Вітаю, шановні члени комісії, до вашої уваги пропонується магістерська дисертація на тему СПОСІБ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ОПТИМІЗАЦІЇ МАРШРУТІВ КОМІВОЯЖЕРА**,** яку виконав студент групи КП-31мп Маховой Олександр Вікторович під керівництвом Юрчишина Василя Яковича.

**АКТУАЛЬНІСТЬ ДОСЛІДЖЕННЯ**

Оптимізація маршрутів є однією з ключових задач у сучасній логістиці, транспортуванні та управлінні ресурсами. Задача комівояжера (TSP) є класичним прикладом цієї проблеми та часто використовується для моделювання подібних задач у реальному світі. Її актуальність підтверджується широким спектром застосувань, серед яких логістика та транспортування, маршрутизація дронів та робототехніка, управління ресурсами та багато іншого.

Задача комівояжера полягає у знаходженні найкоротшого можливого маршруту, що проходить через заданий набір міст, відвідуючи кожне місто рівно один раз і повертаючись до початкового міста. Задача є NP-складною. Це означає, що на сьогодні не існує відомого алгоритму, який би знаходив оптимальне рішення для будь-якої кількості точок за поліноміальний час

Існуючі методи вирішення задачі зазвичай зосереджуються або на якості отриманого рішення, або на швидкості виконання, що ускладнює їх використання для великих наборів даних. Запропонований підхід спрямований на досягнення ефективного компромісу між якістю рішення та часом його виконання, що дозволить зберігати практичну цінність у реальних умовах.

**ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ**

Об ’єктом дослідження є процеси оптимізації маршрутів у задачі комівояжера шляхом застосування математичних методів і алгоритмів.

Предметом дослідження є методи, алгоритми та програмні засоби для вирішення задачі комівояжера та оптимізації результатів.

**ПОСТАНОВКА ЗАДАЧІ**

Метою дослідження є підвищення ефективності розв’язання задачі комівояжера шляхом розробки нового методу, який поєднує геометричні підходи для побудови початкового маршруту та локальну оптимізацію, забезпечуючи баланс між якістю рішення та часом його виконання, а також впровадження цього методу у програмне забезпечення з подальшою оцінкою його продуктивності у порівнянні з існуючими підходами. Для цього нам потрібно по перше проаналізувати існуючі методи та засоби розв’язання задачі комівояжера. По друге - сформулювати гіпотезу щодо покращення методу для підвищення ефективності розв’язання. По третє - розробити модифікований метод розв’язання задачі комівояжера. Далі обґрунтувати вибір засобів для програмної реалізації та реалізувати програмне забезпечення для запропонованого методу. І наостанок необхідно провести аналіз отриманих результатів і зробити висновки.

**ІСНУЮЧІ МЕТОДИ**

Задачу комівояжера розв'язують трьома основними підходами: **точними, евристичними та метаевристичними методами**, кожен із яких має свої переваги й обмеження.

**Точні методи** забезпечують оптимальні рішення, але мають високу обчислювальну складність. Метод грубої сили перебирає всі можливі маршрути, що робить його непридатним для великих задач. Алгоритм Беллмана-Хелд-Карпа знижує складність завдяки динамічному програмуванню, проте масштабність його застосування обмежена.

**Евристичні методи**, як-от метод найближчого сусіда та подвійного обходу, забезпечують швидке отримання результату, але часто жертвують якістю рішення. Метод найближчого сусіда працює добре для рівномірно розподілених точок, але створює неоптимальні маршрути для нерівномірних даних. Подвійний обхід, заснований на мінімальному кістяковому дереві, покращує результат, але поступається точним алгоритмам.

**Метаевристичні методи** (імітація відпалу та генетичний алгоритм) демонструють високу ефективність для великих задач, але потребують значних обчислювальних ресурсів і ретельного налаштування параметрів.

**Геометричні методи**, як метод опуклої оболонки, будують маршрут на основі зовнішніх точок, що утворюють опуклу оболонку. Цей підхід є швидким, але часто включає зайві точки, що погіршує результат, особливо для нерівномірно розподілених даних, де внутрішні точки могли б значно покращити маршрут.

**ЗАПРОПОНОВАНИЙ ПІДХІД**

Можна побачити, що жоден з існуючих методів не фокусується на балансі між оптимальністю розв’язку та продуктивністю. Для вирішення цієї проблеми було запропоновано новий гібридний метод, який поєднує геометричну евристику для побудови початкового маршруту та локальну оптимізацію для його подальшого покращення. Основна ідея полягає у використанні крайніх точок набору даних для швидкого створення початкового маршруту. Північна, південна, західна та східна точки задають контур, що охоплює основні межі задачі та мінімізує кількість початкових точок. Це дозволяє значно зменшити обчислювальне навантаження на старті та вирішує проблему інших геометричних евристик.

Далі виконується локальна оптимізація методом повторної вставки. Кожна точка маршруту послідовно видаляється та вставляється у позицію, де приріст довжини маршруту є найменшим. Для цього використовується черга пріоритетів, що відбирає найкращих кандидатів для вставки на кожному кроці. Завдяки структурованому вибору початкового маршруту цей процес можна ефективно розпаралелити, що забезпечує швидке виконання навіть на великих наборах даних.

Наукова новизна полягає у використанні геометрично об́грунтованого методу побудови маршруту на основі крайніх точок. Це дозволяє швидко створити базовий контур задачі з мінімальною кількістю точок, що полегшує подальший процес оптимізації. Крім того, новим є сам запропонований гібридний метод розширення маршруту, що об’єднує чергу пріоритетів, паралельну обробку можливих вставок та локальну оптимізацію. Така комбінація забезпечує баланс між продуктивністю та якістю рішення, роблячи алгоритм ефективним і масштабованим.

**СХЕМА**

Розглянемо алгоритм більш детально. Загальну схему можна побачити на екрані

На першому етапі алгоритм визначає чотири ключові точки: найпівнічнішу, найпівденнішу, найзахіднішу та найсхіднішу. Ці точки обираються з усієї множини координат як базові, оскільки вони створюють найширший контур у графі, що дозволяє алгоритму почати з маршруту, який покриває основні геометричні межі. Ці точки формують початковий маршрут, і вони одразу позначаються як відвідані.

Далі виконується етап поступового розширення маршруту. Для цього використовується **черга пріоритетів**, яка дозволяє ефективно відбирати найбільш вигідні точки для вставки на основі приросту вартості маршруту. Спочатку в чергу додаються всі невідвідані точки, а їх пріоритет визначається за відстанню до останньої точки в поточному маршруті. Щоб зменшити обчислювальну складність, із черги обирається лише обмежена кількість найкращих кандидатів. Це обмеження дозволяє алгоритму суттєво скоротити час виконання, але може трохи вплинути на точність кінцевого рішення.

Для кожного обраного кандидата перевіряються всі можливі місця вставки в поточний маршрут з метою мінімізації приросту загальної вартості маршруту. Обчислення проводиться паралельно: кожен варіант вставки оцінюється незалежно, що значно пришвидшує процес на великих наборах даних.

Таким чином, хоча кількість кандидатів обмежена, для кожного з них все ще виконується повний перебір можливих позицій вставки, що додає обчислювальної складності. Загальна складність цього етапу залежить від кількості точок і приблизно дорівнює з додатковими оптимізаціями за рахунок використання пріоритетної черги і паралельної обробки.

Після завершення побудови початкового маршруту алгоритм переходить до етапу локальної оптимізації за допомогою методу повторної вставки. На цьому етапі кожна точка маршруту по черзі видаляється та вставляється назад у позицію, яка мінімізує приріст загальної довжини маршруту. Для визначення найкращої позиції враховується вартість вставки точки між кожною парою сусідніх міст. Якщо вставка точки призводить до зменшення загальної вартості маршруту, оновлений маршрут зберігається. Процес повторюється і триває доти, поки жодна операція не призводить до покращення маршруту.

Локальна оптимізація має складність для кожної ітерації через необхідність перевірки вставки кожної точки у всі можливі позиції.

Після завершення оптимізації алгоритм завершує маршрут, додаючи першу точку в кінець для замикання циклу, і обчислює його кінцеву вартість. Маршрут також переводиться у текстовий формат для виводу результату. Для зручності всі проміжні маршрути зберігаються, що дозволяє аналізувати, як поступово покращувалося рішення.

Важливо зазначити, що цей алгоритм не є точним, але він забезпечує добре наближене рішення. Загальна складність алгоритму є прийнятною для задач середнього розміру, але на дуже великих наборах даних може знадобитися подальша оптимізація або використання спрощених версій, наприклад іншої локальної оптимізації.

**ЗАСОБИ**

В якості засобів реалізації було обрано мову програмування С# на базі фреймворку .NET Core, через баланс між надійністю, зручністю розробки та ефективністю. В якості засобу для розробки інтерфейсу було обрано WPF, або ж Windows Presentation Foundation - фреймворк користувацького інтерфейсу для десктопних додатків на базі Windows. Для візуалізації результатів було обрано бібліотеку OxyPlot – це крос-платформна бібліотека з відкритим вихідним кодом для додатків .NET, яка забезпечує простий та ефективний спосіб створення різних типів візуалізації даних. Реалізація додатку у вигляді десктопного застосунку дозволила ефективно використовувати обчислювальні ресурси комп'ютера для складних алгоритмічних розрахунків.

**АРХІТЕКТУРА**

При розробці програмного забезпечення було обрано патерн **MVVM (Model-View-ViewModel)**, що забезпечує чітке розділення завдань між логікою додатка, інтерфейсом користувача та зв’язком між ними. Такий підхід спрощує обслуговування, тестування та розширення додатку.

Патерн MVVM складається з трьох основних компонентів (Model, View таViewModel)

У розробленому додатку **Model** зосереджена на даних задачі комівояжера, таких як координати міст, матриця відстаней, результати обчислень і самі класи алгоритмів.

**Другим компонентом є View** — це графічний інтерфейс користувача.

**Трєтім** **компонентом є** **ViewModel** — посередник між Model і View.

Він забезпечує прив’язку даних між логікою програми і елементами інтерфейсу, а також обробляє команди користувача.

Для обробки взаємодії користувача з інтерфейсом використовується Command патерн, що реалізований як клас RelayCommand. Він інкапсулює логіку дій та критерії доступності цих дій. Команди прив’язуються до елементів інтерфейсу через декларативну мову розмітки XAML, що забезпечує чистий і розділений код.

Основним класом додатку є **MainViewModel** – цецентральний компонент, який:

Координує виконання алгоритмів. Управляє станом програми. Реалізує прив’язку даних до елементів інтерфейсу та відповідає за сповіщення інтерфейсу про оновлення.

Також є клас **TSPAlgorithmBase**, це абстрактний клас, який імплементує інтерфейс I**TSPAlgorithm**:

Він забезпечує загальні властивості для всіх алгоритмів, такі як матриця відстаней і метод обчислення маршруту. Також містить спільні утиліти, зокрема для обчислення попарних відстаней між містами та для оцінки вартості маршруту.

**Далі є класи алгоритмів:**  
Кожен алгоритм реалізований у вигляді окремого класу, який наслідує TSPAlgorithmBase. Всього їх 6 штук. Кожен алгоритм реалізує окремий спосіб вирішення задачі комівояжера.

Також є класи **сервісів та утиліт**, які містятьнабір допоміжних методів для загальних операцій.

**СЦЕНАРІЇ ВИКОРИСТАННЯ**

Основні сценарії використання можна побачити на екрані, загалом після запуску додатку є 4 основні опції – вибір файлу з набором даних з провідника, вибір одного з трьох попередньо-визначених наборів, створення випадкового набору з кількістю точок яку вибере користувач та створення нового набору через інтерфейс додатку. Після вибору набору можна запустити будь-який з методів вирішення та після того як метод відпрацює можна буде побачити маршрут та набір графіків з результатами.

**ПРИКЛАД РОБОТИ**

На першому графіку акумулюються дані між усіма запусками алгоритмів на різних наборах даних, можна побачити як збільшення кількості міст впливає на час роботи. На другому графіку можна побачити відсоток від оптимального рішення, він показується якщо був запущений метод брут форс або якщо був запущений набір даних для якого відомий оптимум. На третьому графіку відповідність якості розвязку та часу, чим нижче та лівіше тим краще. І на четвертому графіку показники часу роботи алгоритму, середнє та граничні значення, так само як і на другому графіку.

**ПОРІВНЯННЯ АЛГОРИТМІВ**

Для порівняння були обрані п'ять алгоритмів. **Simulated Annealing (SA)**, **Genetic Algorithm (GA),** **Prim's Approximation**, **Convex Hull та** **Custom Algorithm** — тобто розроблений алгоритм.

Для тестування використовувалися набори даних різного масштабу та розподілу. Перші три набори включають 10, 11 і 12 точок із випадковим розподілом. Інші набори взяті з бібліотеки TSPLIB, в якій містяться задачі для яких знайшли оптимальне рішення. Ці набори відображають різноманітні сценарії: є набори з рівномірним та нерівномірним розподілом, зі скупченнями точок та з більш віддаленим розподілом, а також з середньою та великою кількістю точок. Деякі з наборів відображають реальні географічні сценарії.

Мета полягає в оцінці алгоритмів за двома ключовими показниками: якістю знайденого маршруту та часом виконання, що дозволяє визначити їх придатність для задач різної складності.

**ПОРІВНЯННЯ**

Розроблений алгоритм демонструє високу точність, досягаючи оптимуму на невеликих наборах.

Для реалістичних даних із TSPLIB, як-от pr144 і rat575 та kroE100, він значно перевершує **інші алгоритми** за якістю рішень.

На великих наборах, таких як a280, і nrw1379, розроблений алгоритм перевершує більшість інших алгоритмів за точністю, але трохи поступається алгоритму **Convex Hull** у конкретній задачі а280. Це можна пояснити специфічним розподілом точок та можливо недосконалістю локальної оптимізації.

**ПОРІВНЯННЯ**

На цій таблиці можна побачити ті самі дані але у представленні відсотку від оптимальності.

**ПОРІВНЯННЯ**

**На цій таблиці в нас час виконання у мілісекундах.**

**У загальному випадку розроблений алгоритм** демонструє збалансовану продуктивність, обчислюючи маршрути значно швидше, ніж SA і GA на середніх та великих наборах.

В порівнянні з алгоритмом на основі апроксимації Прима можна побачити кратну різницю в часі виконання на користь апроксимації, але це нівелюється значно кращими результатами оптимальності. В порівнянні з алгоритмом **Convex Hull** розроблений алгоритм має нижчу продуктивність, але це десятки або сотні мілісекунд, що є дуже незначним часом виконання для задач відповідного розміру. Варто зазначити що час стає дійсно значним у задачі **nrw1379,** тому застосовність розробленого алгоритму найбільш доцільна приблизно до 800 точок.

**Поліноміальний час** — це час виконання алгоритму, який можна виразити як **поліном** від розміру вхідних даних n.

Формально це означає, що складність алгоритму має вигляд де k — деяка константа.

**Коротко:** Якщо час зростає повільніше за експоненційну залежність і підпорядковується степеневій функції, це поліноміальний час. Наприклад, .

**Відбір** обирає найкращі рішення на основі їхньої якості; **кросовер**, комбінує частини рішень двох "батьків" для створення нових "нащадків"; **мутація** додає випадкові зміни для збереження різноманітності і уникнення локальних мінімумів

**Кістякове дерево** зв'язаного неорієнтованого [графа](https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) — [ациклічний](https://uk.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB_(%D1%82%D0%B5%D0%BE%D1%80%D1%96%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%B2)) (тобто, без циклів) зв'язний [підграф](https://uk.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B2%D0%BD%D0%B8%D0%BA_%D1%82%D0%B5%D1%80%D0%BC%D1%96%D0%BD%D1%96%D0%B2_%D1%82%D0%B5%D0%BE%D1%80%D1%96%D1%97_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%B2#П) цього графа, який містить усі його вершини. Неформально кажучи, кістякове дерево складається з деякої [підмножини](https://uk.wikipedia.org/wiki/%D0%9F%D1%96%D0%B4%D0%BC%D0%BD%D0%BE%D0%B6%D0%B8%D0%BD%D0%B0) [ребер](https://uk.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B2%D0%BD%D0%B8%D0%BA_%D1%82%D0%B5%D1%80%D0%BC%D1%96%D0%BD%D1%96%D0%B2_%D1%82%D0%B5%D0%BE%D1%80%D1%96%D1%97_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%B2#Р) графа, таких, що рухаючись цими ребрами можна з будь-якої вершини графа потрапити до будь-якої іншої.

**Мінімальне кістякове дерево -** це [кістяк](https://uk.wikipedia.org/wiki/%D0%9A%D1%96%D1%81%D1%82%D1%8F%D0%BA%D0%BE%D0%B2%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE) цього [графу](https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), що має мінімальну можливу вагу, де під вагою [дерева](https://uk.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%82%D0%B5%D0%BE%D1%80%D1%96%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%B2)) розуміється сума ваг його ребер.

**НАУКОВА НОВИЗНА**

Побудова початкового маршруту на основі крайніх точок є новим підходом, який раніше не пропонувався для вирішення задачі комівояжера. Цей метод дозволяє одразу сформувати **геометричний контур**, що охоплює всі основні межі набору точок. На відміну від інших геометричних евристик, які додають забагато точок на стартовому етапі, мій підхід мінімізує початкову кількість точок, що **спрощує подальший пошук глобального оптимуму** і зменшує обчислювальне навантаження.

Використання черги пріоритетів для відбору обмеженої кількості найкращих кандидатів і паралельної обробки можливих вставок стало можливим завдяки структурованому вибору початкового маршруту. Геометричний контур, побудований на основі крайніх точок, створює логічну базу для ефективної роботи цих оптимізацій. На відміну від випадкових або послідовних методів побудови маршруту, які не можна фізично розпаралелити через залежність ітерацій, мій підхід дозволяє незалежно оцінювати вставку кожного кандидата в різні позиції маршруту

Таким чином, новизна полягає як у **геометрично обґрунтованій побудові початкового маршруту**, так і у **гібридному методі** розширення маршруту, який об'єднує чергу пріоритетів, паралельну обробку та локальну оптимізацію.