KAPITEL 6 / *CHAPTER 6* [6]
APPLICATION OF HEURISTIC ALGORITHMS IN SOLVING NP-COMPLEX PROBLEMS ON THE EXAMPLE OF THE TRAVELING SALESMAN PROBLEM

DOI: 10.30890/2709-2313.2023-25-00-016

Вступ

Значення рішення задачі комівояжера (ЗК) на сучасному етапі є надзвичайно важливим. Ця задача є однією з найважливіших задач у теорії графів і комбінаторній оптимізації. Вона має широке застосування в різних сферах діяльності людини, зокрема, в логістиці, економіці, комп'ютерних науках та інших.

В логістиці ЗК використовується для оптимізації маршрутів доставки товарів або послуг. Наприклад, вона може бути використана для пошуку найкоротшого маршруту для доставки вантажів між складами, магазинами або клієнтами. Це дозволяє заощадити час і гроші на логістичних витратах.

В економіці ЗК може бути використана для оптимізації розподілу ресурсів. Наприклад, вона може бути використана для пошуку найвигіднішого способу розподілу товарів між торговими точками. Це дозволяє підвищити прибутковість і ефективність бізнесу.

В комп'ютерних науках ЗК використовується для тестування алгоритмів оптимізації. Вона є хорошим прикладом складної задачі, для якої існує багато ефективних алгоритмів.

В інших сферах діяльності ЗК також має широке застосування. Наприклад, вона може бути використана для оптимізації маршрутів обслуговування клієнтів, маршрутів подорожей, розкладів польотів і т. д.

На сучасному етапі розроблено багато ефективних алгоритмів для вирішення задачі комівояжера. Однак, задача залишається NP-повною, що означає, що для її вирішення в загальному випадку необхідний час, який експоненціально зростає з числом міст. Тому, для вирішення задачі комівояжера

[6] **Authors:** *Skakalina Olena Viktorivna*

для великих наборів даних зазвичай використовують генетичні алгоритми.

Значення рішення задачі комівояжера можна виділити в таких сферах:

- Транспорт: Задача комівояжера широко застосовується в транспортній логістиці для оптимізації маршрутів вантажних перевезень. Це дозволяє скоротити витрати на транспортування, підвищити ефективність використання транспортних засобів та зменшити негативний вплив на навколишнє середовище.
- Збут: Задача комівояжера використовується для оптимізації маршрутів дистрибуції товарів. Це дозволяє компаніям скоротити витрати на доставку, поліпшити обслуговування клієнтів та підвищити рівень продажів.
- Виробництво: Задача комівояжера використовується для оптимізації маршрутів доставки сировини та готової продукції на підприємствах. Це дозволяє компаніям скоротити витрати на виробництво, підвищити ефективність використання ресурсів та зменшити виробничі відходи.

Крім того, задача комівояжера застосовується в інших сферах, таких як:

- Інформаційні технології: Задача комівояжера використовується для оптимізації маршрутів доставки даних у комп'ютерних мережах.
- Сервіс: Задача комівояжера використовується для оптимізації маршрутів обслуговування клієнтів.
- Розваги: Задача комівояжера використовується в деяких іграх, таких як шахи та го.

Рішення задачі комівояжера є складною математичною проблемою. Однак, завдяки розвитку комп'ютерних технологій, розроблено ефективні алгоритми, які дозволяють знаходити практично оптимальні рішення для задач з великою кількістю міст.

Розвиток комп'ютерних технологій дозволив розробити ефективні алгоритми вирішення задачі комівояжера. Це дозволило використовувати цю задачу в багатьох сферах діяльності людини, де раніше її застосування було неможливим.

6.1. Аналіз проблеми, постановка задачі дослідження

Задачею про комівояжера визначається задача з математичного програмування за пошуком найкоротшого шляху руху мандрівного торговця (комівояжера), ціль якого зводиться до того, щоб відвідати всі об'єкти, залучені до задачі, за найменший термін і з мінімальними витратами. В теорії графів - це знаходження маршруту, який поєднує два або понад вузлів, при використанні відповідного критерію оптимальності. Завдання бродячого торговця полягає в пошуку найбільш оптимального шляху, що пролягає через помічені міста хоча б по одному разу. В умовах задачі зазначається показник вигідності маршруту (найдешевший, найбільш короткий, найменш часозатратний і т. п.) і відповідні матриці вартості, відстаней і т. п. Як правило вказується, що шлях має включати кожне місто маршруту тільки один раз, за такої умови вибір виконується поміж так званих гамільтонових циклів. Завдання мандрівного торговця - traveling salesman problem (TSP) – це NP-складна задача дискретної оптимізації. Для її рішення відсутні швидкі поліноміальні алгоритми. В термінах теорії графів ця задача визначається в такий спосіб: слід знайти найкоротший шлях, що проходить через визначені вузли графа хоча б по одному разу з подальшим поверненням до вузла на початку маршруту [1].

6.2. Класифікація алгоритмів комбінаторної оптимізації

Завдання комівояжера (або TSP від англ. Travelling salesman problem) - одна з найвідоміших задач комбінаторної оптимізації, що полягає в пошуку са-мого вигідного маршруту, що проходить через зазначені міста хоча б по одному разу з наступним поверненням в початковий місто.

Вперше проблема TSP була сформульована в 1930 році і собою в сього-денні в інформаційно-комунікаційних технологіях однією з найбільш вивчених проблем оптимізації [2-8]. Незважаючи на те, що проблема обчислювально

складна (відноситься до класу NP-складних), відомо багато евристичних і точних алгоритмів, за допомогою яких розв'язувались практичні завдання.

Дослідження по розв'язанню задачі комівояжера можна розділити умовно на два напрями [2-8].

1. Розробка точних алгоритмів, які працюють досить швидко лише для невеликих розмірі-8в задач;

2. Розробка "неоптимальних" або евристичних алгоритмів, які забезпечують апроксимовані рішення за розумний час.

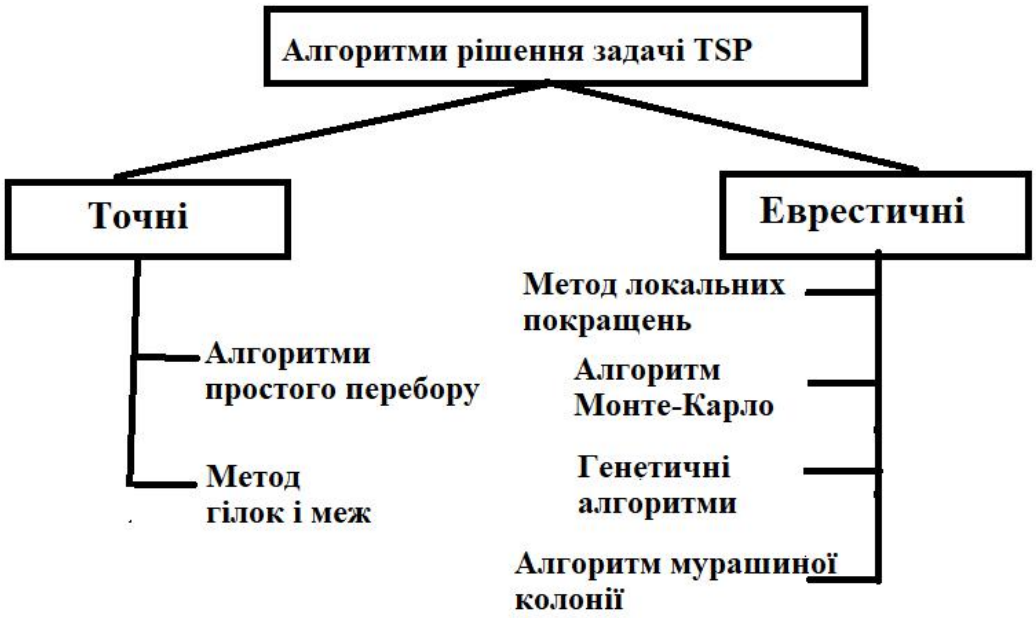

Рисунок 1– Алгоритми рішення задачі TSP

Можна виділити ще пошук особливих випадків для проблеми TSP ("під-задачі"), для яких можливі або краща, або точна евристика.

Важливо також розрізняти задачі TSP симетричні та асиметричні. Для симетричного випадку (зазвичай називається просто TSP), для всіх відстаней у D рівність $d_{ij} = d_{ji}$ виконується, тобто не має значення, рухаємось від і до j або навпаки, відстань однакова. У асиметричному випадку (званому ATSP) відстані не рівні для всіх пар міст. Проблеми такого роду виникають, коли ми не маємо справи з просторовими відстанями між містами, але, наприклад, з вартістю або необхідним часом, пов'язаним з поїздкою між місця, де ціна на квиток на літак

між двома містами може бути різною. В даному разі розглянемо симетричний варіант задачі TSP.

Цілком очевидно, що завдання може бути вирішена перебором всіх варіантів пересувань комівояжера і вибором оптимального. Вся річ в тому, що кількість можливих маршрутів N дуже швидко зростає з ростом кількості для відвідування міст n N=n!.

Наприклад, для n = 100 кількість варіантів буде представлятися 158-значним числом. Сучасна ЕОМ, здатна перебирати мільйон операцій в секунду, буде битися з завданням протягом приблизно 144 років.

Вважається доведеним, що не існує точного алгоритму рішення TSP, що має поліноміальну складність (тобто має асимптотичну оцінку часу виконання алгоритму $T(n) = O(n^{\alpha})$) трудомісткості виконання. Отже, задача TSP для скільки-небудь великих n стає майже нерозв'язною для точних алгоритмів.

TSP - проблема, яка важка для NP, і її так легко описати, а так складно вирішити. В такому випадку слід відмовитися від спроб відшукати точне рішення задачі комівояжера і зосередитися на пошуку наближеного - нехай не оптимальні, але хоча б близького до нього. З причини великої практичної важливості завдання корисними будуть і наближені рішення [2-8].

Еволюційний алгоритм (EA) - це алгоритм, вперше поданий Чарльзом Дарвіном у 1859 році. Він забезпечує рішення для різних проблеми оптимізації. Він копіює процес еволюції, що відбувається в природі, тобто мутація, рекомбінація та Природний відбір. GA (Генетичний алгоритм) - це тип еволюційного алгоритму. GA забезпечує вирішення проблеми в форму рядків чисел і застосовує та операторі, як мутація та рекомбінація. Це починається з початкового та за допомогою цих операторів знаходить найбільш підходяще покоління населення.

Алгоритм ACO (Оптимізація колонії мурашок) - це також евристичний алгоритм, вперше знайдений Марко Доріго, який забезпечує оптимальне рішення, імітуючи спосіб, яким мурахі знаходять їжу. ACO - це тип CI (інтелект роїв) техніка.

Алгоритм PSO (Рой частинок), вперше знайдений Кеннеді та Еберхартом, також забезпечує оптимальне рішення для проблеми і надихається зграєю птахів, риб та стадами тварин. Це імітує те, як вони знаходять собі їжу середовища і дотримується підходу до обміну інформацією.

Окрім представлення TSP, як задачі комбінаторної оптимізації, TSP також може бути сформульована як теоретична задача теорії графів.

Представимо TSP в формулюванні задачі теорії графів. Теорія графів визначає проблему як знаходження гамільтонового циклу з найменшою вагою для заданого повного зваженого графа. Такого роду постановка задачі широко поширена в інженерних програмах та деяких промислових проблемах, таких як машинне планування, стільникове виробництво та проблеми призначення частоти можуть бути сформульовані як TSP.

Нехай задано зважений граф G = (V, E), в якому міста відповідають множині вершин V = {1, 2,. . . , n} і кожне ребро $e_i \in E$ має відповідну вагу w_i, що представляє відстань між містами, які вона з'єднує. Якщо граф не є повним, відсутні ребра можна замінити ребрами з дуже великими відстанями.

Метою рішення задачі TSP є пошук гамільтонового циклу, тобто циклу, який відвідує кожен вузол на графі рівно один раз, з найменшою можливою вагою для заданого графа. Це формулювання природно призводить до процедур, що передбачають пошук мінімальних каркасів дерев для заданого графа.

TSP також можуть бути представлені як цілочисельні та лінійні програми програмування. Формулювання цілочисельного програмування (IP) базується на задачі присвоєння з додаткове обмеження відсутності підтурів [9].

Метод гілок і меж — це алгоритм дискретної оптимізації, який використовується для пошуку оптимального рішення задачі, в якій можливі лише дискретні значення змінних. Метод працює за рахунок побудови дерева рішень, в якому кожний вузол представляє можливий варіант рішення задачі. Метод гілок і меж починається з кореневого вузла дерева, який представляє початкове рішення задачі. Потім, алгоритм використовує критерій відсікання, щоб відкинути вузли, які не можуть містити оптимального рішення. Критерій

відсікання є функцією, яка оцінює ймовірність того, що вузол містить оптимальне рішення [10].

Якщо вузол не відкидається, то алгоритм розгалужує його на два нових вузла, які представляють два можливих варіанти продовження рішення задачі. Потім, алгоритм повторює цей процес для кожного з нових вузлів.

Процес розгалуження продовжується до тих пір, поки не буде знайдено оптимальне рішення задачі або поки не буде обстежено все дерево рішень.

Метод гілок і меж є ефективним алгоритмом для вирішення багатьох задач дискретної оптимізації. Однак, він може бути неефективним для задач з дуже великими деревами рішень [10].

Ось приклад того, як метод гілок і меж можна використовувати для вирішення задачі комівояжера. Задача комівояжера полягає в тому, щоб знайти найкоротший маршрут, який проходить через всі міста, відвідувані комівояжером.

Для вирішення цієї задачі ми можемо побудувати дерево рішень, в якому кожний вузол представляє можливий маршрут комівояжера. Потім, ми можемо використовувати критерій відсікання, щоб відкинути вузли, які не можуть містити оптимальний маршрут.

Одним із критеріїв відсікання, який можна використовувати для задачі комівояжера, є наступний. Якщо довжина маршруту від вузла до його батьківського вузла більше або дорівнює довжині найкоротшого відомого маршруту, то вузол можна відкинути.

Цей критерій відсікання працює, тому що, якщо довжина маршруту від вузла до його батьківського вузла більше або дорівнює довжині найкоротшого відомого маршруту, то цей вузол не може бути продовженням найкоротшого відомого маршруту.

Використовуючи цей критерій відсікання, ми можемо обмежити розмір дерева рішень, яке потрібно обстежити. Це може значно покращити ефективність алгоритму.

Метод гілок і меж використовується для вирішення багатьох задач

дискретної оптимізації, включаючи:

- Задача комівояжера
- Задача про максимальний потік
- Задача про розміщення
- Задача про розклад
- Задача про рюкзак
- Задача про призначення

Метод гілок і меж є потужним інструментом для вирішення задач дискретної оптимізації. Він може бути використаний для вирішення широкого кола задач, включаючи задачі з дуже складними обмеженнями [10].

Алгоритм Дейкстри - широко використовуваний алгоритм для пошуку найкоротшого шляху в зваженому графі від однієї вершини до всіх інших. Він був розроблений Едсгером Дейкстрою в 1959 році і відрізняється своєю простотою та ефективністю.

Принцип роботи:

1. Вхідні дані:

Зважений граф G = (V, E), де V - множина вершин, а E - множина ребер, кожному ребру присвоєна вага (ціна).

Початкова вершина s, з якої потрібно знайти найкоротші шляхи.

2. Ініціалізація:

Для кожної вершини v в G зберігаємо два значення:

dist[v]: оціночна відстань від початкової вершини s до v (спочатку нескінченна, крім dist[s] = 0).

prev[v]: попередник v на найкоротшому шляху (спочатку None).

3. Основний цикл:

•Вибираємо невідвідану вершину v з мінімальною оцінкою dist[v].

•Позначаємо v як відвідану.

•Для кожного невідвіданого сусіда w вершини v:

•new_dist = dist[v] + weight(v, w): розраховуємо оціночну відстань до w через v.

• Якщо new_dist < dist[w]: оновлюємо оціночну відстань dist[w] та попередник prev[w] на v.

4. Вихід:

dist[v] містить відстань від початкової вершини s до кожної вершини v.

prev[v] дозволяє відновити найкоротший шлях від s до v, рухаючись по попередниках.

Переваги:

• Простота в реалізації.

• Гарантовано знаходить найкоротший шлях.

• Ефективний для графів с невід'ємними вагами.

Недоліки:

• Неефективний для графів с великими від'ємними вагами .

• Не може знайти всі найкоротші шляхи, якщо граф має цикли с від'ємними вагами .

Алгоритми комбінаторної оптимізації класифікуються за таким показником як точність отримання рішення (рис. 2).

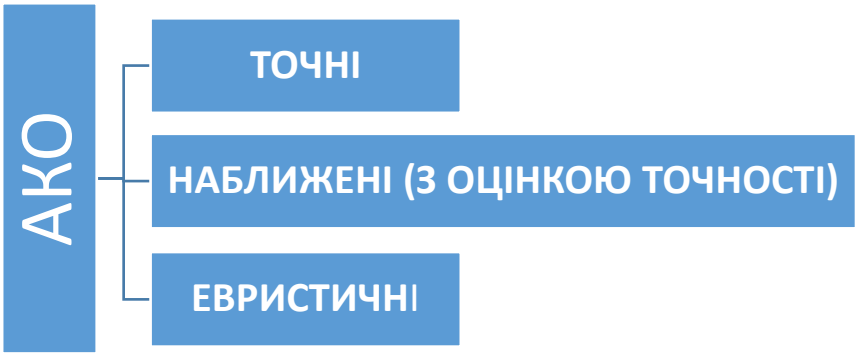

Рисунок 2 – Класифікація алгоритмів комбінаторної оптимізації за точністю

6.3. Еволюційне направлення в задачах оптимізації

Еволюційне направлення в штучному інтелекті – це метод навчання машин, який використовує ідеї еволюції природних систем. Цей метод передбачає створення популяції агентів, які конкурують між собою за ресурси. Агенти, які краще справляються з даним завданням, отримують більші ресурси і мають більшу ймовірність вижити та передати свої гени наступному поколінню. Цей процес поступово призводить до того, що популяція агентів стає більш ефективною в даному завданні [11].

Еволюційне направлення може бути використано для навчання різних видів агентів, включаючи нейронні мережі, популяційні алгоритми та генетичні алгоритми. Цей метод може бути особливо ефективним для навчання агентів, які повинні вирішувати складні завдання або адаптуватися до змінних умов навколишнього середовища.

Одним із прикладів застосування еволюційного направлення в штучному інтелекті є розробка роботів, які можуть ходити по нерівній місцевості. Еволюційні алгоритми можуть бути використані для створення популяції роботів, які мають різні параметри, такі як довжина ніг, кути колін та кути стоп. Роботи, які краще справляються з ходьбою по нерівній місцевості, отримують більші ресурси і мають більшу ймовірність вижити та передати свої гени наступному поколінню. Цей процес поступово призводить до того, що популяція роботів стає більш ефективною в ходьбі по нерівній місцевості.

Іншим прикладом застосування еволюційного направлення в штучному інтелекті є розробка систем машинного перекладу. Еволюційні алгоритми можуть бути використані для створення популяції систем машинного перекладу, які мають різні параметри, такі як методи перекладу, словники та правила граматики. Системи машинного перекладу, які краще переводять текст з однієї мови на іншу, отримують більші ресурси і мають більшу ймовірність вижити та передати свої гени наступному поколінню. Цей процес поступово призводить до того, що популяція систем машинного перекладу стає більш ефективною в

перекладі тексту [11].

Інші методи еволюційного направлення в штучному інтелекті :

- Еволюційна стратегія
- Еволюційна оптимізація
- Еволюційні алгоритми навчання

Еволюційне направлення є перспективним методом навчання машин, який має потенціал для вирішення широкого кола завдань. Цей метод може бути особливо ефективним для навчання агентів, які повинні вирішувати складні завдання або адаптуватися до змінних умов навколишнього середовища [11].

6.4. Генетичні алгоритми як евристичний напрямок

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 *(reproductive plan)* 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 [15].

На даному етапі розвитку будь-яка певна стратегія побудови рішення відсутня, існує величезна кількість окремих реалізованих алгоритмів, не дуже схожих один на одного. Але все ж роботу всіх цих алгоритмів можна представити у вигляді традиційної схеми роботи даних алгоритмів (рис.3).

Критеріїв зупинки пошуку рішення може бути декілька: часові рамки, кількість створених популяцій та зменшення покращення фітнес-функції у порівнянні з попередніми ітераціями.

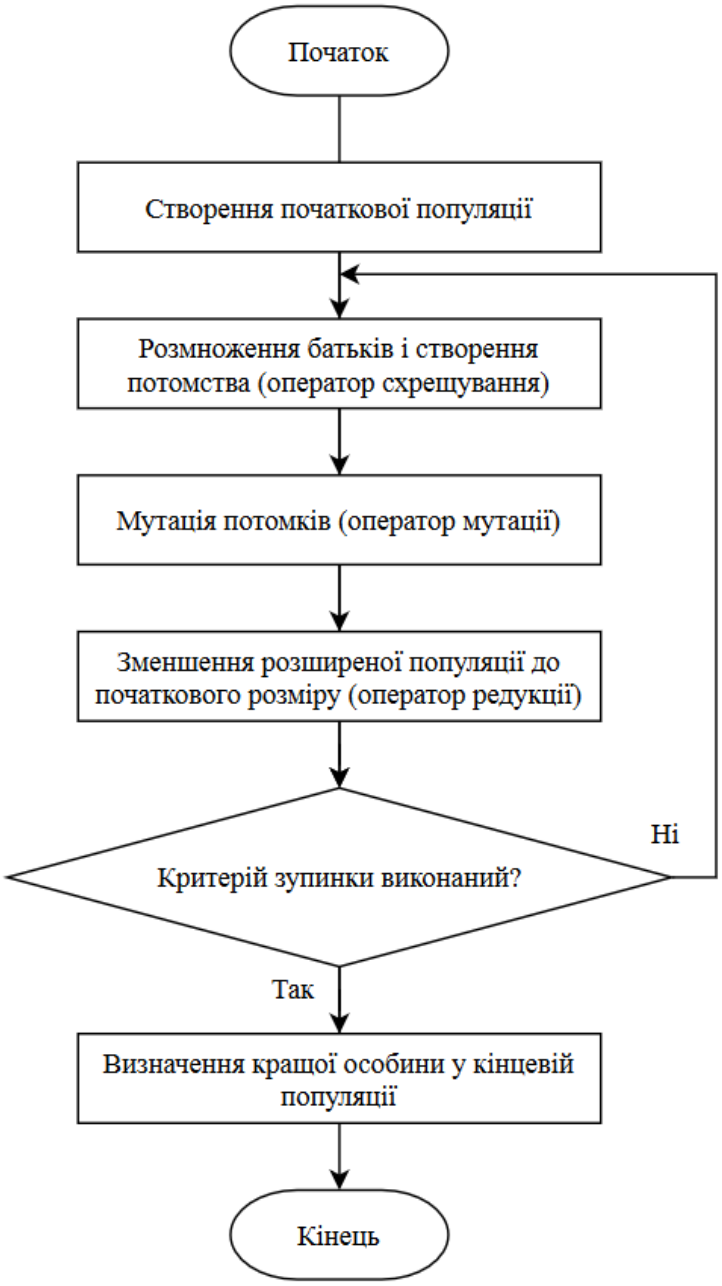

Рисунок 3– Узагальнена схема генетичних алгоритмів

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 [16]. 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.

6.5. Алгоритм мурашиної колонії як евристичний напрямок

Поведінка колонії мурах забезпечує досягнення загальних цілей на основі низькорівневої взаємодії. Колонія не має централізованого управління, і обмін інформацією між її особами відбувається локально, тобто між окремими її представниками, або ж непрямим обміном, який лежить в основі мурашиних алгоритмів. Непрямий обмін є взаємодією між мурахами шляхом зміни деякої галузі навколишнього середовища за допомогою спеціальної хімічної речовини - феромону - секрету спеціальних залоз, що відкладаються мурахами при переміщенні у просторі. Концентрація феромону на шляху визначає перевагу руху по даному напрямку. Адаптивність поведінки мурах реалізується випаром феромону в просторі протягом деякого часу. Тому ми можемо провести деяку

аналогію між розподілом феромону у навколишньому колонію простору, і «глобальною» пам'яттю мурашника, має динамічний характер.

Ідея мурашиного алгоритму полягає у моделюванні поведінки мурах, пов'язаного з їхньою здатністю знаходити найкоротший шлях від мурашника до джерела їжі і адаптуватися до умов, що змінюються, знаходячи новий найкоротший шлях. При своєму русі мураха мітить свій шлях феромоном, і ця інформація використовується іншими мурахами. Це і є основним правилом поведінки кожного з представників колонії у разі, якщо старий маршрут став недоступним. Якщо під час руху мурашка зіткнулася з перепоною, то з рівною ймовірністю вона обійде її ліворуч чи праворуч. Те саме відбудеться і на зворотному шляху. Однак ті мурахи, які оберуть найкоротший шлях, проходитимуть його швидше і за кілька переміщень він буде сильнішим збагачений феромоном. Оскільки його кількість впливає на вибір шляху мурахами, то незабаром переважна більшість представників колонії віддаватиме перевагу цьому маршруту. Однак випаровування пахучого сліду в повітрі гарантує, що знайдене локально-оптимальне рішення не буде єдиним, і мурахи продовжуватимуть шукати й інші шляхи. Якщо ми моделюємо процес такої поведінки на графі, ребра якого - всілякі шляхи руху, то протягом деякого часу найбільш збагачений феромоном шлях і буде найкоротшим, що дасть вирішення завдання [21].

6.6. Вирішення задачі за допомогою мурашиного алгоритму

Умова задачі. Необхідно знайти найкоротший маршрут, що починається в стартовому місті і закінчується в ньому. Маршрут повинен проходити всі міста тільки один раз.

Мурашиний алгоритм буде реалізовано для знаходження найкоротшого маршруту об’їзду всіх обласних центрів України, тобто 25 міст, початок шляху знаходиться в місті Вінниця. В таблиці 1 наведено відстані між містами

дистрибуції в кілометрах.

Таблиця 1 – Відстані між обласними центрами України

	Місто	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	Вінниця	-	645	868	125	748	366	256	316	1057	382	360	471	428	593	311	844	602	232	575	734	521	120	343	312	396
2	Дніпропетровськ	645	-	252	664	81	901	533	294	394	805	975	343	468	196	957	446	430	877	1130	213	376	765	324	891	672
3	Донецьк	868	252	-	858	217	1171	727	520	148	1111	1221	611	731	390	1045	591	706	1100	1391	335	560	988	547	1141	867
4	Житомир	125	664	858	-	738	431	131	407	1182	257	423	677	557	468	187	803	477	298	671	690	624	185	321	389	271
5	Запоріжжя	748	81	217	738	-	1119	607	303	365	681	833	377	497	270	925	365	477	977	1488	287	297	875	405	957	747
6	Ів-Франківськ	366	901	1171	431	1119	-	561	618	1402	328	135	747	627	898	296	1070	908	134	280	1040	798	246	709	143	701
7	Київ	256	533	727	131	607	561	-	298	811	388	550	490	489	337	318	972	346	427	806	478	551	315	190	538	149
8	Кіровоград	316	294	520	407	303	618	298	-	668	664	710	174	294	246	627	570	506	547	883	387	225	435	126	637	363
9	Луганськ	1057	394	148	1182	365	1402	811	668	-	1199	1379	857	977	474	1129	739	253	1289	1539	333	806	1177	706	1292	951
10	Луцьк	382	805	1111	257	681	328	388	664	1199	-	152	780	856	725	70	1052	734	159	413	866	869	263	578	336	949
11	Львів	360	975	1221	423	833	135	550	710	1379	152	-	850	970	891	232	1173	896	128	261	1028	1141	240	740	278	690
12	Миколаїв	471	343	611	677	377	747	490	174	857	780	850	-	120	420	864	282	681	754	999	556	51	590	300	642	640
13	Одеса	428	468	731	557	497	627	489	294	977	856	970	120	-	540	741	392	800	660	1009	831	171	548	420	515	529
14	Полтава	593	196	390	468	270	898	337	246	474	725	891	420	540	-	665	635	261	825	1149	141	471	653	279	892	477
15	Рівне	311	957	1045	187	925	296	318	627	1129	70	232	864	741	665	-	1157	664	162	484	805	834	193	508	331	458
16	Сімферополь	844	446	591	803	365	1070	972	570	739	1052	1173	282	392	635	1157	-	896	1097	1363	652	221	964	696	981	1112
17	Суми	602	430	706	477	477	908	346	506	253	734	896	681	800	261	664	896	-	774	1138	190	732	662	540	883	350
18	Тернопіль	232	877	1100	298	977	134	427	547	1289	159	128	754	660	825	162	1097	774	-	338	987	831	112	575	176	568
19	Ужгород	575	1130	1391	671	1488	280	806	883	1539	413	261	999	1009	1149	484	1363	1138	338	-	1299	1065	455	984	444	951
20	Харків	734	213	335	690	287	1040	478	387	333	866	1028	556	831	141	805	652	190	987	1299	-	576	854	420	1036	608
21	Херсон	521	376	560	624	297	798	551	225	806	869	1141	51	171	471	834	221	732	831	1065	576	-	641	351	713	691
22	Хмельницький	120	765	988	185	875	246	315	435	1177	263	240	590	548	653	193	964	662	112	455	854	641	-	463	190	455
23	Черкаси	343	324	547	321	405	709	190	126	706	578	740	300	420	279	508	696	540	575	984	420	351	463	-	660	330
24	Чернівці	312	891	1141	389	957	143	538	637	1292	336	278	642	515	892	331	981	883	176	444	1036	713	190	660	-	695
25	Чернігів	396	672	867	271	747	701	149	363	951	949	690	640	529	477	458	1112	350	568	951	608	691	455	330	695	-

Розроблюваний алгоритм буде працювати з координатами міст і розраховувати результати в градусах, тому далі наведено таблицю 2 з координатами цих міст в градусах.

Наведемо реалізацію мурашиного алгоритму оптимізації маршруту комівояжера, який повинен об'їхати 25 міст найкоротшим шляхом.

Мурашиний алгоритм був запрограмований в системі MATLAB. Для завдання з 25 обласними центрами України алгоритм без елітних мурах після 300 ітерацій знайшов оптимальний маршрут довжиною 5216,18 кілометрів за 14.16 секунд тільки в одному випадку з п'яти. Рішення можна поліпшити простим збільшенням кількості ітерацій до 1 – 2 тисячі. Графік оптимізації цільової функції за кожну ітерацію наведено на рис. 4.

Проведені експерименти свідчать, що популяція рішень ніколи не вироджується до одного, спільного для всіх мурах маршруту. Навпаки, алгоритм продовжує синтезувати нові, можливо кращі рішення.

Таблиця 2 – Координати міст дистрибуції

	Місто	Координати (x=східна довгота, y=північна широта)
1	Вінниця	x= 28.481; y= 49.2328
2	Дніпро	x= 35.0387; y= 48.4593
3	Донецьк	x= 37.8022; y= 48.023
4	Житомир	x= 28.6767; y= 50.2649
5	Запоріжжя	x= 35.1903; y= 47.8229
6	Ів-Франківськ	x= 24.7097; y= 48.9215
7	Київ	x= 30.5238; y= 50.4547
8	Кропивницький	x= 32.2597; y= 48.5132
9	Луганськ	x= 39.3171; y= 48.5671
10	Луцьк	x= 25.3424; y= 50.7593
11	Львів	x= 24.0232; y= 49.8383
12	Миколаїв	x= 31.9974; y= 46.9659
13	Одеса	x= 30.7326; y= 46.4775
14	Полтава	x= 34.5407; y= 49.5937
15	Рівне	x= 26.2274; y= 50.6231
16	Сімферополь	x= 34.1108; y= 44.9572
17	Суми	x= 34.8003; y= 50.9216
18	Тернопіль	x= 25.5892; y= 49.5534
19	Ужгород	x= 22.3; y= 48.6167
20	Харків	x= 36.2527; y= 49.9808
21	Херсон	x= 32.6178; y= 46.6558
22	Хмельницький	x= 26.9965; y= 49.4216
23	Черкаси	x= 32.0621; y= 49.4285
24	Чернівці	x= 25.9403; y= 48.2915
25	Чернігів	x= 31.2849; y= 51.5055

Отже, в будь-якому місті для мурашки існує кілька перспективних альтернатив продовження маршруту. Еволюція маршрутів комівояжера, знайдених алгоритмом, показана на рис. 5 – 7, а також повна інформація з результатами наведена в таблиці 3.

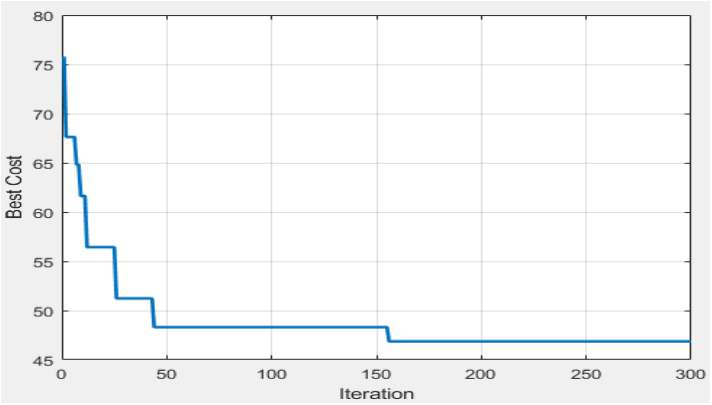

Рисунок 4 – Оптимальні поточні рішення мурашиного алгоритму

Таблиця 3 – Результати мурашиного алгоритму з різними ітераціями

Ітерації	Шлях в градусах	Шлях в км	Час роботи алгоритму
100	50.85	5660.68	6.36 с.
200	49.51	5511.37	9.18 с.
300	46.86	5216.18	14.16 с.

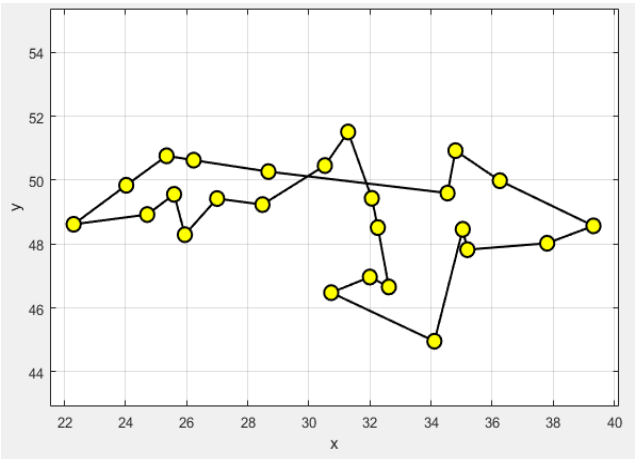

Рисунок 5 – Шлях знайдений за 100 ітерацій

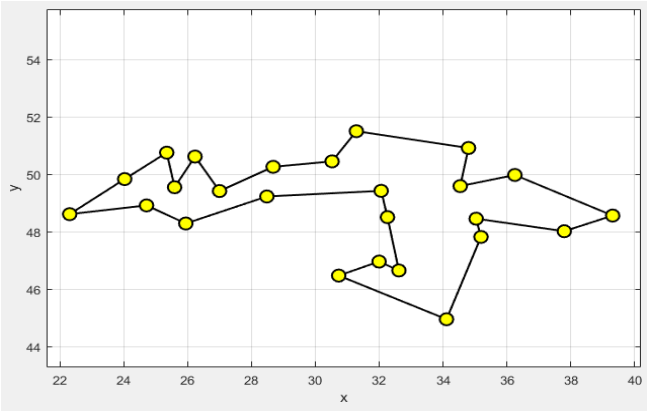

Рисунок 6 – Шлях знайдений за 200 ітерацій

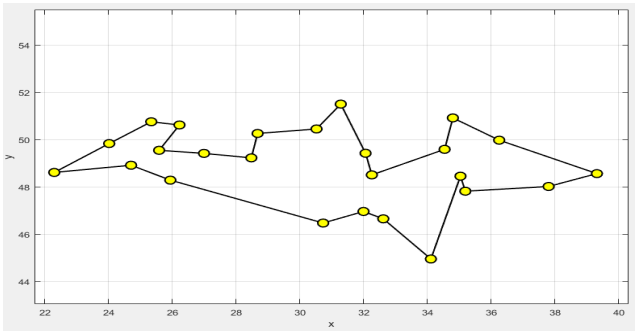

Рисунок 7 – Шлях знайдений за 300 ітерацій

Отже, згідно з маршрутом кращої ітерації, яка проілюстрована на рисунку 3.2.4 маємо маршрут, що наведено в таблиці 4.

Таблиця 4 – Найкращий із знайдених маршрутів комівояжера

№	З якого міста відправлятися	В яке місто їхати
1	Вінниця	Житомир
2	Житомир	Київ
3	Київ	Чернігів
4	Чернігів	Черкаси
5	Черкаси	Кропивницький
6	Кропивницький	Полтава
7	Полтава	Суми
8	Суми	Харків
9	Харків	Луганськ
10	Луганськ	Донецьк
11	Донецьк	Запоріжжя
12	Запоріжжя	Дніпро
13	Дніпро	Сімферополь
14	Сімферополь	Херсон
15	Херсон	Миколаїв
16	Миколаїв	Одеса
17	Одеса	Чернівці

№	З якого міста відправлятися	В яке місто їхати
18	Чернівці	Ів-Франківськ
19	Ів-Франківськ	Ужгород
20	Ужгород	Львів
21	Львів	Луцьк
22	Луцьк	Рівне
23	Рівне	Тернопіль
24	Тернопіль	Хмельницький
25	Хмельницький	Вінниця

У порівнянні з точними методами, наприклад, динамічним програмуванням або методом гілок і меж, мурашиний алгоритм знаходить близькі до оптимуму рішення за значно менший час навіть для задач невеликої розмірності (n > 20). Час оптимізації мурашиним алгоритмом є поліноміальною функцією від розмірності O (t,·n^2,·m,), тоді як для точних методів залежність експоненціальна.

Даний мурашиний алгоритм оптимізації маршруту комівояжера після відповідних модифікацій може використовуватися для вирішення різних комбінаторних задач, наприклад - квадратичної задачі про призначення, що також є NP-складною задачею.

6.7. Реалізація рішення TSP-задачі генетичним алгоритмом

Умова задачі для роботи генетичного алгоритму залишається точно такою ж як і для алгоритму мурашиної колонії. Необхідно знайти найкоротший маршрут, що починається в стартовому місті і закінчується в ньому. Маршрут повинен проходити всі міста тільки один раз. Об'їхати потрібно 25 обласних центрів України.

Реалізований генетичний алгоритм для завдання з 25 обласними центрами

України після 300 ітерацій знайшов оптимальний маршрут довжиною 5146,57 кілометрів за 2.44 секунд тільки в одному випадку з п'яти. В даному випадку рішення після 300 ітерацій не поліпшувалось, тому ставити більше немає сенсу. Графік оптимізації цільової функції за кожну ітерацію наведено на рис. 8.

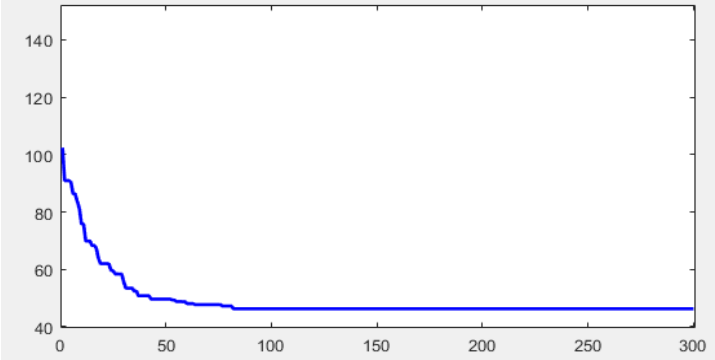

Рисунок 8 – Найкращі поточні рішення генетичного алгоритму

Проведені експерименти свідчать, що рішення знаходиться максимум за 300 ітерацій і зазвичай воно одне й те ж саме. Алгоритм не продовжує синтезувати нові, можливо кращі рішення, якщо вони взагалі існують.

Еволюція маршрутів комівояжера, знайдених алгоритмом, показана на рис. 9-11, а також повна інформація з результатами наведена в таблиці 5.

Таблиця 5 – Результати генетичного алгоритму з різними ітераціями

Ітерації	Шлях в градусах	Шлях в км	Час роботи алгоритму
100	46.9	5220.35	1.7 с.
200	46.24	5146.57	2.14 с.
300	46.24	5146.57	2.44 с.

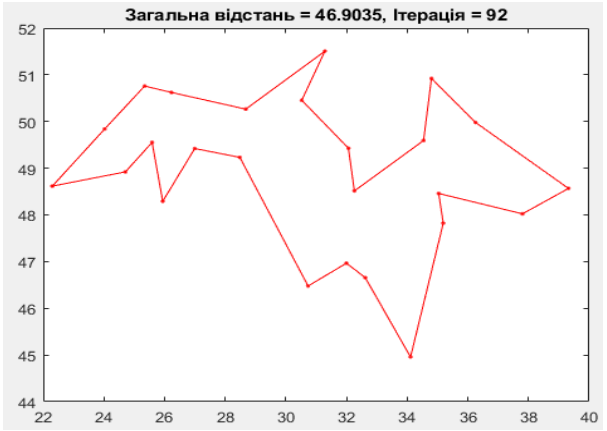

Рисунок 9 – Шлях знайдений за 100 ітерацій

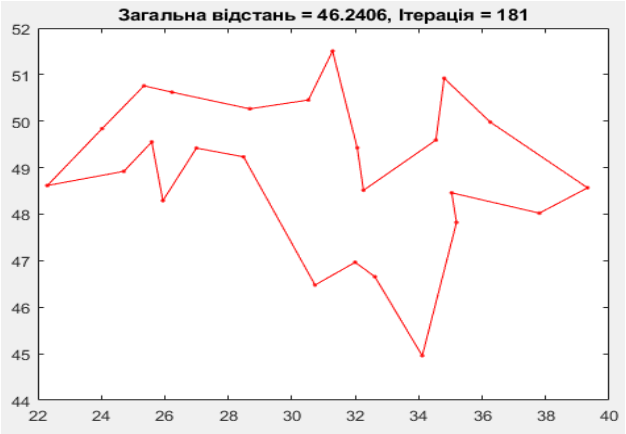

Рисунок 10 – Шлях знайдений за 200 ітерацій

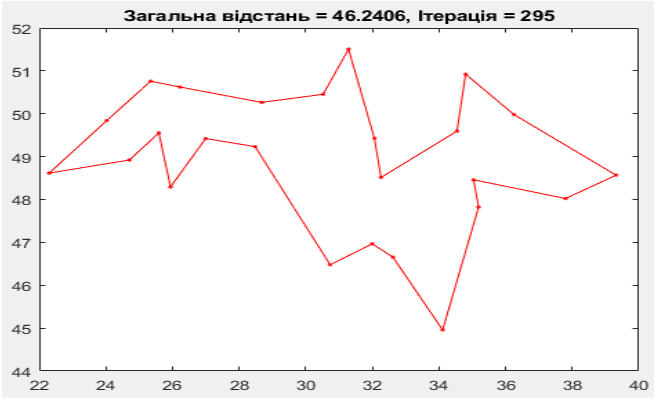

Рисунок 11 – Шлях знайдений за 300 ітерацій

Отже, згідно з маршрутом кращих результатів, які проілюстровані на рисунках 9 та 11 маємо маршрут, що наведено в таблиці 6.

Для підведення висновку про ефективність алгоритму мурашиної колонії та генетичного алгоритму було прийнято рішення провести порівняння їх роботи один з одним для вирішення однакової задачі комівояжера по об'їзду всіх обласних центрів України. Порівняння за кількістю ітерацій, довжиною найкращого маршруту та часом роботи алгоритму наведено в таблиці 7.

Результати роботи алгоритмів показують, що мурашиний алгоритм потребує більше часу на вирішення задачі. Мурашиний алгоритм може знаходить рішення для 25 міст за 300 ітерацій. Це пов'язано з тим, що результати алгоритму мурашиної колонії сходяться довше до певного значення (мінімального рішення) порівняно з генетичним алгоритмом.

Таблиця 6 – Найкращий із знайдених маршрутів комівояжера

№	З якого міста відправлятися	В яке місто їхати
1	Вінниця	Одеса
2	Одеса	Миколаїв
3	Миколаїв	Херсон
4	Херсон	Сімферополь
5	Сімферополь	Запоріжжя
6	Запоріжжя	Дніпро
7	Дніпро	Донецьк
8	Донецьк	Луганськ
9	Луганськ	Харків
10	Харків	Суми
11	Суми	Полтава
12	Полтава	Кропивницький
13	Кропивницький	Черкаси
14	Черкаси	Чернігів
15	Чернігів	Київ
16	Київ	Житомир
17	Житомир	Рівне
18	Рівне	Луцьк
19	Луцьк	Львів
20	Львів	Ужгород
21	Ужгород	Ів-Франківськ
22	Ів-Франківськ	Тернопіль
23	Тернопіль	Чернівці
24	Чернівці	Хмельницький
25	Хмельницький	Вінниця

Таблиця 7 – Порівняння роботи генетичного та мурашиного алгоритмів

Число ітерацій	Мурашиний алгоритм		Генетичний алгоритм	
	Краща довжина маршруту (км)	Час роботи алгоритму	Краща довжина маршруту (км)	Час роботи алгоритму
100	5660.68	6.36 с.	5220.35	1.7 с.
200	5511.37	9.18 с.	5146.57	2.14 с.
300	5216.18	14.16 с.	5146.57	2.44 с.

У цьому дослідженні генетичний алгоритм може забезпечити більш якісне рішення, оскільки генетичний алгоритм у кожному запуску представляє краще середнє мінімальне рішення. Якщо популяція велика, для цього знадобиться багато ітерацій. У середньому кожен запуск процесу генетичного алгоритму

представляє краще рішення, ніж мурашиний, при якому в кожному запуску графік рішення проблеми комівояжера на генетичному алгоритмі є відносно стабільним з низькими флуктуаціями рішення порівняно з алгоритмом мурашиної колонії.

Обидва алгоритми можна класифікувати як хороші та здатні представити оптимальне рішення проблеми комівояжера. Результати моделювання для шляху з 25 міст показують оптимальний маршрут подорожі, вибираючи кращий шлях без перетину маршрутів.

Висновки

В процесі виконання дослідження було реалізовано два алгоритми для вирішення задачі комівояжера як приклада NP-складної задачі. А саме в системі MATLAB запрограмовані алгоритм мурашиної колонії та генетичний алгоритм які повинні знайти оптимальний шлях для об'їзду 25 обласних центрів України з візуальним представленням цього шляху за допомогою графіка, що з'єднує точки з координатами, які відповідають астрономічним координатам кожного міста.

Був проведений аналіз проблеми рішення задачі комівояжера, виконана постановка задачі дослідження, доведена актуальність її вирішення. Проведена загальна постановка задачі комівояжера, виконана класифікація алгоритмів комбінаторної оптимізації, розглянуті сучасні напрямки штучного інтелекту - еволюційне направлення та еволюційне програмування.

Далі розглянуті концепції генетичних алгоритмів, проаналізовані генетичні алгоритми саме як евристичний напрямок. Проаналізовані типи функції пристосованості ГА та їх основні оператори, проаналізований алгоритм мурашиної колонії, виконана реалізація рішення TSP – задачі за допомогою мурашиного алгоритму.

Також розкривається питання, що таке взагалі еволюційне обчислення, та еволюційні алгоритми. А саме йдеться про те, що мурашині алгоритми засновані

на імітації самоорганізації соціальних комах за допомогою використання динамічних механізмів, з якими система досягає глобальної мети в результаті локальної низькорівневої взаємодії елементів.

Далі наведене рішення TSP – задачі генетичним алгоритмом. Виконаний порівняльний аналіз роботи двох евристичних алгоритмів. Доведена ефективність застосування евристичних алгоритмів для рішення NP- складних задач.