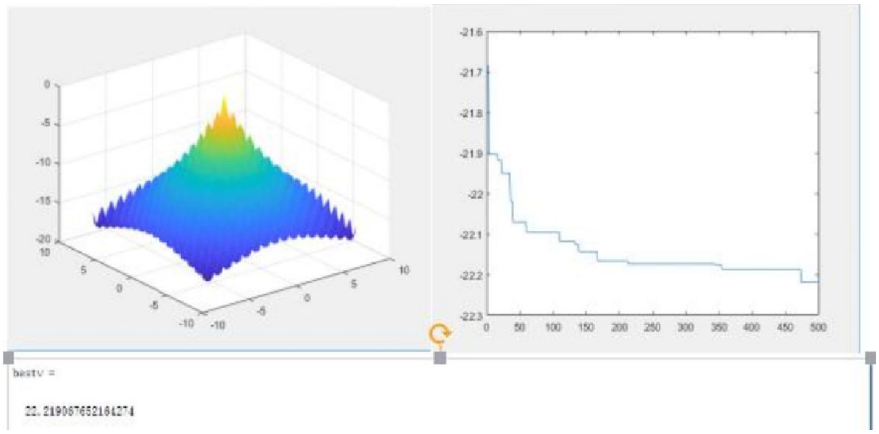


Рисунок 4.7 – Результат роботи цільової функції Екліпри *N=20, K=10*

Так само підвищемо розмір популяції, при цьому залишимо розмірність простору рішень, маємо оптимальну придатність –22.165640402833631:

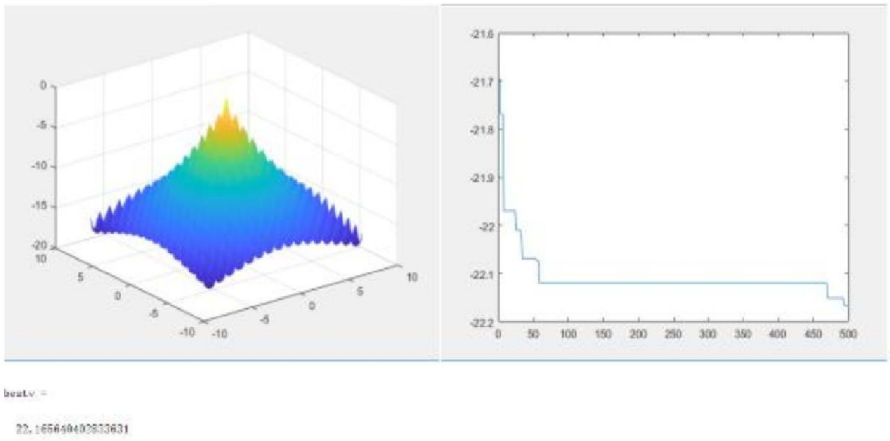


Рисунок 4.8 – Результат роботи цільової функції Екліпри *N=40, K=10*

Останній тестовий випадок виконується з оптимальною придатністю 22,157869382223808:

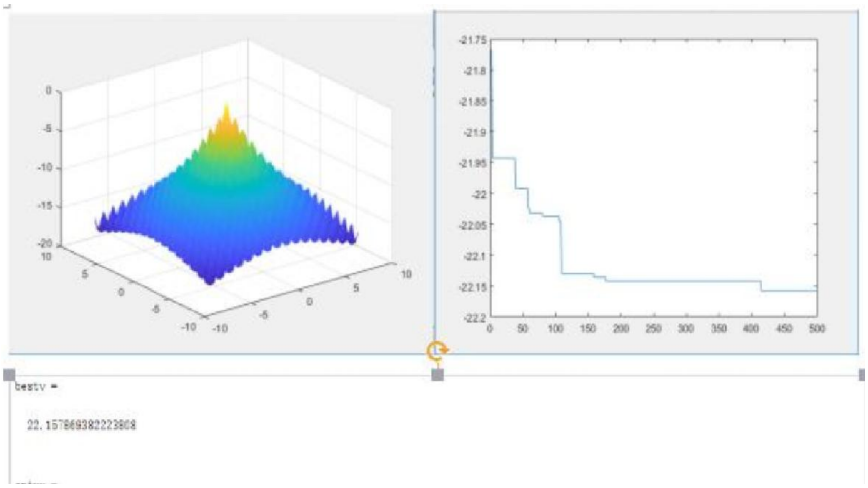


Рисунок 4.9 – Результат роботи цільової функції Екліпри *N=80, K=10*

Порівняльний аналіз дає зрозуміти, що для функції Еклі, коли розмірність низька, то зміна розміру популяції найменше впливає на точність оптимального рішення.Оптимальна придатність функції залежить від розмірності простору рішень, таким чином, що коли розмірність відносно мала (*K*=2) при збільшенні популяції оптимальність не змінюється. При

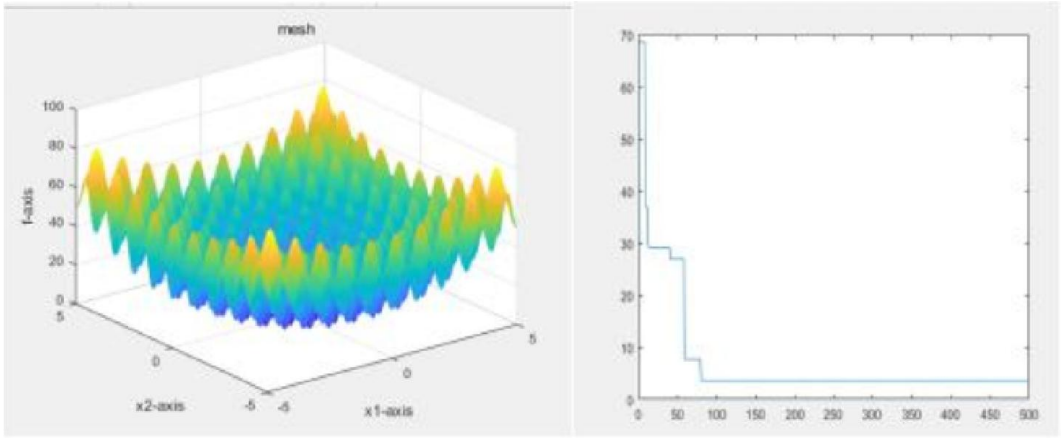
розмірності K=5 при зібльшенні популяції оптимальна придатність збільшується, а при K=10 навпаки зменшується.

4.1.2 Тестова функція Растрігена. Функція Растригіна – це впукла функція, що використовується для тестування ефективності алгоритмів оптимізації. Знаходження мінімуму цієї функції є досить важким завданням через велику область пошуку і велику кількість локальних мінімумів. Математичний опис функції Растрігена виглядає так:

$$f(\mathbf{x}) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i)\right], \text{де } A = 10 \text{ и } x_i \in [-5.12, 5.12]. \quad (4.2)$$

Протестуємо генетичний алгоритм за допомогою цієї функції. Для кожної розмірності простору рішень (K= 2, 5, 10) обчислимо функцію з розміром популяції (N = 10, 20, 40, 80).

Перший тестовий випадок: K= 2, N= 20. Крива, що відтворена на рисунку – еволюційна крива оптимальної придатності, тут вона становить -3.604579690618246.

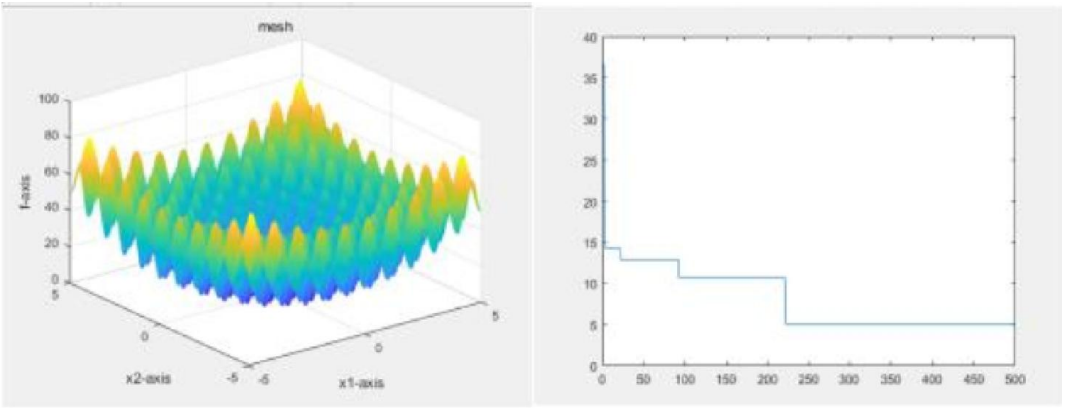



```
bestv =

   -3.604579690618246
```

Рисунок 4.10 – Результат роботи цільової функції Растрігенапри *N=20, K=2*

Далі розглянемо випадок, коли розмір популяції становить 40, розмірність простору цільової функції дорівнює 2. Оптимальна придатність функції = -4.993192447254399:

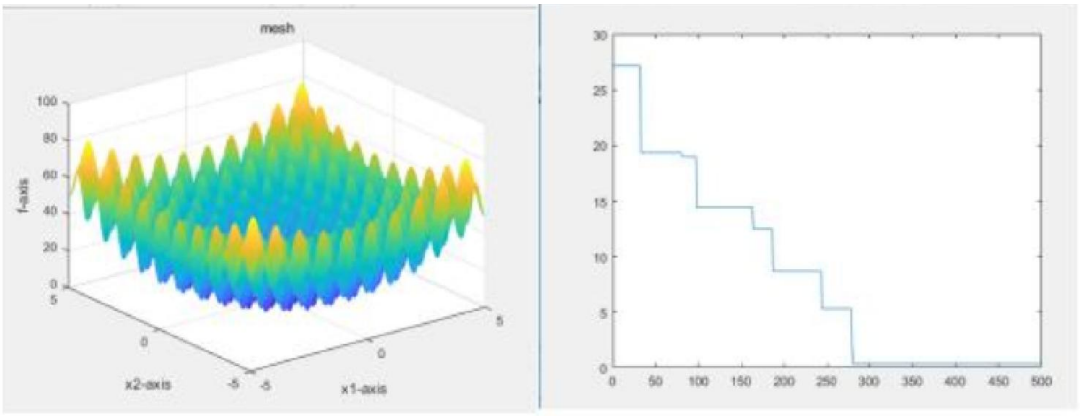



```
bestv =

  -4.993192447254399
```

Рисунок 4.11 – Результат роботи цільової функції Растрігенапри *N=40, K=2*

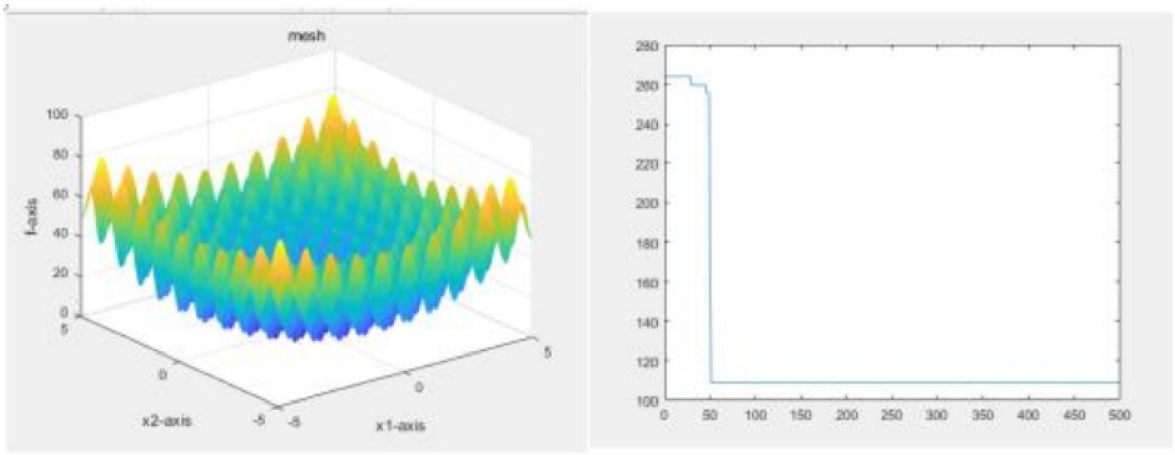
Аналогічним способом розглянемо усі тестові випадки:




```
bestv =

  -0.340269261768334
```

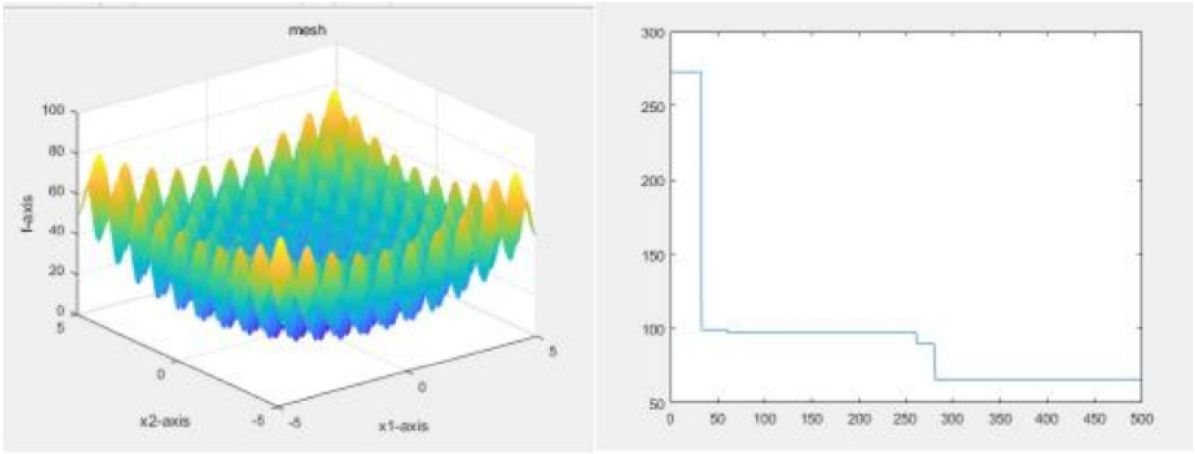
Рисунок 4.12 – Результат роботи цільової функції Растрігенапри *N=80, K=2,* придатність -0,340269261768334




```
bestv =

   -1.088091660864812e+02
```

Рисунок 4.13 – Результат роботи цільової функції Растрігенапри *N=20, K=5,* придатність -1.088091660864812e + 02




```
bestv =

  -65.724915692571770
```

Рисунок 4.14 – Результат роботи цільової функції Растрігенапри *N=40, K=5,* придатність -65.724915692571770

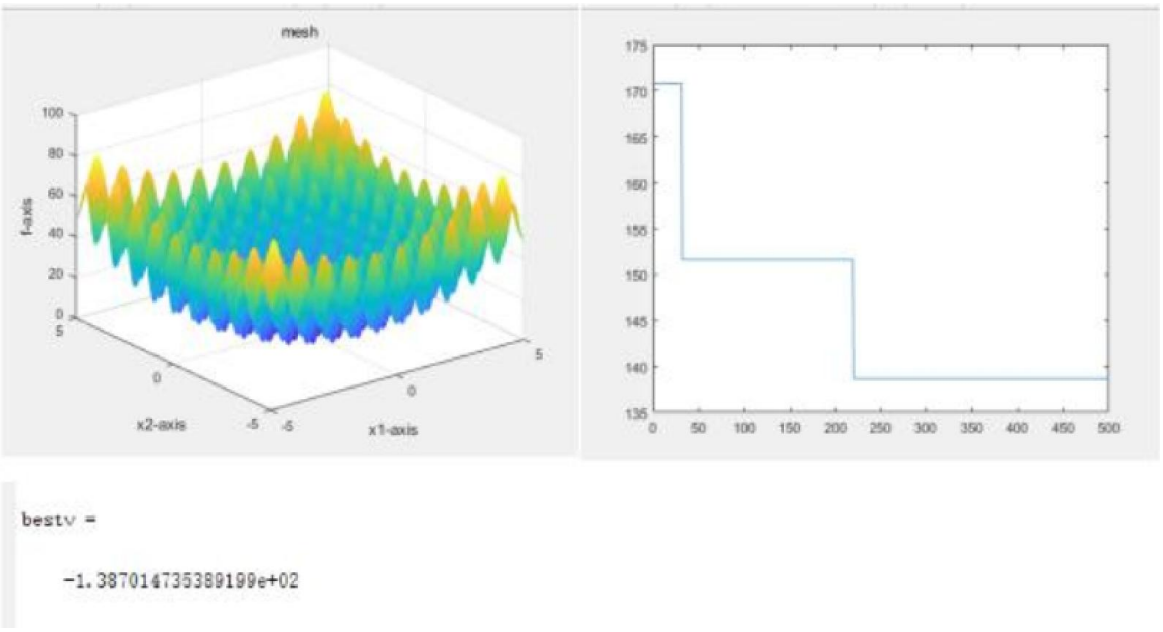


Рисунок 4.15 – Результат роботи цільової функції Растрігенапри *N=80, K=5,* придатність -1,387014735389199e + 02

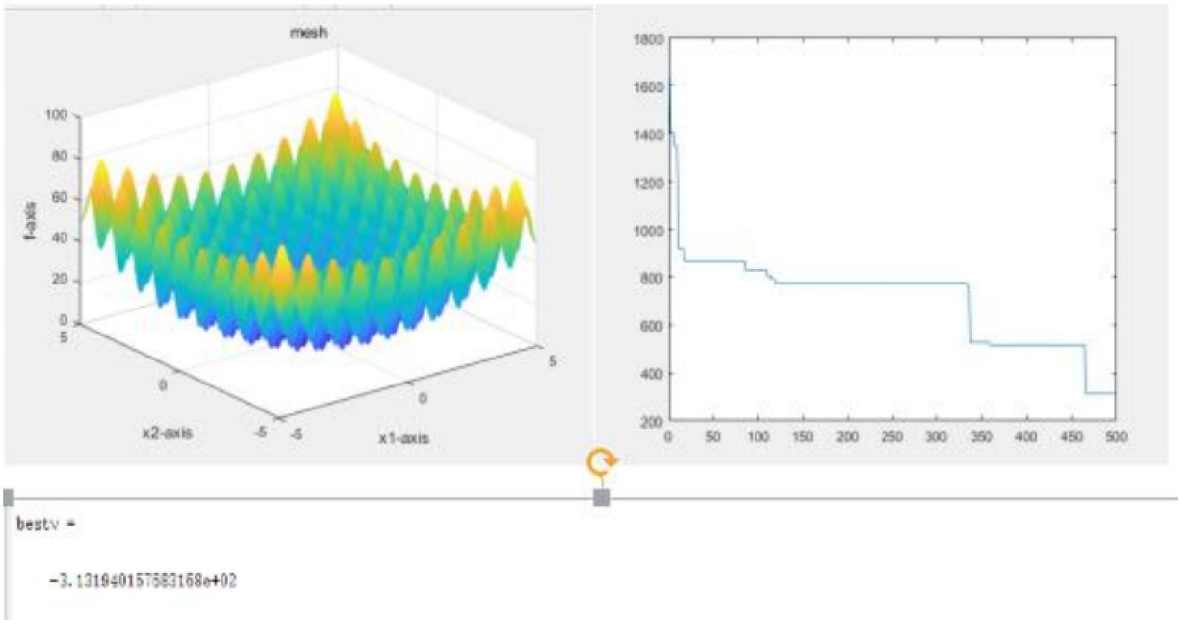


Рисунок 4.16 – Результат роботи цільової функції Растрігенапри *N=20, K=10,* придатність -3.131940157583158e + 02

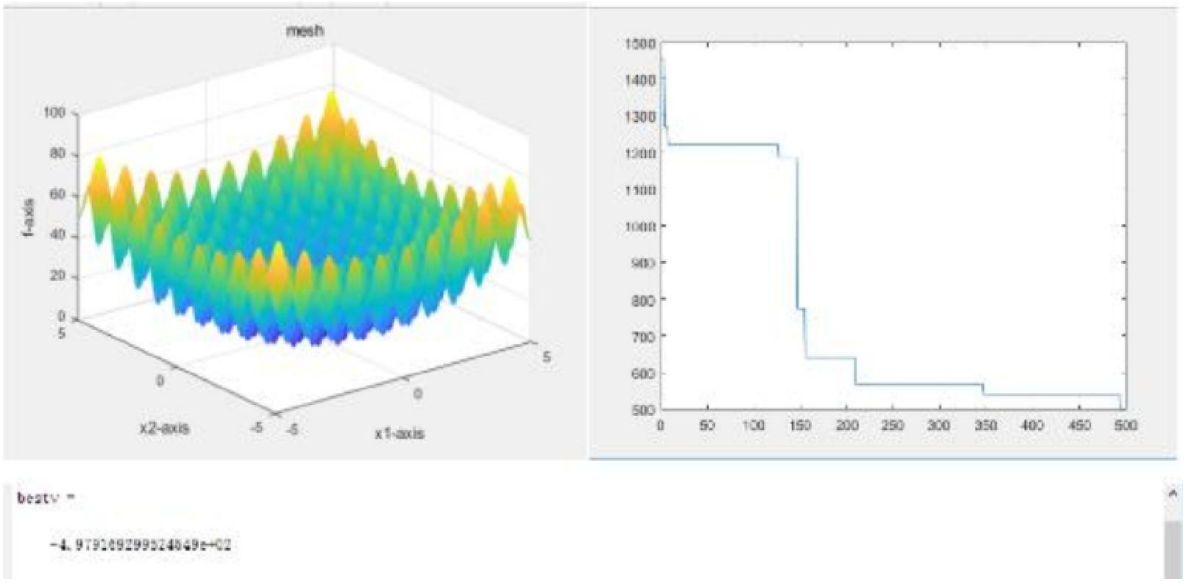


Рисунок 4.17 – Результат роботи цільової функції Растрігенапри *N=40, K=10,* придатність -4.979169299524549e + 02

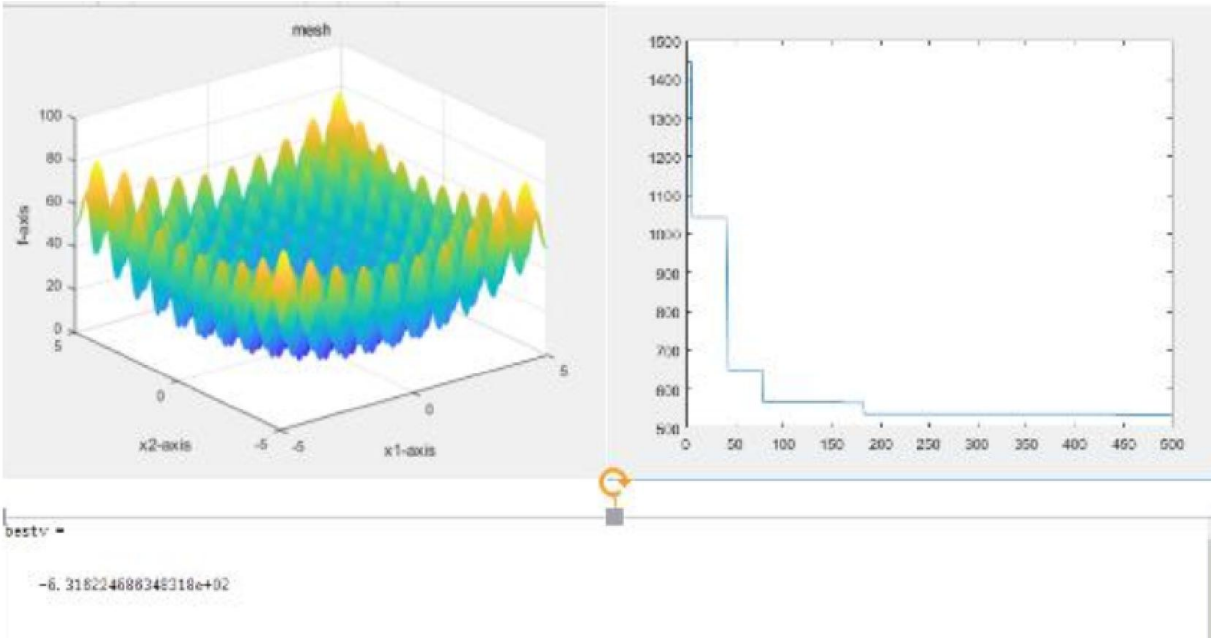


Рисунок 4.18 – Результат роботи цільової функції Растрігенапри *N=80, K=10,* придатність -5.316224586348318e + 02

Отже, для функції Растригіна зміна розмірності простору рішень та зміна популяції роблять оптимальне рішення більш точним.

Вибір фітнес-функції безпосередньо впливає на швидкість збіжності генетичного алгоритму і те, як швидко і чи можна взагалі знайти оптимальне

рішення. Неправильний проектування фітнес-функції може призвести до того, що відмінності індивідума будуть занадто малі, що призведе до екстремальних локальних значень.

4.1.3 Порівняльні таблиці.

Таблиця 4.1 – Залежність часу (сек) виконання алгоритму від розміру популяції та розмірності простору рішень

Популяція / Розмірність	20	40	80
2	0,284	0,5	0,857
5	0,566	0,936	1,387
10	0,876	1,199	1,468

Таким чином, можна зробити висновок, за умови, що розмір простору рішень лишається статичним, чим більша популяція тим більше потрібно ітерацій (залежно від умови зупинки алгоритму), тоді і час виконання алгоритму більший, крива оптимальності постійно змінюється. За умови статичного, незмінного розміру популяції спостерігається така залежність: чим більша розмірність простору рішень, тим менша швидкість виконання алгоритму.

Таблиця 4.2 – Залежність пристосованості алгоритму від розміру популяції та розмірності простору рішень *(функція Еклі)*

Популяція / Розмірність	20	40	80
2	22,289642531756776	22,289642531756776	22,289642531756776
5 ↑	22.201765015798429	22.243328373427431	22,251041872858458
10↓	22.219067652164274	22.165640402833631	22,157869382223808

Таблиця 4.3 – Залежність пристосованості алгоритму від розміру популяції та розмірності простору рішень *(функція Растрігена)*

Популяція / Розмірність	20	40	80
2↓↑	-3.604579690618246	-4.993192447254399	-0,340269261768334
5↑↓	-1.088091660864812e + 02	-65.724915692571770	-1,387014735389199e + 02
10↓↓	-3.131940157583158e + 02	-4.979169299524549e + 02	-5.316224586348318e + 02

4.2 Тестування квантового алгоритму

Більшість квантових алгоритмів використовують ідеї та підходи, на яких ґрунтується кубітне представлення популяції. Наприклад, деякі алгоритми, засновані на ідеях квантових обчисленьпредставлені у [47]. Потенційні підходи до вирішення задач оптимізації включають варіант переходу до кутрітного представлення популяції. З огляду на це, у цьому розділі будуть порівнюватись квантові генетичні алгоритми із кубітною(QGA) та кутрітною(QGA_3) системою ініціалізації популяції.

Оцінка та аналіз алгоритмів буде виконано за таких функцій:

– Функція де Йонга:

$$F_1 = 100\left(x_1^2 - x_2^2\right) + \left(1 - x_1\right)^2, \quad -2.048 \le x_i \le 2.048,\ i = 1, 2. \tag{4.3}$$

Ця функція має один глобальний мінімум: $F_1(1,1) = 0$.

– Функція ціни Гольдштейна:

$$F_2 = \left[1 + (x + x + 1) \cdot (19 - 14x + 3x - 14x + 6x\,x + 3x)\right] \cdot \left[30 + (2x - 3x) \cdot (18 - 32x + 12x + 48x - 36x\,x + 27x)\right], \quad -2 \le x \le 2,\ i = 1, 2. \tag{4.4}$$

Функція має один глобальний мінімум: $F_2 = (0, -1) = 3$.

– Функція Шаффера:

$$F_3 = 0.5 + \frac{\sin^2\left(x_1^2 + x_2^2\right) - 0.5}{\left[1.0 + 0.001\left(x_1^2 + x_2^2\right)\right]^2}, \quad -100 \le x_i \le 100,\ i = 1, 2 \tag{4.5}$$

Глобальний мінімум: $F_3(0,0) = 0$.

– Функція шести пік:

$$F_4 = 10 + \frac{\sin(1/x)}{0.1 + (x - 0.16)^2}, \quad 0 \le x \le 1. \tag{4.5}$$

Глобальний мінімум функції: $F_4(0,1275) = 19,8949$.

– Функція подвійного полюса:

$$F_5 = \left(4 - 21x_1^2 + \frac{1}{3}x_1^4\right)x_1^2 + x_1x_2 + \left(-4 + 4x_2^2\right)x_2^2, \quad -3 \le x_i \le 3,\ i = 1, 2. \tag{4.6}$$

Глобальний мінімум функцій: $F_5(-0.0898, 0.7126) = F_5(0.0898, -0.7126) = -1.031628$.

– Багатопікова позитивна функція:

$$F_6 = e^{-0.001x}\cos^2(0.8x), \quad -2 \le x \le 5. \tag{4.7}$$

Глобальний максимум: $F_6(0) = 1$.

Також перевіримо вже розглянуту вище функцію Еклі. Її глобальний мінімум$F_7(0,0) = 0$.

Функція Растрігена, що вже була застосована для тестування генетичного алгоритму вище, тут її глобальний мінімум: $F_8(0,0) = 0$.

Для імітаційного тестування QGAта QGA_3 було обрано такі параметри: розмір популяції $s = 10$, загальне число ітерацій $t = 500$і точність $\varepsilon = 1 * 10^{-6}$.

Далі зупинимось на виборі параметра μ. Цей параметр означає еволюцію квантової системи та грає ключову роль у алгоритмі, він визначає балас між глобальним (операція квантового спостереження) і локальним (квантові гейти) пошуком рішення. Для оптимального значення μ необхідно врахувати його вплив на поведінку основних параметрів, що характеризують евалюційність квантового алгоритму, таких як наприклад пристосованість найкращої особини та стандартне відхилення від функції. Характерна поведінка параметра μ та його вплив показано на рисунку 4.19 на прикладі функції F_8.

Якщо $\mu < 0{,}95$, то визначальною опетацією є квантове спостереження, ми шукаємо глобальний екстремум функцій через його імовірнісну природу. У той же час кут повороту квантів більше 15 градусів, тобто занадто великий для локального пошуку. Зі збільшенням значення μ, кут повороту наближається до свого оптимального значення ≈ 4 градуси, це значення кута

відповідає $\mu \approx$ 0,998/0,996: при цьому значенні параметра функція пристосованості та стандартне відхилення досягають мінімального значення.

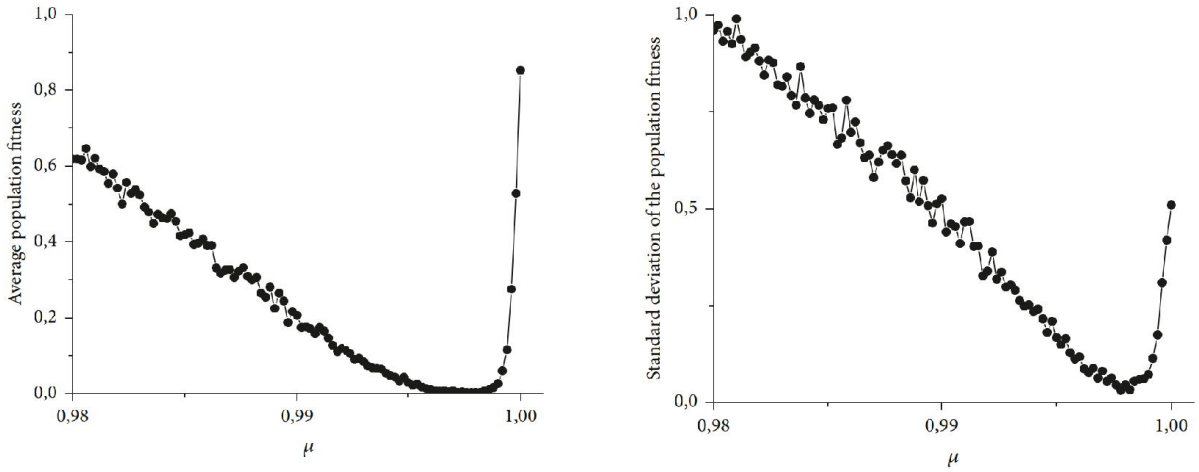


Рисунок 4.19 – Ілюстрація впливу параметра μ. На графіках однакова кількість ітерацій, t = 500.

Поведінка квантового алгоритму при $\mu >$ 0,998 залежить вже від кількості ітерацій:

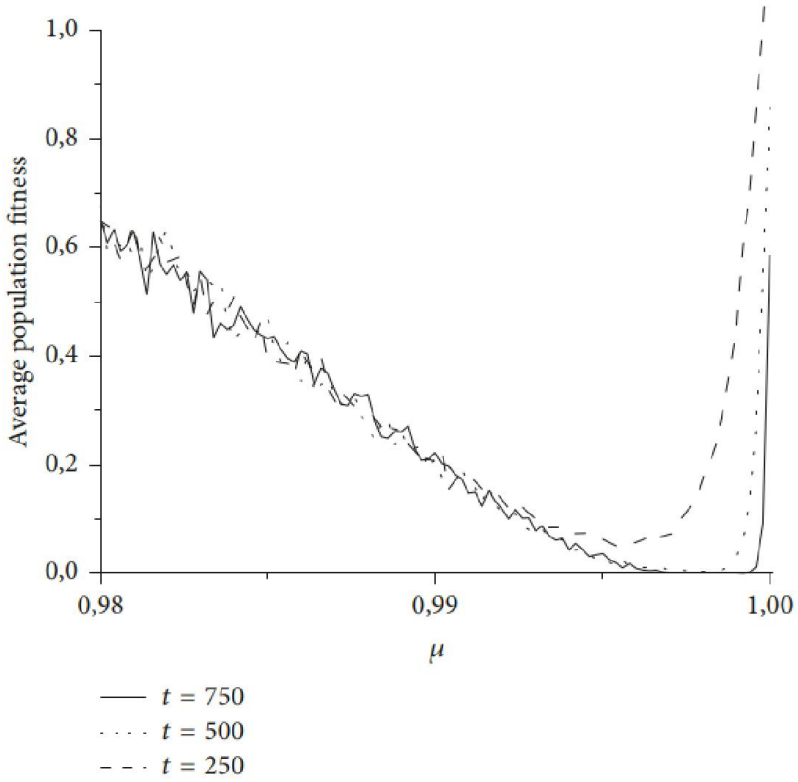


Рисунок 4.20 – Фітнес-функція для різних параметрів ітерацій при μ>0,998

Роль оператора катастофи відображена на рисунку 4.21:

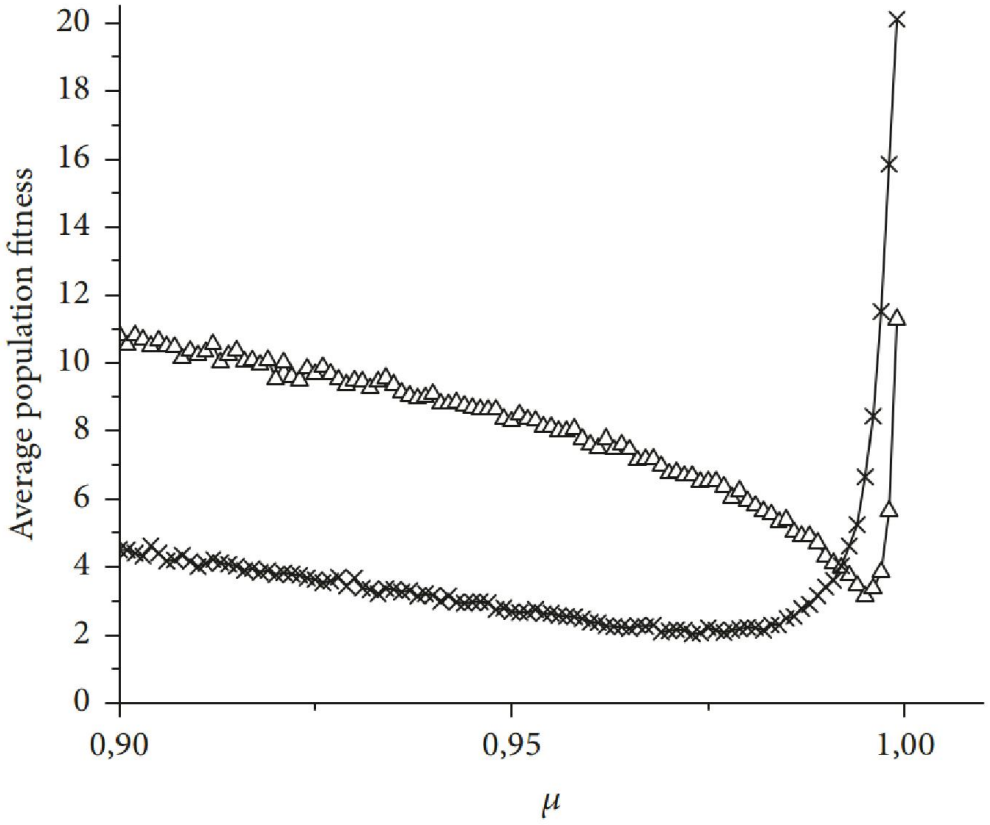


Рисунок 4.21 – Квантовий оператор катасрофи на прикладі функцій$F_{7,8}$, де символом «×»позначена доріжка виконання QGA_3 з оператором катастрофи, а доріжка «∆» QGA_3 без оператора катастрофи

Зважаючи на імовірнісний механізм роботи квантових генетичних алгоритмів всі дані в Таб.4.4 приведено як результат усереднення по 100 запусках. Тут f_{avr} – середнє значення функції пристосованості найкращої особини популяції, f_{sd} (sd) – його середньоквадратичне відхилення:

Таблиця 4.4 – Результати роботи тестових функцій

Function	QGA				QGA_3			
	f_{opt}	$\overline{f}$	sd	$\overline{t}$ (sec)	f_{opt}	$\overline{f}$	sd	$\overline{t}$ (sec)
F_1	0.000	0.011	0.011	0.035	0.000	0.005	0.012	0.024
F_2	3.000	3.050	0.132	0.034	3.000	3.000	0.001	0.022
F_3	0.000	0.004	0.012	0.041	0.000	0.001	0.002	0.029
F_4	19.895	19.894	0.001	0.014	19.895	19.895	0.000	0.011
F_5	−1.031	−1.021	0.008	0.034	−1.031	−1.031	0.000	0.024
F_6	1.000	1.000	0.000	0.018	1.000	1.000	0.000	0.012
F_7	0.008	0.035	0.071	0.038	0.000	0.001	0.003	0.022
F_8	0.013	0.051	0.152	0.037	0.000	0.002	0.032	0.024

На основі експерементальних результатів можна сказати, що QGA_3 дозволяє зменшити час обчислень, будучи більш продуктивним за звичайний QGA. Також, у випадку великої кількості параметрів у алгоритмі, які потрібно оптимізувати, ефективність QGA_3 можна підвищити за рахунок додавання квантового оператора катастрофи. При цьому час роботи алгоритму не збільшується більше ніж на 4%.

ВИСНОВКИ

У дипломній роботі магістра розроблено квантовий генетичний алгоритм для вирішення задачі побудови оптимального шляху. Також представлені генетичні алгоритми у різних середовищах розробки таких як *Matlab*та *PyCharm*задлядостовірного тестування. Запропоновані квантові генетичні алгоритми мають розбіжну структуру між собою: кубітне та кутрітне представлення алгоритму. Проведено дослідження предметної області, спроектовано квантовий генетичний алгоритм як симбіоз генетичного алгоритму та квантової механіки. Було проведеногрунтовний аналіз програмних аналогів та наукових досліджень в області квантових генетичних алгоритмів, досліджено методи побудови алгоритмів та їх особливості, проведено тестування на симуляторі за допомогою тестових функцій, алгоритми є ефективними та відповідає вимогам поставленим на етапі постановки завдання, доведена оптимальність.

Робота також є практичною, оскільки задачі оптимізації сьогодні широко розповсюджені, а квантові обчислення не тільки вдосконалюють їх рішення, а і розширюють можливості. Квантові алгоритми створюються для роботи на квантових комп’ютерах, проте вчені займаються розробками на квантових симуляторах, таким чином створючи певну модель поведінки алгоритмів. Перспективи та можливості квантових комп’ютерів зосереджені на великій області досліджень: квантові обчислення можуть передбачити поведінку елементарних частинок, модулювати молекули ДНК та розробляти нові лікарські препарати; вони допомагають зрозуміти поведінку атомів, оскільки їх можна змоделювати на квантових комп’ютерах; квантові агоритми шикоро використовуються у штучному інтелекті, оскільки вони пришвидшують обчислювальні процеси.

У даній роботі розглянута перспектива використання квантових алгоритмів у задачах логістики у симбіозі з генетичними алгоритмами, які в

свою чергу створюють свою еволіцію у задачах пошуку рішень. Поєднання цих алгоритмів є перспективним та ефективним рішеням задач оптимізації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Genetic Algorithm Tom V. Mathew Assistant Professor, Department of Civil Engineering, Indian Institute of Technology Bombay, Mumbai-400076.

2. Иванова Е.А. « Можливість застосування генетичних алгоритмів у рішенні задач складання розкладу» // Colloquium-journal. 2018. No 3-1 (14). С. 36-38.

3. Омельченко Д.А., Фешина Е.В «Сфери застосування нейронних мереж у майбутньому « //У збірнику: концепції фундаментальних і прикладних наукових досліджень , збірник статей Міжнародної науково-пракладної конференції: у 2 частинаxx. 2018. С. 36-40.

4. Омельченко Д.А. Іванова Е.А. Кубанский державний аграрний університет ім. І. Т. Трубліна «Генетические алгоритмы. Прошлое, настоящее и перспективы применения в будущем»

5. Hisoblash va amaliy matematika muammolari ilmiy jurnali no1 2015, mundarija scientific journal problems of computational and applied mathematics, issue 1: mathematical modelling analysis of features of genetic algorithms c.87-102 Kabildjanov A.S.

6. Genetic Quantum Algorithm and its Application to Combinatorial Optimization Problem, Kuk-Hyun Han

7. Ситник Г. Інформаційна безпека [Електронний ресурс] / Галина Ситник // СерчІнформ – Режим доступу до ресурсу: https://searchinform.ru/informatsionnaya-bezopasnost/.

8. Дослідженння IBM і Ponemon Institute [Електронний ресурс] // IBM – Режим доступу до ресурсу: https://www-03.ibm.com/press/ru/ru/pressrelease/50084.wss.Шквір В. Д.

9. Гуз А.М.Організація захисту інформації з обмеженим

доступом : Підручник /[А.М. Гуз, О.Д. Довгань, А.І. Марущак та ін.;за заг. ред. Є.Д.Скулиша]. –К. : Наук.-вид. відділ НА СБ України, 2011. –378 с.

10. Інтернет ресурс: https://quantum-computing.ibm.com/;

11. Anton Z. Dance of the Photons: From Einstein to QuantumTeleportation;

12. Силва В. Разработка с использованием квантовых компьютеров

13. P. W. Shor, ”Quantum Computing,” Documenta Mathematica, vol. Extra Volume ICM, pp. 467- 1486, 1998, http://east.camel.math.ca/ EMIS/ journals/ DMJDMV/ xvol-icm/ 001 Shor.MAN.htm1.

14. P. W. Shor, ”Algorithms for Quanhim Computation: Discrete Logarithms and Factoring,” in Proceedings of the 35th Annual Syrnposiunz on Foundations of Computer Science, pp. 124- 134, 1994.

15. L. K. Grover, ”A fast quantum mechanical algorithm for database search,” in Proceedings of the 28th ACM Symposium on Theory of Computing, pp. 212-219, 1996.

16. L. K. Grover, ”Quantum Mechanical Searching,” in Proceedings ofthe 1999 Congress on Evolutionary Computation, pp. 2255-2261. Jul 1999.

17. L. Spector, H. Bamum, H. J. Bemstein and N. Swamy, ”Finding a Better-than-Classical Quanhim AND/OR Algorithtn using Genetic Programming,” in Proceedings of the 1999 Congress on Evolutionary Computation, pp. 2239-2246, Jul 1999.

18. A. Narayanan and M. Moore, ”Quantum-inspired genetic algorithms,” in Proceedings of IEEE International Conference on Evolutionary Computation, pp. 61-66, 1996.

19. T. Hey, ”Quantum computing: an introduction,” Computing & Control Engineering Journal, pp. 105-1 12, Jun 1999.

20. A. Narayanan, “Quantum computing for beginners,” in Proceedings of the 1999 Congress on Evolutionary Computation, pp. 223 1-2238, Jul 1999.

21. R. Hinterding, ”Representation, Constraint Satisfaction and the Knapsack Problem,” in Proceedings of the 1999 Congress on Evolutionary Computation, pp. 1286-1292, Jul 1999.

22. Michalewicz, Genetic Algorithms i Data Structures Evolution Programs, Springer-Verlag, 3rd, revised and extended edition, 1999.

23. J.-H. Kim and H. Myung, “Evolutionary Programming Techniques for Constrained Optimization Problems,” IEEE Transactions on Evolutionary Computation, Vol. 121 X. Yao, Evolutionary Computation: Theory and Applications, World Scientific, Singapore, 1999.

24. P. W. Shor, Fault-tolerant quantum computation, in Pro c. 37nd Annual Symposium on Foundations of Computer Science, IEEE Computer So ciety Press, Los Alamitos, CA (1996), 56{65.

25. D. R. Simon, On the p o wer of quantum computation, SIAM J. Computing 26 (1997), 14741483.

26. A. Steane, Multiple particle interference and quantum error correction, Proc. Roy. Soc. London Ser. A 452 (1996), 25512577.

27. C. Zalka, Ecient simulation of quantum systems by quantum computers, Pro c. Roy. So c. London Ser. A 454 (1998), 313322.

28. W. H. Zurek, Decoherence and the transition from quantum to classical, Physics To da y 44 (1991) 3644.

29. Tkachuk V., Quantum Genetic Algorithm on Multilevel Quantum Systems, Mathematical Problems in Engineering, V. 2018, Article ID 9127510, 12 pages.

30. Li J.-P., Balazs M.E., Parks G.T., Clarkson P. J., A Species Conserving Genetic Algorithm for Multimodal Function Optimization, Evolutionary Computation, V. 10, N. 3, pp.207-234, 2002.

31. R. Feynman. Quantum mechanical computers. Foundations of Physics, 21(6/7):467-488, 1986.

32. P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In 35th Annual Symposium on Foundation of Computer Science. IEEE Press, 1994.

33. P. H. Winston. Artificial Intelligence (3rd Edition). Addison Wesley, 1992.

34. Nowotniak R. Higher-Order Quantum-Inspired Genetic Algorithms / R. Nowotniak, J. Kucharski // Federated Conference on Annals of Computer Science and Information Systems. — 2014. — P. 465–470.

35. Talbi H. A Quantum-Inspired Evolutionary Algorithm for Multiobjective Image Segmentation / H. Talbi, M. Batouche, A. Draa // International Journal of Mathematical, Physical and Engineering Sciences. — 2007. — Vol. 1, N 2. — P. 109–114.

36. В.М. Курейчик «Перспективні інформаційні технології на основі методів, інспарованих природними системами», УДК 681.3.

37. Grover L.K. A Fast Quantum Mechanical Algorithm for Data-base Search. Proc. 28th Ann. New York: ACM Press, 1996,pp. 212-219.

38. GroverL.K.Synthesis of Quantum Superpositions by Quantum Computation.Physical Rev. Letters. 2000. Vol 85. No.6,pp.1334-1337.

39. Kurejchik V.V., Polypanova E.E. Evolutionary optimization algorithm based on bee colony. UFY journal, technical Sciences ,pp. 41-46

40. R. Nowotniak and J. Kucharski, “Higher-order quantuminspired genetic algorithms,” in Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, FedCSIS 2014, pp. 465–470, Poland, September 2014.

41. Дослідницька стаття: Василій Стефанюк «Квантові генетичні алгоритми у представлення кутріту та їх застосування».

42. AdrianoBarenco, CharlesH. Bennett, RichardCleve, DavidP. DiVincenzo, NormanMargolus, PeterShor, TychoSleator, JohnSmolin, HaraldWeinfurter. Elementary gates for quantum computation. arXiv:9503016v1, 1995

43. Інтернет ресурс: https://qiskit.org/textbook/ch-paper-implementations

44. КовальовС.М., РодзінС.І.Інформаційні технологіх: інтелектуалізація навчання, моделювання еволюції, розпізнання мови.Ростов-на-Дону:ВидавництвоСКНЦВШ, 2002.

45. РодзінС.І. Гібридні інтелектуальні системи на аснові алгоритмів еволюційного програмування // Новиништучного інтелекту.2000. №3. С.159-170.

46. Інтернет ресурс: http://www.mscmga.ms.ic.ac.uk/info.html

47. G. Zhang, “Quantum-inspired evolutionary algorithms: a survey and empirical study,” Journal of Heuristics, vol. 17, pp. 303–351, 2011.

48.

Додати праці керівника

ДОДАТОК А

РОЗРОБЛЕНИЙ КОД МОДИФІКАЦІЇ ГЕНЕТИЧНОГО АЛГОРИТМУ ДЛЯ ПОШУКУ ОПТИМАЛЬНОГО ШЛЯХУ У СЕРЕДОВИЩІ *MATLAB*

```
function varargout = tsp_ga(varargin)

% Initialize default configuration
   defaultConfig.xy         = 10*rand(50,2);
   defaultConfig.dmat       = [];
   defaultConfig.popSize    = 100;
   defaultConfig.numIter    = 1e4;
   defaultConfig.showProg   = true;
   defaultConfig.showResult  = true;
   defaultConfig.showWaitbar = false;

% Interpret user configuration inputs
if ~nargin
      userConfig = struct();
elseif isstruct(varargin{1})
      userConfig = varargin{1};
else
try
         userConfig = struct(varargin{:});
catch
         error('Expected inputs are either a structure or parameter/value pairs');
end
end

% Override default configuration with user inputs
   configStruct = get_config(defaultConfig,userConfig);

% Extract configuration
   xy         = configStruct.xy;
   dmat       = configStruct.dmat;
   popSize    = configStruct.popSize;
   numIter    = configStruct.numIter;
   showProg   = configStruct.showProg;
   showResult  = configStruct.showResult;
   showWaitbar = configStruct.showWaitbar;
if isempty(dmat)
      nPoints = size(xy,1);
      a = meshgrid(1:nPoints);
      dmat = reshape(sqrt(sum((xy(a,:)-xy(a',:)).^2,2)),nPoints,nPoints);
end

% Verify Inputs
   [N,dims] = size(xy);
   [nr,nc] = size(dmat);
if N ~= nr || N ~= nc
```

```
        error('Invalid XY or DMAT inputs!')
end
    n = N;

% Sanity Checks
    popSize     = 4*ceil(popSize/4);
    numIter     = max(1,round(real(numIter(1))));
    showProg    = logical(showProg(1));
    showResult  = logical(showResult(1));
    showWaitbar = logical(showWaitbar(1));

% Initialize the Population
    pop = zeros(popSize,n);
    pop(1,:) = (1:n);
for k = 2:popSize
        pop(k,:) = randperm(n);
end

% Run the GA
    globalMin = Inf;
    totalDist = zeros(1,popSize);
    distHistory = zeros(1,numIter);
    tmpPop = zeros(4,n);
    newPop = zeros(popSize,n);
if showProg
        figure('Name','TSP_GA | Current Best Solution','Numbertitle','off');
        hAx = gca;
end
if showWaitbar
        hWait = waitbar(0,'Searching for near-optimal solution ...');
end
for iter = 1:numIter
% Evaluate Each Population Member (Calculate Total Distance)
for p = 1:popSize
            d = dmat(pop(p,n),pop(p,1)); % Closed Path
for k = 2:n
                d = d + dmat(pop(p,k-1),pop(p,k));
end
            totalDist(p) = d;
end

% Find the Best Route in the Population
        [minDist,index] = min(totalDist);
        distHistory(iter) = minDist;
if minDist < globalMin
            globalMin = minDist;
            optRoute = pop(index,:);
if showProg
% Plot the Best Route
                rte = optRoute([1:n 1]);
if dims > 2, plot3(hAx,xy(rte,1),xy(rte,2),xy(rte,3),'r.-');
else plot(hAx,xy(rte,1),xy(rte,2),'r.-'); end
```

```
            title(hAx,sprintf('Total Cost = %1.2f$, Iteration = %d',minDist*320,iter));
            drawnow;
end
end

% Genetic Algorithm Operators
      randomOrder = randperm(popSize);
for p = 4:4:popSize
          rtes = pop(randomOrder(p-3:p),:);
          dists = totalDist(randomOrder(p-3:p));
          [ignore,idx] = min(dists); %#ok
          bestOf4Route = rtes(idx,:);
          routeInsertionPoints = sort(ceil(n*rand(1,2)));
          I = routeInsertionPoints(1);
          J = routeInsertionPoints(2);
for k = 1:4 % Mutate the Best to get Three New Routes
              tmpPop(k,:) = bestOf4Route;
switch k
case 2 % Flip
                    tmpPop(k,I:J) = tmpPop(k,J:-1:I);
case 3 % Swap
                    tmpPop(k,[I J]) = tmpPop(k,[J I]);
case 4 % Slide
                    tmpPop(k,I:J) = tmpPop(k,[I+1:J I]);
otherwise% Do Nothing
end
end
          newPop(p-3:p,:) = tmpPop;
end
      pop = newPop;

% Update the waitbar
if showWaitbar && ~mod(iter,ceil(numIter/325))
          waitbar(iter/numIter,hWait);
end

end
if showWaitbar
      close(hWait);
end

if showResult
% Plots the GA Results
      figure('Name','TSP_GA | Results','Numbertitle','off');
      subplot(2,2,1);
      pclr = ~get(0,'DefaultAxesColor');
if dims > 2, plot3(xy(:,1),xy(:,2),xy(:,3),'.','Color',pclr);
else plot(xy(:,1),xy(:,2),'.','Color',pclr); end
      title('City Locations');
      subplot(2,2,2);
      imagesc(dmat(optRoute,optRoute));
      title('Distance Matrix');
```

```
        subplot(2,2,3);
        rte = optRoute([1:n 1]);
if dims > 2, plot3(xy(rte,1),xy(rte,2),xy(rte,3),'r.-');
else plot(xy(rte,1),xy(rte,2),'r.-'); end
        title(sprintf('Total Distance = %1.4f',minDist));
        subplot(2,2,4);
        plot(distHistory,'b','LineWidth',2);
        title('Best Solution History');
        set(gca,'XLim',[0 numIter+1],'YLim',[0 1.1*max([1 distHistory])]);
end

% Return Output
if nargout
        resultStruct = struct( ...
'xy',          xy, ...
'dmat',        dmat, ...
'popSize',     popSize, ...
'numIter',     numIter, ...
'showProg',    showProg, ...
'showResult',  showResult, ...
'showWaitbar', showWaitbar, ...
'optRoute',    optRoute, ...
'minDist',     minDist);

        varargout = {resultStruct};
end

end
% Subfunction to override the default configuration with user inputs
function config = get_config(defaultConfig,userConfig)

% Initialize the configuration structure as the default
   config = defaultConfig;

% Extract the field names of the default configuration structure
   defaultFields = fieldnames(defaultConfig);

% Extract the field names of the user configuration structure
   userFields = fieldnames(userConfig);
   nUserFields = length(userFields);

% Override any default configuration fields with user values
for i = 1:nUserFields
        userField = userFields{i};
        isField = strcmpi(defaultFields,userField);
if nnz(isField) == 1
           thisField = defaultFields{isField};
           config.(thisField) = userConfig.(userField);
end
end

end
```

(з координатами x, y):

```
clear all
clc

fileName='Input/1.xlsx'

[A B] = xlsread(fileName, 1);
size = size(A, 1);

xy = double.empty(0, 2);
disp(size);
for i = 1:size
   xy(i, 1) = A(i, 1);
   xy(i, 2) = A(i, 2);
end

userConfig = struct('xy',xy,'NUMITER', 5000);
resultStruct = tsp_ga(userConfig);
```

(з координатами x, y, z):

```
clear all
clc

fileName='Input/2.xlsx'

[A B] = xlsread(fileName, 1);
size = size(A, 1);

xyz = double.empty(0, 3);
disp(size);
for i = 1:size
   xyz(i, 1) = A(i, 1);
   xyz(i, 2) = A(i, 2);
   xyz(i, 3) = A(i, 3);
end

userConfig = struct('xy',xyz,'NUMITER', 5000);
resultStruct = tsp_ga(userConfig);
```

ДОДАТОК Б

РОЗРОБЛЕНИЙ КОД МОДИФІКАЦІЇ ГЕНЕТИЧНОГО АЛГОРИТМУ ДЛЯ ПОШУКУ ОПТИМАЛЬНОГО ШЛЯХУ НА PYTHON

```
from deap import base
from deap import creator
from deap import tools
from deap import algorithms

import random
import array
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tsp

# set the random seed for repeatable results
RANDOM_SEED = 42
random.seed(RANDOM_SEED)

# create the desired traveling salesman problem instace:
TSP_NAME = "bayg29"  # name of problem
tsp = tsp.TravelingSalesmanProblem(TSP_NAME)

# Genetic Algorithm constants:
POPULATION_SIZE = 300
MAX_GENERATIONS = 200
HALL_OF_FAME_SIZE = 1
P_CROSSOVER = 0.9  # probability for crossover
P_MUTATION = 0.1   # probability for mutating an individual

toolbox = base.Toolbox()
# define a single objective, minimizing fitness strategy:
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))

# create the Individual class based on list of integers:
creator.create("Individual", array.array, typecode='i', fitness=creator.FitnessMin)

# create an operator that generates randomly shuffled indices:
toolbox.register("randomOrder", random.sample, range(len(tsp)), len(tsp))

# create the individual creation operator to fill up an Individual instance with shuffled indices:
toolbox.register("individualCreator", tools.initIterate, creator.Individual, toolbox.randomOrder)

# create the population creation operator to generate a list of individuals:
toolbox.register("populationCreator", tools.initRepeat, list, toolbox.individualCreator)

# fitness calculation - compute the total distance of the list of cities represented by indices:
def tpsDistance(individual):
```

```
return tsp.getTotalDistance(individual),  # return a tuple
toolbox.register("evaluate", tpsDistance)


# Genetic operators:
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("mate", tools.cxOrdered)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=1.0/len(tsp))

# Genetic Algorithm flow:
def main():
# create initial population (generation 0):
population = toolbox.populationCreator(n=POPULATION_SIZE)
# prepare the statistics object:
stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("min", np.min)
    stats.register("avg", np.mean)
# define the hall-of-fame object:
hof = tools.HallOfFame(HALL_OF_FAME_SIZE)
# perform the Genetic Algorithm flow with hof feature added:
population, logbook = algorithms.eaSimple(population, toolbox, cxpb=P_CROSSOVER,
mutpb=P_MUTATION,
ngen=MAX_GENERATIONS, stats=stats, halloffame=hof, verbose=True)
# print best individual info:
best = hof.items[0]
print("-- Best Ever Individual = ", best)
print("-- Best Ever Fitness = ", best.fitness.values[0])
# plot best solution:
plt.figure(1)
    tsp.plotData(best)
# plot statistics:
minFitnessValues, meanFitnessValues = logbook.select("min", "avg")
    plt.figure(2)
    sns.set_style("whitegrid")
    plt.plot(minFitnessValues, color='red')
    plt.plot(meanFitnessValues, color='green')
    plt.xlabel('Generation')
    plt.ylabel('Min / Average Fitness')
    plt.title('Min and Average fitness over Generations')
# show both plots:
plt.show()
if __name__ == "__main__":
    main()
```

ДОДАТОК В
РОЗРОБЛЕНИЙ КОД МОДИФІКАЦІЇ КВАНТОВОГО ГЕНЕТИЧНОГО АЛГОРИТМУ ДЛЯ ПОШУКУ ОПТИМАЛЬНОГО ШЛЯХУ НА *PYTHON* ТА СИМУЛЯТОРІ *QASM*

(Частково код є власністю розробок Qiskit)

```
def _num_to_latex(num, precision=5):
r = np.real(num)
i = np.imag(num)
common_factor = None
ifnp.isclose(abs(r), abs(i)) andnotnp.isclose(r, 0) andnotnp.isclose(i, 0):
common_factor = abs(r)
r = r / common_factor
i = i / common_factor
common_terms = {
        1 / math.sqrt(2): "\\tfrac{1}{\\sqrt{2}}",
        1 / math.sqrt(3): "\\tfrac{1}{\\sqrt{3}}",
math.sqrt(2 / 3): "\\sqrt{\\tfrac{2}{3}}",
math.sqrt(3 / 4): "\\sqrt{\\tfrac{3}{4}}",
        1 / math.sqrt(8): "\\tfrac{1}{\\sqrt{8}}",
    }
def _proc_value(val):
val_mod = np.mod(val, 1)
ifnp.isclose(val_mod, 0) ornp.isclose(val_mod, 1):
returnstr(int(np.round(val))
forterm, latex_strincommon_terms.items():
ifnp.isclose(abs(val), term):
ifval> 0:
returnlatex_str
else:
return "-" + latex_str
frac = Fraction(val).limit_denominator()
```

```
num, denom = frac.numerator, frac.denominator
ifnum + denom< 20:
ifval> 0:
returnf"\\tfrac{{{abs(num)}}}{{{abs(denom)}}}"
            else:
                return f"-\\tfrac{{{abs(num)}}}{{{abs(denom)}}}"
        else:
            return "{:.{}f}".format(val, precision).rstrip("0")
    if common_factor is None:
        common_facstring = None
    else:
        common_facstring = _proc_value(common_factor)
    realstring = _proc_value(r)
    if i > 0:
        operation = "+"
        imagstring = _proc_value(i)
    else:
        operation = "-"
        imagstring = _proc_value(-i)
    if imagstring == "1":
        imagstring = ""  # Don't want to return '1i', just 'i'
    if imagstring == "0":
        return realstring  # realstring already contains the negative sign (if needed)
    if realstring == "0":
        if operation == "-":
            return f"-{imagstring}i"
        else:
            return f"{imagstring}i"
    if common_facstring is not None:
        return f"{common_facstring}({realstring} {operation} {imagstring}i)"
    else:
        return f"{realstring} {operation} {imagstring}i"
```

```
def _matrix_to_latex(matrix, precision=5, prefix="", max_size=(8, 8)):
    if min(max_size) < 3:
        raise ValueError("""Smallest value in max_size must be greater than or equal to 3""")
    out_string = f"\n{prefix}\n"
    out_string += "\\begin{bmatrix}\n"

    def _elements_to_latex(elements):
        # Takes a list of elements (a row) and creates a latex
        # string from it; Each element separated by `&`
        el_string = ""
        for el in elements:
            num_string = _num_to_latex(el, precision=precision)
            el_string += num_string + " & "
        el_string = el_string[:-2]  # remove trailing ampersands
        return el_string

    def _rows_to_latex(rows, max_width):
        # Takes a list of lists (list of 'rows') and creates a
        # latex string from it
        row_string = ""
        for r in rows:
            if len(r) <= max_width:
                row_string += _elements_to_latex(r)
            else:
                row_string += _elements_to_latex(r[: max_width // 2])
                row_string += "& \\cdots & "
                row_string += _elements_to_latex(r[-max_width // 2 + 1 :])
            row_string += " \\\\\n "
        return row_string
    max_width, max_height = max_size
    if matrix.ndim == 1:
        out_string += _rows_to_latex([matrix], max_width)
```

```
    elif len(matrix) > max_height:
        out_string += _rows_to_latex(matrix[: max_height // 2], max_width)
        if max_width >= matrix.shape[1]:
            out_string += "\\vdots & " * matrix.shape[1]
        else:
            # In this case we need to add the diagonal dots in line with the column
            # of horizontal dots
            pre_vdots = max_width // 2
            post_vdots = max_width // 2 + np.mod(max_width, 2) - 1
            out_string += "\\vdots & " * pre_vdots
            out_string += "\\ddots & "
            out_string += "\\vdots & " * post_vdots
        out_string = out_string[:-2] + "\\\\\n "
        out_string += _rows_to_latex(matrix[-max_height // 2 + 1 :], max_width)
    else:
        out_string += _rows_to_latex(matrix, max_width)
    out_string += "\\end{bmatrix}\n"
    return out_string


def array_to_latex(array, precision=5, prefix="", source=False, max_size=8):
    try:
        array = np.asarray(array)
        _ = array[0] + 1  # Test first element contains numerical data
    except TypeError as err:
        raise TypeError(
            """array_to_latex can only convert numpy arrays containing numerical data,
        or types that can be converted to such arrays"""
        ) from err

    if array.ndim<= 2:
        if isinstance(max_size, int):
            max_size = (max_size, max_size)
        outstr = _matrix_to_latex(array, precision=precision, prefix=prefix, max_size=max_size)
```

```
    else:
        raise ValueError("array_to_latex can only convert numpy ndarrays of dimension 1 or 2")
    if source is False:
        try:
            from IPython.display import Latex
        except ImportError as err:
            raise MissingOptionalLibraryError(
                libname="IPython",
                name="array_to_latex",
                pip_install="pip install ipython",
            ) from err
        return Latex(f"$${outstr}$$")
else:
return outstr
```

ДОДАТОК Г

ТЕСТОВІ ФУНКЦІЇ НА МОВІ *PYTHON*

```
defdraw_pic(X, Y, Z, z_max, title, z_min=0):
    fig = plt.figure()
    ax = Axes3D(fig)
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=plt.cm.hot)
    ax.set_zlim(z_min, z_max)
ax.set_title(title)
plt.show()

defget_X_AND_Y(X_min, X_max, Y_min, Y_max):
    X = np.arange(X_min, X_max, 0.1)
    Y = np.arange(Y_min, Y_max, 0.1)
    X, Y = np.meshgrid(X, Y)
return X, Y

# Растригинтестова
defRastrigin(X_min = -5.52, X_max = 5.12, Y_min = -5.12, Y_max = 5.12):
    A = 10
    X, Y = get_X_AND_Y(X_min, X_max, Y_min, Y_max)
    Z = 2 * A + X ** 2 - A * np.cos(2 * np.pi * X) + Y ** 2 - A * np.cos(2 * np.pi * Y)
return X, Y, Z, 100, "Rastrigin function"

# Экли
defЕклі(X_min = -5, X_max = 5, Y_min = -5, Y_max = 5):
    X, Y = get_X_AND_Y(X_min, X_max, Y_min, Y_max)
    Z = -20 * np.exp(-0.2 * np.sqrt(0.5 * (X**2 + Y**2))) - \
np.exp(0.5 * (np.cos(2 * np.pi * X) + np.cos(2 * np.pi * Y))) + np.e + 20
return X, Y, Z, 15, "Еклі function"

# Перевірки сфери
defSphere(X_min = -3, X_max = 3, Y_min = -3, Y_max = 3):
    X, Y = get_X_AND_Y(X_min, X_max, Y_min, Y_max)
    Z = X**2 + Y**2
return X, Y, Z, 20, "Sphere function"

z_min = None
# X, Y, Z, z_max, title = Rastrigin()
# X, Y, Z, z_max, title = Еклі()
# X, Y, Z, z_max, title = Sphere()
# X, Y, Z, z_max, title = Beale()
X, Y, Z, z_max, title = Booth()
# X, Y, Z, z_max, title = Bukin()
# X, Y, Z, z_max, title = three_humpCamel()
# X, Y, Z, z_max, title, z_min = Holder_table()

draw_pic(X, Y, Z, z_max, title, z_min)
```

ДОДАТОК Д

СЕРТИФІКАТ SCIENTIA