УДК 519.857
DOI https://doi.org/10.32782/2663-5941/2025.5.2/36

Семаньків М.В.
Карпатський національний університет імені Василя Стефаника

ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ МЕТОДІВ РОЗВ'ЯЗАННЯ TSP ЗАДАЧІ

У статті розглянуто класична NP-складна оптимізаційна задача комівояжера, що передбачає пошук оптимального маршруту обходу множини вершин-міст з мінімальною сумарною довжиною (вартістю). В процесі дослідження проаналізовано методи та алгоритми розв'язання задачі комівояжера, зокрема точні та евристичні підходи, їх ефективність, обчислювальна складність та якість отриманих наближених рішень. Досліджено сучасні наукові публікації, присвячені підвищенню точності та швидкодії алгоритмів для TSP, оптимізації їх програмної реалізації, зменшенню часу виконання, а також можливості обробки великої кількості вхідних даних. Відзначено, що для невеликих задач ефективно працюють точні методи, проте зі зростанням кількості міст їх застосування стає обмеженим через експоненційний ріст обчислювальної складності. Для великих задач доцільним є використання наближених або метаевристичних алгоритмів, які хоча і не гарантують оптимального розв'язку, проте здатні забезпечити високу якість результатів у прийнятні проміжки часу. Дослідження проводилось на основі реальних даних про міста Івано-Франківської області, що дозволило оцінити практичну ефективність різних методів. Для реалізації алгоритмів та проведення обчислювальних експериментів обрано мову програмування Python, яка завдяки широкому спектру бібліотек (NumPy, SciPy, GeoPandas) забезпечує гнучкість та зручність у реалізації. Охарактеризовано переваги та недоліки методів для знаходження рішень та визначено шляхи їх подальшого вдосконалення. Запропоновано оптимізацію методу повного перебору шляхом фіксації початкового міста, що дозволяє відкинути зайві циклічні зсуви послідовностей та уникнути розгляду дзеркальних маршрутів, які відрізняються лише напрямом обходу. Практична реалізація методу гілок та меж у Python показала, що через інтерпретовану природу мови недоцільним є складне відсікання гілок, адже це значно ускладнює програмну реалізацію й може не виправдати витрат часу. Окремо запропоновано удосконалення методу найближчого сусіда: вибір найвіддаленішого міста як початкової точки дозволяє скоротити час виконання алгоритму у порівнянні зі звичайним випадковим вибором старту та у середньому покращує якість маршруту на 8–10%, що підтверджує доцільність даного підходу для практичного застосування у реальних транспортних задачах. Таким чином, робота систематизує та аналізує існуючі методи розв'язання задачі комівояжера, пропонує власні шляхи удосконалення алгоритмів, а також підкреслює перспективи застосування комбінованих підходів, що поєднують точні та метаевристичні методи для досягнення кращого балансу між якістю рішень та витратами часу на їх обчислення.

Ключові слова: *задача комівояжера, Python, обчислювальна складність.*

Постановка проблеми. Задача комівояжера (TSP) полягає у знаходженні найкращого маршруту, який дозволяє відвідати всі задані міста та повернутися до початкової точки з мінімальною вартістю подорожі або пройшовши мінімальну відстань. Більшість прийнятних по часу та обчислювальній складності методів для розв'язання даної задачі є евристичними, які дають наближений до нього результат деколи із доволі значною похибкою. Окрім них, існують і точні методи, проте їх виконання займає значний проміжок часу для достатньо великої вибірки [1].

Аналіз останніх досліджень і публікацій. В [2] розглянуто узагальнення класичного TSP із застосуванням тих же базових алгоритмічних підходів, проте із розширеними моделями та новими умовами для забезпечення допустимості маршруту. Авторами представлено нову варіацію класичної задачі комівояжера, яка поєднує концепцію "околів", де дозволяється відвідувати будь-яку точку всередині регіону, та "перешкод", які маршрут не може перетинати. Але запропоновані моделі та алгоритми протестовано на задачах відносно невеликої розмірності, тому ефективність на дуже великих реальних прикладах залишається відкритим питанням.

У статті [3] запропоновано нові механізми скорочення пошукового простору, зокрема функції

меж та правила фільтрації ребер на основі пізнього часу відправлення. Хоч метод здатний швидше, ніж інші сучасні підходи, знаходити еталонні рішення, але для дуже великих задач масштабованість залишається викликом та не завжди гарантує швидке доведення оптимальності у випадку найскладніших сценаріїв.

В роботі [4] автори запропонували метод IH-OPT для генерації складних TSP екземплярів із великим інтегральним розривом, реалізували гілок та меж для його розв'язання. Проте метод обмежений розмірами та має високу обчислювальну складність і не завжди знаходить рішення в межах часу.

Метою статті є дослідження методів розв'язання задач комівояжера та вдосконалення алгоритмів їх реалізації.

Запропоновано наступні рішення та досліджено їх ефективність:

– оптимізація методу повного перебору за рахунок відсікання дзеркальних маршрутів;

– спрощення алгоритму гілок та меж внаслідок врахування особливостей його реалізації в мові програмування Python;

– удосконалення методу найближчого сусіда, що базується на виборі найвіддаленішого міста як початкового, для пошуку оптимального шляху між населеними пунктами.

Виклад основного матеріалу. Методи розв'язання задачі комівояжера TSP відрізняються підходом до пошуку оптимального маршруту та співвідношенням точність-швидкість. Методи повного перебору гарантують знаходження оптимального маршруту, але мають експоненційну складність і швидко стають непридатними для великих наборів міст. Евристичні та метаевристичні методи не гарантують глобального оптимуму, але дають близькі до оптимальних рішення за набагато менший час. Динамічне програмування (алгоритм Хелда-Карпа) зменшує складність порівняно з методом перебору, але все ще залишається експоненційним методом. Таким чином, вибір методу залежить від розміру задачі та вимог до точності: для малих n обирають точні методи, а для великих – евристики [1, 5].

Методом повного перебору можна розглядати задачу до 15 заданих міст. Для задач до 20 міст можна використати динамічне програмуванням (метод Беллмана–Гельда–Карпа має складність $O(n^2 \cdot 2^n)$). Використання методу гілок та меж та комбінованих оптимізацій дозволяє обчислити розв'язок для 100–200 міст. Сучасні спеціалізовані солвери (наприклад, Concorde TSP Solver) можуть знаходити оптимальні маршрути для задач з десятками тисяч міст.

Проведено дослідження реалізації розв'язання задачі комівояжера для міст Івано-Франківської області, що задані своїми координатами. Нижче буде продемонстровано результат аналізу вибірки із 8 міст (рис. 2), де показано маршрути, що визна-

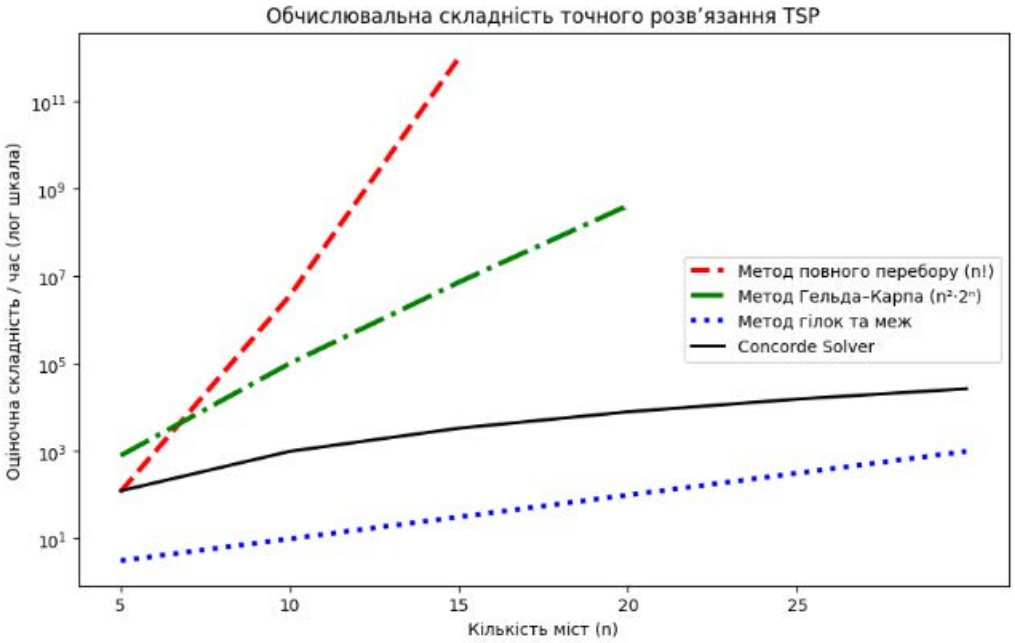

Рис. 1. Залежність часу виконання від кількості заданих міст для точних методів розв'язання задачі комівояжера

чені двома методами розв'язання (точним – методом повного перебору, наближеним – методом найближчого сусіда) у вигляді ліній сполучення між містами, побудованими на координатній площині.

Оптимізація методу повного перебору за рахунок відсікання дзеркальних маршрутів. Основа методу повного перебору полягає перегляді усіх можливих перестановок міст та отримання дистанції комівояжера для кожного із наборів. А згодом, вибір найменшої як оптимального маршруту. Перевагами алгоритму є точність (отриманий результат буде істинно оптимальним) та простота реалізації. Недоліки алгоритму: час виконання (для аналізу вибірки із одинадцяти міст знадобилося 51 хв.); висока потреба у обчислювальних ресурсах; висока складність (даний алгоритм є експоненціальним, що означає степеневий ріст із збільшенням кількості вхідних даних) [6].

Для методу повного перебору запропоновано його модифікувати наступним чином: фіксуємо перше місто, щоб уникнути циклічних зсувів (варіанти 1-2-3-1 і 2-3-1-2 мають одну й ту ж сумарну відстань), відкидаємо дзеркальні маршрути (адже маршрут 1-2-3-1 матиме ту ж довжину, що і 1-3-2-1, адже це той самий маршрут у зворотньому порядку). Тобто, задамо впорядкованість перестановок міст, що відсікатиме 50% дзеркальних варіантів.

Складність методу повного перебору оцінюють в $n!$. Але оскільки маршрут починається і закінчується в одному місті можна початкове місто зафіксувати (щоб не рахувати однакові маршрути з різних стартів). Це зменшить складність до $(n-1)!$. Якщо розглянути маршрути 1-2-3-1 та 1-3-2-1, то для задачі класичного TSP вважається, що ці два маршрути еквівалентні, оскільки сумарна відстань буде однаковою. Звідси складність оптимізації становить $(n-1)!/2$. Згідно запропонованої модифікації алгоритму до коду програми додано кілька додаткових рядків, але продуктивність зросте майже вдвічі для кожного набору міст.

Спрощення алгоритму гілок та меж. Ще одним відомим методом розв'язання є метод гілок та меж, де рішення про включення або виключення вершини зі шляху приймається на основі обчислення нижньої границі. Завдяки цьому кількість перевірених шляхів значно менша, особливо якщо застосовуються ефективні оцінки нижніх меж. У середньому працює набагато швидше, ніж $n!$ (метод перебору), і може знайти оптимальний маршрут для 20-30 міст за допустимий час [1, 6].

Запропоновано алгоритм, де кроку із виключенням вершини немає. Це значно спростило проблему просування по відкинутій гілці й пришвидшило виконання алгоритму, реалізованого на мові програмування Python. Модифікований метод без процесу відкидання гілок нагадує метод простого перебору, де є мінімум логіки та гілок. В свою чергу він при програмній реалізації використовує менше перевірок, копій, розгалужень, тобто виконує менше операцій, що зменшує час виконання. У задачі комівояжера метод гілок і меж зазвичай вважається ефективнішим за метод прямого пере-

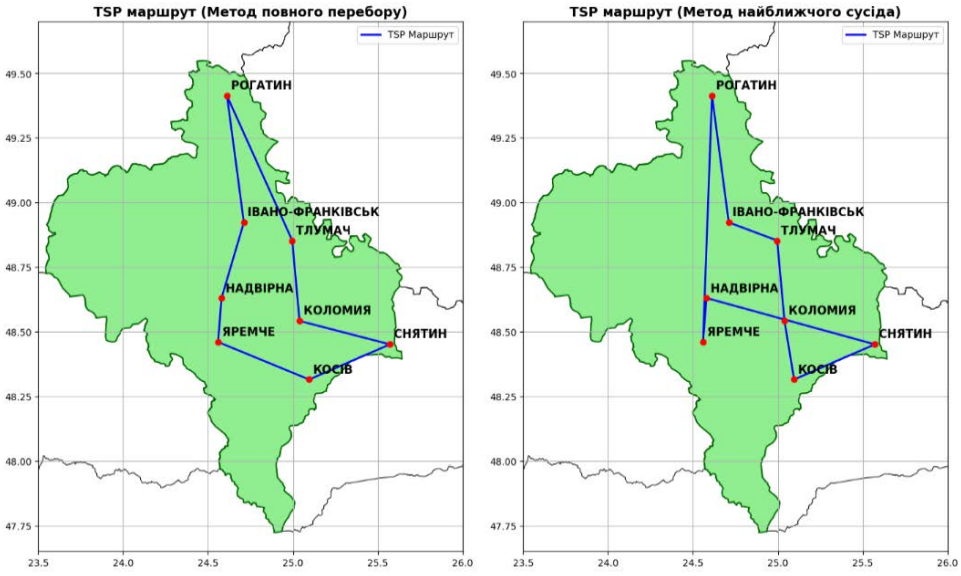

Рис. 2. Візуалізація обчислених маршрутів для восьми міст Івано-Франківської області точним та наближеним методами

бору, бо відсікає частину пошуку (не розглядає явно гірші варіанти) та має меншу кількість обходів. Але на практиці "відкидання" може сповільнювати роботу через накладні витрати на обчислення меж. Обчислення нижньої межі для кожного вузла дерева може бути ресурсомістким, особливо якщо використовуються матричні зменшення, жадібні оцінки, евристики або копії матриць. Наприклад, у Python це може коштувати більше часу, ніж просто продовжити обхід. Тому задача з перевіркою меж може в сумі працювати довше, ніж просто пройти всі варіанти без перевірки меж. Ще однією особливістю, що ускладнює та сповільнює роботу є витрати на рекурсію і створення копій, адже метод гілок та меж часто використовує глибоку рекурсію та копії даних (наприклад, скорочення матриць відстаней). Ще теж ускладнює реалізацію на мові Python, адже дана мова програмування не оптимізована для глибокої рекурсії та копіювання великих структур.

Таблиця 1

Порівняння швидкодій базового методу гілок та меж і модифікованого

Кількість міст	Метод гілок та меж (с)	Метод гілок та меж без процесу відкидання гілок (с)
6	1.99	1.33
7	1.46	1.36
8	2.81	1.63
9	9.18	2.16
10	79.76	13.63
11	917.86	219.52

На основі наведених даних в табл. 1. можна зробити висновок, що при в мові програмування Python доцільніше не використовувати відсікання гілок у методі гілок та меж, це дозволить підвищити ефективність розв'язання задачі комівояжера, особливо при зростанні кількості міст. Для малих n (6–8) різниця в часі обчислень незначна, але вже починаючи з 9 міст ефект суттєвий: час виконання з відсікання зростає набагато швидше. При 11 містах метод із відкиданням працює більш ніж у 4 рази довше. Отже, у Python відсікання гілок у методі гілок та меж уповільнює виконання, оскільки перевірка умов відсікання потребує додаткових обчислень (обчислення нижніх меж, виклики функцій, рекурсії), які в інтерпретованій мові створюють значні накладні витрати.

Удосконалення методу найближчого сусіда. На сьогоднішній день більш популярні наближені методи розв'язання задачі комівояжера, ніж точні, через її обчислювальну складність. Точні алгоритми (повний перебір, гілки та межі, динамічне програмування) гарантують знаходження оптимального маршруту, але їхня складність зростає факторіально і дуже швидко стає неприйнятною навіть для середньої кількості міст (наприклад, понад 20–25). Наближені та евристичні методи (найближчого сусіда, генетичні алгоритми, мурашині алгоритми, методи локального пошуку) працюють значно швидше, дають рішення, яке близьке до оптимального, і дозволяють ефективно розв'язувати задачі для сотень і тисяч міст, що робить їх практичними для реальних логістичних і виробничих систем [6, 7].

У методі найближчого сусіда вибір початкового міста має велике значення, адже метод є жадібним і будує маршрут, на кожному кроці вибираючи найближче ще не відвідане місто. Оскільки вибір на першому кроці впливає на всі наступні, то початкова точка може призвести до різних маршрутів і, відповідно, до різної сумарної довжини шляху. На практиці часто застосовують одну з двох стратегій: запустити алгоритм із кожного міста як початкового та вибирати найкоротший маршрут серед отриманих або вибрати "найкраще" стартове місто за якимось критерієм (наприклад, місто з мінімальною сумою відстаней до інших) [7].

Запропоновано обрати початковою точкою місто, яке є найвіддаленіше від усіх інших міст (найвіддаленіше місто від центра мас, тобто середнього значення координат, стає стартовим). Якщо в методі найближчого сусіда обрати найдальше від усіх місто як початкове, то це покращує результат у багатьох випадках, адже метод найближчого сусіда має схильність «замикатися» у локальні групи міст, а останні кілька кроків часто створюють довгі «стрибки», які збільшують сумарну відстань. Якщо почати з найбільш віддаленого міста, алгоритм спершу забере з розгляду крайні точки, і подальший маршрут охоплюватиме більш компактну групу міст, що часто дає коротший цикл. Але слід зауважити, що це не гарантує оптимального рішення, бо метод все ще жадібний – він не враховує глобальну картину, лише локально мінімізує відстань на кожному кроці, але зменшує час виконання.

При вибору найбільш віддаленого від усіх інших міста алгоритм побудує коротший маршрут, ніж при випадковому старті. Це відбувається тому, що віддалене місто буде приєднане на ранніх кроках, коли вибір наступних міст найбільш гнучкий, і алгоритм не буде змушений повернутись до нього в кінці маршруту через довгий

Таблиця 2

Час виконання програм

Кількість міст	Метод найближчого сусіда (с)	Найвіддаленіше місто як початкове в методі найближчого сусіда (с)
6	1.30	1.38
7	1.75	1.22
8	2.28	1.18
9	1.42	1.29
10	1.34	1.17
11	1.42	1.19
12	1.73	1.19

Таблиця 3

Відстань маршруту для міст Івано-Франківської області

Кількість міст	Відповідь точних методів (сумарна відстань маршруту, км)	Метод найближчого сусіда (сумарна відстань маршруту, км)	Найвіддаленіше місто як початкове (сумарна відстань маршруту, км)
6	266.79	325.77	302.35
7	287.80	329.60	305.07
8	325.90	360.19	326.08
9	329.44	398.29	362.62
10	360.03	418.48	384.57
11	373.64	452.33	415.16
12	393.95	465.95	458.69

і дорогий гак. Такий вибір початкової точки частково компенсує головний недолік евристики найближчого сусіда – її схильність залишати далекі міста на кінець, що збільшує загальну довжину маршруту. У результаті маршрут стає більш збалансованим і може виявитися ближчим до оптимального.

Згідно даних табл. 2, вибір найвіддаленішої вершини як початкової дозволяє скоротити час виконання методу найближчого сусіда приблизно на 5–25% залежно від кількості міст. Найбільший виграш спостерігається при 8 містах (економія часу ≈ 48%), тоді як при 6 або 9–12 містах скорочення є помірним (≈ 5–15%). Це показує, що такий підхід особливо ефективний для середніх розмірів задачі, забезпечуючи суттєве прискорення алгоритму без ускладнення його структури.

Метод найближчого сусіда дає розв'язки довші (більші відстані маршруту) від точного маршруту на 15–22%, тоді як вибір найвіддаленішої вершини як початкової дозволяє зменшити відхилення до 0–13%, а в окремих випадках (8 міст) результат майже співпадає з точним розв'язком (+0.1%). Таким чином, використання найвіддаленішої вершини як стартової у середньому покращує результат на 8–10% у порівнянні з класичним методом найближчого сусіда.

Висновки. Для методу повного перебору важливим є кількість перевірок, що буде здійснено за час виконання, адже кількість операцій факторіально зростає при збільшенні кількості міст. Тому для даного методу фіксування стартового міста та відкидання варіантів розв'язків, що будуть отримані в результаті циклічних зсувів та дзеркального відображення, суттєво впливає на час виконання та знаходження розв'язку задачі комівояжера

Практична реалізація методу гілок та меж в Python показала, що відсікання гілок ускладнює виконання алгоритму через накладні витрати на обчислення меж та їх перевірок, тому в даній програмування рекомендовано відмовитись ввід відсіканні в методу гілок та меж.

Запропонований вибір найвіддаленішої вершини як початкової у методі найближчого сусіда дозволяє до 25% зменшити час виконання алгоритму та до 13% зменшити відхилення з оптимальним маршрутом порівняно зі звичайним випадковим вибором стартової точки. Це пов'язано з тим, що початок маршруту з вершини, віддаленої від інших, дозволяє алгоритму швидше сформувати більш збалансований маршрут, уникаючи надмірних повернень та зайвих кроків. Такий підхід покращує ефективність обчислень, особливо при збільшенні кількості міст.

Список літератури:

1. Matai R., Singh S. P., Mittal M. L. Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches / R. Matai, S. P. Singh, M. L. Mittal. In: Traveling Salesman Problem, Theory and Applications. InTech Open Access Publisher, 2010. DOI: 10.5772/12909.

2. Puerto J., Valverde-Martín C. The hampered travelling salesman problem with neighbourhoods. *Computers & Industrial Engineering*. 2024. Vol. 189. P. 109-120.

3. Fontaine R., Dibangoye J., Solnon C. Exact and anytime approach for solving the time dependent traveling salesman problem with time windows. European Journal of Operational Research. 2023. Vol. 311, № 3. P. 833–844. DOI: https://doi.org/10.1016/j.ejor.2023.06.001

4. Vercesi E., Gualandi S., Mastrolilli M., Gambardella L. M. On the generation of metric TSP instances with a large integrality gap by branch-and-cut. Mathematical Programming Computation. 2023. Vol. 15. P. 475–503. DOI: 10.1007/s12532-023-00235-7.

5. Gutin A., Yeo A., Zverovich A. Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP. *Discrete Applied Mathematics*. 2002. Vol. 117. P. 81–86.

6. Cook W. In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation. Princeton : Princeton University Press, 2011. 228 p. ISBN 978-0-691-15282-0.

7. Sengupta S., Das S. Selective Nearest Neighbors Clustering / Pattern Recognition Letters. Vol. 155, March 2022. P. 178–185. DOI: 10.1016/j.patrec.2021.10.005.

Semankiv M.V. IMPROVING THE EFFICIENCY OF ALGORITHMS FOR THE TRAVELING SALESMAN PROBLEM

The article considers the classical NP optimization problem of the Traveling Salesman Problem (TSP), which involves finding the optimal route that visits a set of cities with the minimum total distance (cost). In the course of the study, methods and algorithms for solving the TSP were analyzed, including exact and heuristic approaches, their efficiency, computational complexity, and the quality of the obtained approximate solutions. Recent scientific publications devoted to improving the accuracy and performance of TSP algorithms were examined, focusing on software implementation optimization, reduction of execution time, and the ability to handle large input datasets. It was noted that for small-scale problems, exact methods are effective; however, with the increasing number of cities, their application becomes limited due to the exponential growth of computational complexity. For larger problems, approximate or metaheuristic algorithms are more appropriate, as they do not guarantee an optimal solution but are capable of delivering high-quality results within acceptable timeframes. The research was conducted using real data from the cities of the Ivano-Frankivsk region, which allowed for evaluating the practical efficiency of different methods. Python was chosen as the programming language for implementing the algorithms and conducting computational experiments, as its wide range of libraries (NumPy, SciPy, GeoPandas) provides flexibility and convenience in implementation. The advantages and disadvantages of various methods for finding solutions and for further optimization were characterized. An optimization of the brute-force method was proposed by fixing the starting city, which allows the elimination of redundant cyclic shifts of sequences and avoids considering mirror routes that differ only in traversal direction. The practical implementation of the branch and bound method in Python showed that, due to the interpreted nature of the language, complex branch pruning is not advisable, as it significantly complicates the implementation and may not justify the time costs. Additionally, an improvement to the nearest neighbor method was suggested: selecting the most distant city as the starting point reduces algorithm execution time compared to the usual random choice of a start and, on average, improves route quality by 8–10%. This confirms the practical feasibility of this approach in real-world transportation tasks. Thus, the work systematizes and analyzes existing methods of solving the Traveling Salesman Problem, proposes original algorithmic improvements, and emphasizes the prospects of applying combined approaches that integrate exact and metaheuristic methods to achieve a better balance between solution quality and computation time.

Key words: *Traveling Salesman Problem (TSP), Python, computational complexity.*

Дата надходження статті: 18.09.2025
Дата прийняття статті: 01.10.2025
Опубліковано: 16.12.2025