УДК 004.42
DOI https://doi.org/10.32689/maup.it.2023.4.1

Едуард ЖАРІКОВ
доктор технічних наук, професор, завідувач кафедри інформатики та програмної інженерії, Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського", Берестейський проспект (Перемоги), 37, Київ, Україна, індекс 03056 (zharikov.edu@i.ua)
ORCID: 0000-0003-1811-9336

Володимир ЖНАКІН
магістрант кафедри інформатики та програмної інженерії, Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського", Берестейський проспект (Перемоги), 37, Київ, Україна, індекс 03056 (vovancom98@gmail.com)
ORCID: 0009-0003-3074-4748

Eduard ZHARIKOV
Doctor of Technical Sciences, Professor, Head of the Department of Informatics and Software Engineering of National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", 37, Beresteysky (Peremohy) Ave, Kyiv, Ukraine, postal code 03056 (zharikov.edu@i.ua)

Volodymyr ZHNAKIN
Master's Student of the Department of Informatics and Software Engineering of National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", 37, Beresteysky (Peremohy) Ave, Kyiv, Ukraine, postal code 03056 (vovancom98@gmail.com)

Бібліографічний опис статті: Жаріков, Е., Жнакін, В. (2023). Удосконалений геометричний алгоритм та програмне забезпечення в задачах оптимізації транспортних маршрутів. *Інформаційні технології та суспільство*, DOI: https://doi.org/10.32689/maup.it.2023.4.1

Bibliographic description of the article: Zharikov, E., Zhnakin, V. (2023). Udoskonalenyi heometrychnyi alhorytm ta prohramne zabezpechennia v zadachakh optymizatsii transportnykh marshrutiv [Improved geometric algorithm and software in transport route optimization problems]. *Informatsiini tekhnolohii ta suspilstvo – Information technology and society*, DOI: https://doi.org/10.32689/maup.it.2023.4.1

УДОСКОНАЛЕНИЙ ГЕОМЕТРИЧНИЙ АЛГОРИТМ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ В ЗАДАЧАХ ОПТИМІЗАЦІЇ ТРАНСПОРТНИХ МАРШРУТІВ

Анотація*. Сучасні системи керування транспортними маршрутами потребують розроблення програмного забезпечення, що реалізує більш точні та швидкі алгоритми розв'язання задачі комівояжера, які забезпечують для великої кількості точок пошук найкращого маршруту з порівняно невеликою похибкою за короткий час. Основними недоліками існуючих реалізацій є стохастичність, обмежена адаптивність до параметрів задачі та велика чутливість до початкових умов, що призводить до невірних рішень та непотрібних витрат ресурсів. У статті наведена постановка задачі пошуку найменшого можливого циклічного маршруту, що проходить через заданий набір міст, починаючи і закінчуючи в тому самому місті. У результаті, алгоритм повинен знайти послідовність відвідування міст, щоб загальна довжина шляху між ними була мінімальною, і шлях проходив через кожне місто рівно один раз. Проведено експериментальні дослідження та порівняно ефективність удосконаленого геометричного алгоритму для 10, 50, 100 та 1000 міст. Результати роботи удосконаленого геометричного алгоритму для 10 міст порівняно з базовим геометричним алгоритмом та алгоритмом повного перебору, щоб переконатися, що є покращення не тільки часу пошуку маршруту, але й довжини знайденого шляху. Для 50, 100 1000 міст результати роботи удосконаленого геометричного алгоритму порівняємо з базовим геометричним алгоритмом, генетичним алгоритмом і алгоритмом оптимізації мурашиної колонії, щоб переконатися, що удосконалений геометричний алгоритм працює не гірше і швидше ніж інші алгоритми. У випадках, коли удосконалений геометричний алгоритм працює з похибкою відносно інших алгоритмів, він дозволяє отримати суттєвий виграш за часом виконання. Експерименти показали, що запропонований у роботі удосконалений геометричний алгоритм дозволяє детерміновано знаходити найкращий маршрут з похибкою 6,82% для 1000 міст. При цьому він знаходить рішення у 2,8 рази швидше, ніж алгоритм мурашиних колоній, та у 265 раз швидше, ніж генетичний алгоритм. Наведено приклад реалізації удосконаленого геометричного алгоритму мовою програмування swift для використання на платформи iOS.*

Ключові слова: *комбінаторна оптимізація, задача комівояжера, оптимальний маршрут, час виконання, точність.*

AN IMPROVED GEOMETRIC ALGORITHM AND SOFTWARE IN TRANSPORT ROUTE OPTIMIZATION PROBLEMS

Abstract. *Modern transport route management systems require the development of software that implements more accurate and faster algorithms for solving the traveling salesman problem, which provide for a large number of points the search for the best route with a relatively small error in a shorter time.*

The main disadvantages of the existing implementations are stochasticity, limited adaptability to the parameters of the problem and high sensitivity to the initial conditions, which leads to incorrect decisions and loss of resources. The article presents the formulation of the problem of finding the smallest possible circular route passing through a given set of cities, starting and ending in the same city. As a result, the algorithm must find a sequence of visiting cities so that the total length of the path between them is minimal, and the path passes through each city exactly once. Experimental studies were conducted, and the efficiency of the improved geometric algorithm was compared for 10, 50, 100 and 1000 cities. The performance of the improved geometric algorithm for 10 cities is compared to the basic geometric algorithm and the brute force algorithm to verify that there is an improvement not only in the route search time but also in the length of the path found. For 50, 100, 1000 cities, the results of the improved geometric algorithm will be compared with the basic geometric algorithm, the genetic algorithm, and the ant colony optimization algorithm to make sure that the improved geometric algorithm works no worse and faster than other algorithms. In cases where the improved geometric algorithm works with an error relative to other algorithms, it allows to obtain a significant gain in execution time. Experiments show that the improved geometric algorithm proposed in the article allows deterministically finding the best route with an error of 6.82% for 1000 cities. At the same time, it finds solutions 2.8 times faster than the ant colony optimization algorithm and 265 times faster than the genetic algorithm. The example of implementation of the improved geometric algorithm in the swift programming language for use on the iOS platform is given.

Key words: *combinatorial optimization, traveling salesman problem, optimal route, execution time, accuracy.*

Вступ

У сучасному світі, де ефективне управління маршрутами та оптимізація шляхів мають ключове значення для багатьох галузей, задача комівояжера [1] залишається однією з найважливіших. Необхідність знаходження оптимального маршруту для відвідування набору заданих точок є особливо актуальною в логістиці.

Проте, існуючі реалізації програмного забезпечення для розв'язання задачі комівояжера [2] часто стикаються з рядом проблем, які обмежують їхню ефективність та точність. Однією з основних проблем є недостатня точність наявних алгоритмів, що використовуються для розрахунку маршруту через велику кількість точок. Це може призводити до невірних рішень та непотрібних витрат ресурсів.

Запропоноване розв'язання задачі оптимізації пошуку транспортного маршруту спрямоване на усунення недоліків існуючих підходів у спосіб розробки програмного забезпечення на основі удосконаленого геометричного алгоритму. Основними недоліками існуючих реалізацій є обмежена адаптивність до різноманітних параметрів та велика чутливість до початкових умов [3]. Запропонований авторами геометричний алгоритм здатний ефективно пристосовуватися до різних умов та забезпечує високу точність у визначенні оптимального за певним критерієм маршруту.

Хоча запропоновано багато методів та алгоритмів розв'язання задачі комівояжера [4], її складність та важливість залишаються актуальними викликами для дослідників та розробників відповідного програмного забезпечення. Враховуючи ці виклики, запропоноване програмне забезпечення на основі нового геометричного алгоритму має на меті подолати недоліки та обмеження, що існують у існуючих підходах. За рахунок використання вдосконалених геометричних методів та покращених стратегій оптимізації [5], у цій роботі досягнуто значного покращення у точності та швидкості розв'язання задачі комівояжера, що відкриває нові перспективи для практичного використання запропонованого удосконаленого геометричного алгоритму в реальних умовах.

1. Постановка задачі

Задача комівояжера (англ. Traveling Salesman Problem, TSP) являє собою задачу комбінаторної оптимізації, в якій необхідно знайти найменший можливий циклічний маршрут, що проходить через заданий набір міст (точок), починаючи і закінчуючи в тому самому місті. Метою розв'язку задачі є пошук такої послідовності відвідування міст, щоб загальна довжина шляху між ними була мінімальною, і шлях проходив через кожне місто рівно один раз.

Визначення задачі комівояжера включає:

– **набір міст (точок)**: вхідними даними є множина міст, кожне з яких представлене з координатами або відстанями від інших міст. Міста можуть мати різні відстані між собою [6];

– **цільову функцію**: метою є знаходження такої послідовності обходу міст, щоб сума відстаней між цими містами була мінімальною [7];

– **обмеження**: кожне місто має бути відвідане рівно один раз, і маршрут повинен починатися і закінчуватися в тому самому місті [8];

– **симетричність**: відстань від міста A до B дорівнює відстані від B до A [9].

Не зважаючи, що задача комівояжера залишається однією з найвідоміших і досліджених у галузі комбінаторної оптимізації [10], її розв'язання для великої кількості міст все ще має велике практичне значення у різних галузях, від планування маршрутів вантажівок та обслуговування клієнтів, до оптимізації виробничих процесів. У зв'язку з цим розробка ефективних методів розв'язання задачі комівояжера залишається актуальним завданням і є об'єктом постійного дослідження та розробки покращених алгоритмів.

2. Існуючі методи розв'язання задачі комівояжера

Існує декілька традиційних алгоритмів для знаходження оптимального маршруту у задачі комівояжера. Найвідоміші з них – це метод найближчого сусіда [11], метод гілок і меж [12], алгоритм мурашиної колонії [13-15] та генетичні алгоритми [16-18]. Алгоритм, який точно знаходить найкоротший маршрут відвідування всіх міст – це алгоритм повного перебору [19], однак він потребує значного часу роботи при кількості міст більше п'ятнадцяти, що ставить під сумнів його практичне застосування у сучасному програмному забезпеченні.

Прості алгоритми знаходження оптимального рішення, такі як метод найближчого сусіда та метод гілок і меж мають масу окремих випадків, де їх застосування покаже один з найгірших варіантів для побудови найкоротшого маршруту. І лише генетичні і мурашині алгоритми дозволяють отримати порівняно хороші результати як у довжині найкращого маршруту, так і за часом пошуку цього маршруту.

Тому показники роботи запропонованого геометричного алгоритму розв'язання задачі комівояжера будемо порівнювати з:

1) алгоритмом повного перебору, щоб визначити, чи може геометричний алгоритм знаходити найкоротші шляхи обходу для кількості міст не більше 15 і робити це значно швидше;

2) генетичним і мурашиним алгоритмами, щоб визначити, чи може геометричний алгоритм знаходити оптимальні маршрути не гірше за маршрути, отримані цими алгоритмами і робити це швидше, або знаходити маршрути достатньо близькі до оптимальних, але робити це суттєво швидше.

3. Принцип роботи базового геометричного алгоритму

Геометричний алгоритм полягає у пошуку найкоротшої відстані між точками, вирішуючи завдання обчислення периметра *n*-кутника, використовуючи такі принципи:

1) необхідно уникати можливих перетинів маршруту [20];

2) необхідно рухатися від однієї точки до іншої, використовуючи найкоротшу відстань від обраної наступної точки до прямої, яка вже побудована через дві точки, що передували їй;

3) кути, які виходять шляхом додавання кожної нової точки повинні бути якнайменш гострими і якнайменш розгорнутими, тобто всіма силами прагнути до кута 180 градусів.

Таким чином, кількість можливих варіантів побудови маршруту k можна обчислити за формулою

$$k = \frac{1}{6}\left(3^{n-1} + 3\right),$$

де *n* – кількість точок або міст, які потрібно відвідати, тобто алгоритмічна та часова складність запропонованого геометричного алгоритму експоненційно зростатиме. На рисунку 1 представлена прогресія можливих варіантів маршрутів. У такому стані геометричний алгоритм хоч і буде показувати себе значно краще, ніж алгоритм повного перебору, проте він буде виконуватися дуже довго для кількості точок більше 50, і вже не зможе конкурувати з генетичним або мурашиним алгоритмами. Тому виникає необхідність оптимізувати геометричний алгоритм для роботи з великою кількістю міст.

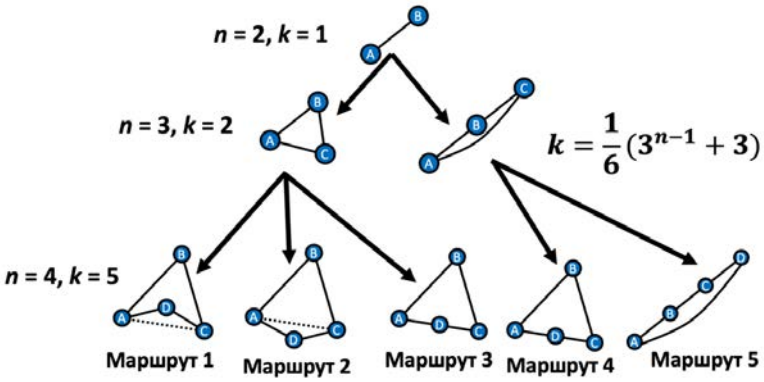

Рис. 1. Прогресія варіантів фігур в залежності від кількості міст

4. Удосконалення геометричного алгоритму

Як бачимо з рисунка 1, для $n = 4$ є п'ять варіантів можливих побудов маршрутів, при чому маршрути №3 та №4 є цілком ідентичними за визначенням. І це маршрути, в яких четверта точка збігається з однією з існуючих прямих. Тоді розпишемо загальну закономірність, як відбувається додаванням кожної наступної точки до вже побудованих фігур. Як бачимо з рисунка 1, для двох точок є один варіант обходу – і це пряма. Для трьох точок існує два варіанти обходу, залежно від їхнього взаємного розташування: якщо всі три точки лежать на одній прямій, то ця фігура – це пряма; а якщо одна з точок не лежить на прямій, де знаходяться дві інші – це трикутник. Отже, **першою умовою** для перевірки буде – лежить (або не лежить) наступна точка на існуючій прямій.

Для чотирьох точок розглянемо можливі одержані фігури детально. Якщо до цього три точки лежали на одній прямій, то четверта точка може також лежати на цій прямій, або бути поза нею. А якщо до цього три точки не лежали на одній прямій, то четверта точка може або належати одній з отриманих прямих, або лежати поза отриманим трикутником, або лежати всередині отриманого трикутника. Таким чином, точка може лежати на отриманій фігурі, бути всередині неї або лежати за межами отриманої фігури – і це буде **другою умовою** перевірки для удосконалення геометричного алгоритму.

Нехай 0 – позначення маршруту, що являє собою пряму лінію, 1 – позначення маршруту, який є трикутником. Тоді для позначення нової фігури після додавання нової точки введемо такі позначення: + (плюс) – точка лежить за межами отриманої раніше фігури; – (мінус) – точка лежить усередині отриманої фігури; для позначення фігури, де нова додана точка лежить на одній із вже побудованих прямих не додаватимемо ні + (плюс) ні – (мінус) до позначення нової фігури. Тоді нова прогресія варіантів маршрутів буде виглядати так, як показано на рисунку 2.

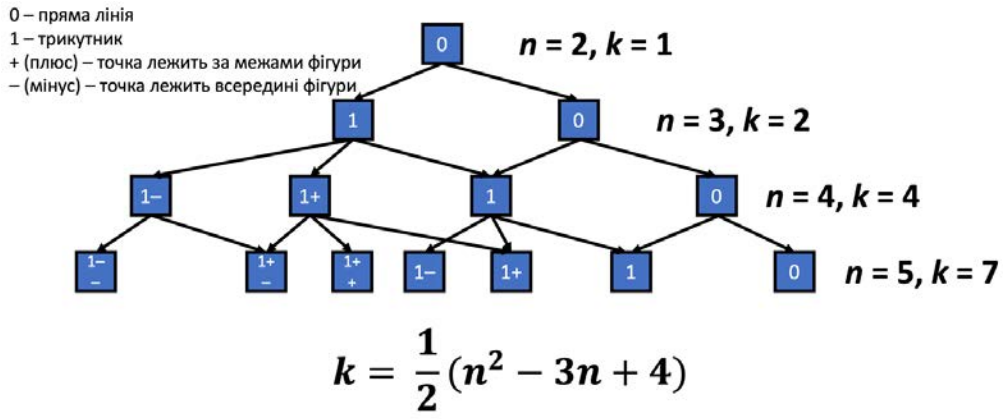

Рис. 2. Прогресія варіантів фігур у залежності від кількості міст для удосконаленого геометричного алгоритму

Отже, кількість варіантів маршрутів для удосконаленого геометричного алгоритму можна визначити за формулою

$$k = \frac{1}{2}\left(n^2 - 3n + 4\right),$$

де n – кількість точок або міст, які необхідно відвідати. Отже, алгоритмічна і часова складність удосконаленого алгоритму буде слідувати квадратичній функції, що є значним поліпшенням у порівнянні з базовою версією геометричного алгоритму, оскільки тепер алгоритм зможе виконати обчислення найкращого маршруту для великої кількості міст, наприклад, n>1000.

На рисунку 3 наведено код мовою програмування swift для реалізації удосконаленого геометричного алгоритму. Вибір мови програмування обумовлений необхідністю використання платформи iOS для реалізації програмного забезпечення пошуку найкращого маршруту серед заданих точок.

```swift
// Функція для перевірки, чи точка лежить на прямій
func pointOnLine(point: Point, line: Line) -> Bool {
    return line.A * point.x + line.B * point.y + line.C == 0
}
// Структура для представлення прямокутника
struct Rectangle {
    var left: Double
    var right: Double
    var top: Double
    var bottom: Double
}
// Функція для перевірки, чи точка знаходиться всередині, на межі або поза межами прямокутника
func pointInRectangle(point: Point, rectangle: Rectangle) -> String {
    if point.x >= rectangle.left && point.x <= rectangle.right &&
       point.y >= rectangle.top && point.y <= rectangle.bottom {
        return "Точка знаходиться всередині прямокутника"
    } else if point.x == rectangle.left || point.x == rectangle.right ||
              point.y == rectangle.top || point.y == rectangle.bottom {
        return "Точка лежить на межі прямокутника"
    } else {
        return "Точка знаходиться поза межами прямокутника"
    }
}
// Приклад використання функцій
let point = Point(x: 2.0, y: 3.0)
let line = Line(A: 1.0, B: -1.0, C: -1.0)
let rectangle = Rectangle(left: 1.0, right: 5.0, top: 2.0, bottom: 4.0)

let isPointOnLine = pointOnLine(point: point, line: line)
let pointStatusInRectangle = pointInRectangle(point: point, rectangle: rectangle)

print("Перевірка прямої: \(isPointOnLine ? "Точка лежить на прямій" : "Точка не лежить на прямій")")
print("Перевірка прямокутника: \(pointStatusInRectangle)")
```

Рис. 3. Код swift реалізації удосконаленого геометричного алгоритму

5. Оцінка ефективності удосконаленого геометричного алгоритму та експериментальні дослідження

Проводити експериментальні дослідження та порівнювати ефективність удосконаленого геометричного алгоритму будемо за такою схемою постановки експерименту.

Для n=10 порівняємо результати роботи удосконаленого геометричного алгоритму з базовим геометричним алгоритмом та алгоритмом повного перебору, щоб переконатися, що є покращення не тільки часу пошуку маршруту, але й довжини знайденого шляху.

Для n=50, n=100 та n=1000 порівняємо результати роботи удосконаленого геометричного алгоритму з базовим геометричним алгоритмом, генетичним алгоритмом і мурашиним алгоритмом, щоб переконатися, що удосконалений геометричний алгоритм працює не гірше і швидше ніж інші алгоритми, або новий геометричний алгоритм працює з невеликою похибкою відносно інших алгоритмів, але дає суттєвий виграш за часом виконання. Так як генетичний і мурашиний алгоритм є випадковими алгоритмами, то для отримання показників їх роботи будемо виконувати 15 запусків і обчислювати середнє значення часу виконання та довжини маршруту.

Похибку будемо обчислювати відносно найкращого середнього значення. Координати точок, що є вхідними даними для усіх чотирьох алгоритмів, будемо генерувати випадково. Відстань між отриманими точками маршруту будемо вимірювати в умовних одиницях, а час виконання алгоритму в секундах.

На рисунку 4 показано результати для порівняння роботи алгоритму повного перебору, базового геометричного алгоритму і удосконаленого геометричного алгоритму.

```
Кількість міст: 10
————————————————————Алгоритм повного перебору————————————————————
Отриманий шлях = 1897.1
Час виконання = 0.36288 секунд
————————————————————————————————————————————————————————————————
————————————————————Базовий геометричний алгоритм————————————————————
Отриманий шлях = 1897.1
Час виконання = 0.003281 секунд
————————————————————————————————————————————————————————————————
————————————————————Покращений геометричний алгоритм————————————————————
Отриманий шлях = 1897.1
Час виконання = 0.000037 секунд
————————————————————————————————————————————————————————————————
Найкращий шлях = 1897.1
```

Рис. 4. Результати роботи алгоритму повного перебору, базового геометричного алгоритму і удосконаленого геометричного алгоритму

На рисунку 5 показано результати роботи програмного забезпечення для порівняння удосконаленого геометричного алгоритму, генетичного алгоритму і мурашиного алгоритму для n=50. Для генетичного алгоритму у цьому та подальших експериментах взято стартову популяцію в 1000 осіб і кількість поколінь 100.

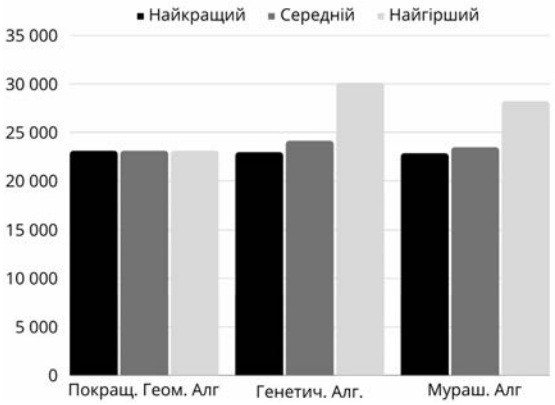

Рис. 5. Довжина отриманого маршруту (в умовних одиницях) для удосконаленого геометричного алгоритму, генетичного алгоритму і мурашиного алгоритму для n=50

На рисунку 6 показано результати роботи програмного забезпечення для порівняння удосконаленого геометричного алгоритму, генетичного алгоритму і мурашиного алгоритму для n=100.

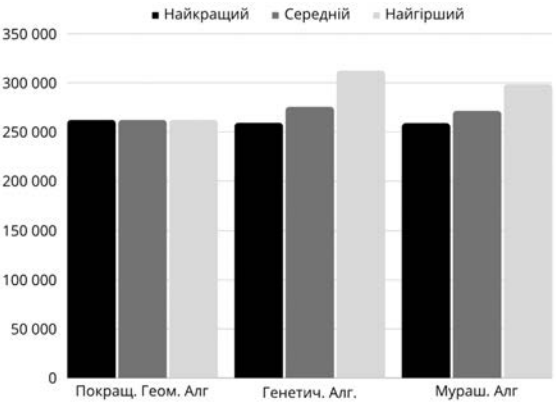

Рис. 6. Довжина отриманого маршруту (в умовних одиницях) для удосконаленого геометричного алгоритму, генетичного алгоритму і мурашиного алгоритму для n=100

На рисунку 7 показано результати роботи програмного забезпечення для порівняння удосконаленого геометричного алгоритму, генетичного алгоритму і мурашиного алгоритму для n=1000.

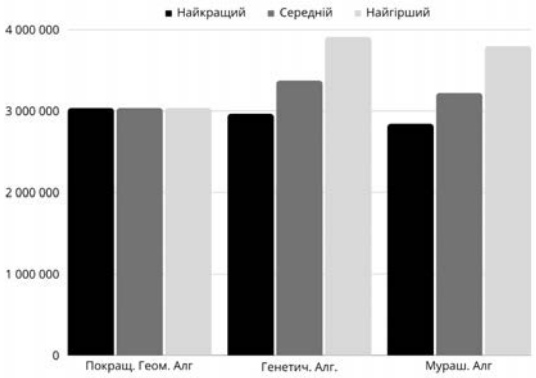

Рис. 7. Довжина отриманого маршруту (в умовних одиницях) для удосконаленого геометричного алгоритму, генетичного алгоритму і мурашиного алгоритму для n=1000

Порівняльні результати роботи досліджуваних алгоритмів показані на рисунку 8.

	Алгоритм повного перебору			Базовий геометричний алгоритм			Покращений геометричний алгоритм		
	час виконання, с.	отриманий шлях	похибка	час виконання, с.	отриманий шлях	похибка	час виконання, с.	отриманий шлях	похибка
n = 10	0,36288	1897,1	0%	0,003281	1897,1	0%	0,000037	1897,1	0%
Найкращий шлях = 1897,1									

	Генетичний алгоритм			Мурашиний алгоритм			Покращений геометричний алгоритм		
	час виконання, с.	отриманий шлях	похибка	час виконання, с.	отриманий шлях	похибка	час виконання, с.	отриманий шлях	похибка
n = 50	5,283	24137,4	5,57%	0,002501	23477,2	2,68%	0,001177	23107,5	1,06%
Найкращий шлях = 22864,8									

	Генетичний алгоритм			Мурашиний алгоритм			Покращений геометричний алгоритм		
	час виконання, с.	отриманий шлях	похибка	час виконання, с.	отриманий шлях	похибка	час виконання, с.	отриманий шлях	похибка
n = 100	11,297	275662,8	6,45%	0,012378	271538,2	4,85%	0,004852	262310,1	1,29%
Найкращий шлях = 258 965,7									

	Генетичний алгоритм			Мурашиний алгоритм			Покращений геометричний алгоритм		
	час виконання, с.	отриманий шлях	похибка	час виконання, с.	отриманий шлях	похибка	час виконання, с.	отриманий шлях	похибка
n = 1000	132,763	3375128,3	18,69%	1,408139	3222648,7	13,33%	0,498502	3037615,3	6,82%
Найкращий шлях = 2843625,4									

Рис. 8. Порівняння часу виконання, довжини отриманих маршрутів і похибок для досліджуваних алгоритмів (середні значення п'ятнадцяти запусків)

На рисунку 9 зображено графік залежності часу виконання програми у залежності від кількості міст для базового геометричного алгоритму, удосконаленого геометричного алгоритму, генетичного алгоритму і мурашиного алгоритму.

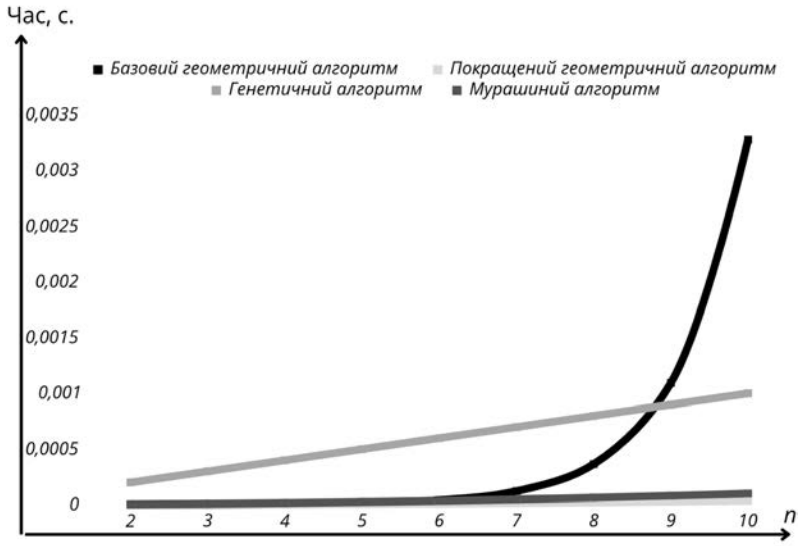

Рис. 9. Графік залежності часу виконання програми у залежності від кількості міст

Як бачимо з рисунків, наведених вище, удосконалений геометричний алгоритм працює значно швидше базового геометричного алгоритму, при чому зберігає і його точність обчислень оптимального шляху. Базовий геометричний алгоритм мав експоненційну часову складність, хоч і меншу, ніж алгоритм повного перебору. Натомість в удосконаленому геометричному алгоритмі часова складність має квадратичну складність, що дозволяє обчислювати маршрути для великої кількості міст. Також, як бачимо з рисунка 9, удосконалений геометричний алгоритм має витрати часу на обчислення нижче ніж одні з кращих варіантів генетичних і мурашиних алгоритмів за досить низької похибки обчислень.

Висновки

У статті запропонований удосконалений геометричний алгоритм розв'язання задачі комівояжера та програмне забезпечення його реалізації на платформі iOS. Удосконалений геометричний алгоритм дозволяє отримати кращі показники ефективності роботи порівняно з класичними методами розв'язання

задачі комівояжера. Отримані результати дозволяють стверджувати, що якісні показники роботи удосконаленого геометричного алгоритму мають тенденцію до поліпшення зі збільшенням кількості міст у порівнянні з мурашиним та генетичним алгоритмами. Він дозволяє детерміновано знаходити найкращий маршрут з порівняно невеликою похибкою (6,82% для 1000 міст) за коротший час, ніж алгоритм мурашиних колоній (у 2,8 рази швидший) та генетичний алгоритм (у 265 раз швидший), що робить його швидким і невибагливим до ресурсів інструментом для пошуку оптимальних маршрутних планів.

Програмне забезпечення, розроблене з урахуванням даного алгоритму, може застосовуватися у різних галузях, де оптимізація маршрутів має критичне значення. Запропонований алгоритм не тільки покращить продуктивність та ефективність логістичних процесів, а й відкриє нові можливості для практичного застосування задачі комівояжера.

Список використаних джерел:

1. Lawler, E. L., et al. The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley. 1985. P. 25-31.
2. Monnot, J., & Toulouse, S. The traveling salesman problem and its variations. Paradigms of combinatorial optimization: problems and new approaches. 2014. P. 173-214.
3. Chandra, A., & Naro, A. A comparative study of metaheuristics methods for solving traveling salesman problem. International Journal of Information Science and Technology. 2022. №6(2), P.1-7.
4. Zhang, C., & Sun, P. Heuristic Methods for Solving the Traveling Salesman Problem (TSP): A Comparative Study. In 2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC). 2023. P. 1-6. IEEE.
5. Reinelt, G. The Traveling Salesman: Computational Solutions for TSP Applications. Springer. 1994. P. 18-21.
6. Toth, P., et al. The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications. SIAM. 2002. P. 39-41.
7. Nagata, Y. Recent developments in combinatorial optimization for the traveling salesman problem: A survey. European Journal of Operational Research. 2011. №213(1). P. 1-10.
8. Vansteenwegen, P., et al. The Traveling Salesman Problem: A Case Study in Local Optimization. Operations Research Perspectives. 2011. P. 65-77.
9. Gutin, G., Punnen, A. The traveling salesman problem and its variations. Springer Science & Business Media, 2006.
10. Jiang, C., Wan, Z., & Peng, Z. A new efficient hybrid algorithm for large scale multiple traveling salesman problems. Expert Systems with Applications. 2020. №139. P. 112867.
11. Cheikhrouhou, O., & Khoufi, I. A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy. Computer Science Review. 2021. №40. P. 100369.
12. Lawler, E. L., et al. The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley. 1985. P. 49-52.
13. Dorigo, M., et al. The Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics. 1996. №26(1). P. 29-41.
14. Dorigo, M., et al. Ant Colony Optimization. MIT Press. 2004. P. 16-20.
15. Blum, C., et al. The hyper-cube framework for ant colony optimization. IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics. 2004. №34(2). P. 1161-1172.
16. Holland, J. H. Adaptation in Natural and Artificial Systems. University of Michigan Press. 1975. P. 17-22.
17. Goldberg, D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional. 1989. P. 24-27.
18. Mitchell, M. An Introduction to Genetic Algorithms. MIT Press. 1998. P. 30-33.
19. Cormen, T. H., et al. Introduction to Algorithms. MIT Press. 2022.
20. An Improvement to the 2-Opt Heuristic Algorithm for Approximation of Optimal TSP Tour. URL: https://www.mdpi.com/2076-3417/13/12/7339

References:

1. Lawler, E. L., et al. (1985). The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley. P. 25-31.
2. Monnot, J., & Toulouse, S. (2014). The traveling salesman problem and its variations. Paradigms of combinatorial optimization: problems and new approaches. P. 173-214.
3. Chandra, A., & Naro, A. (2022). A comparative study of metaheuristics methods for solving traveling salesman problem. International Journal of Information Science and Technology. №6(2), P.1-7.
4. Zhang, C., & Sun, P. (2023). Heuristic Methods for Solving the Traveling Salesman Problem (TSP): A Comparative Study. In 2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC). P. 1-6. IEEE.
5. Reinelt, G. (1994). The Traveling Salesman: Computational Solutions for TSP Applications. Springer. P. 18-21.
6. Toth, P., et al. (2002). The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications. SIAM. P. 39-41.
7. Nagata, Y. (2011). Recent developments in combinatorial optimization for the traveling salesman problem: A survey. European Journal of Operational Research. №213(1). P. 1-10.

8. Vansteenwegen, P., et al. (2011). The Traveling Salesman Problem: A Case Study in Local Optimization. Operations Research Perspectives. P. 65-77.
9. Gutin, G., Punnen, A. (2006). The traveling salesman problem and its variations. Springer Science & Business Media.
10. Jiang, C., Wan, Z., & Peng, Z. (2020). A new efficient hybrid algorithm for large scale multiple traveling salesman problems. Expert Systems with Applications. №139. P. 112867.
11. Cheikhrouhou, O., & Khoufi, I. (2021). A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy. Computer Science Review. №40. P. 100369.
12. Lawler, E. L., et al. (1985). The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley. P. 49-52.
13. Dorigo, M., et al. (1996). The Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics. №26(1). P. 29-41.
14. Dorigo, M., et al. (2004). Ant Colony Optimization. MIT Press. P. 16-20.
15. Blum, C., et al. (2004). The hyper-cube framework for ant colony optimization. IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics. №34(2). P. 1161-1172.
16. Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press. P. 17-22.
17. Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional. P. 24-27.
18. Mitchell, M. (1998). An Introduction to Genetic Algorithms. MIT Press. P. 30-33.
19. Cormen, T. H., et al. (2022). Introduction to Algorithms. MIT Press.
20. An Improvement to the 2-Opt Heuristic Algorithm for Approximation of Optimal TSP Tour. Retrieved from https://www.mdpi.com/2076-3417/13/12/7339