ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ОЛЕСЯ ГОНЧАРА

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ КАФЕДРА МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ ЕОМ

Кваліфікаційна робота

перший (бакалаврський) рівень вищої освіти спеціальність 121 Інженерія програмного забезпечення освітня програма Інженерія програмного забезпечення

РОЗРОБЛЕННЯ ANDROID-ДОДАТКУ ДЛЯ КОУЧСЕРФІНГУ

Виконавець

студент групи ПЗ-21у-1 Войцехов Микола Олександрович

(підпис)

Керівник

доцент кафедри ЕОМ

Ємел’яненко Тетяна Георгіївна

(підпис)

В.о. завідувача кафедри МЗ ЕОМ, д-р техн. наук, професор

Байбуз Олег Григорович

(підпис)

Дніпро – 2024

**ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ОЛЕСЯ ГОНЧАРА**

Факультет прикладної математики Кафедра математичного забезпечення ЕОМ Рівень вищої освіти перший (бакалаврський) Спеціальність 121 Інженерія програмного забезпечення Спеціалізація Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри математичного забезпечення ЕОМ

*Олег БАЙБУЗ*

(підпис)

« » 2023 року

***З А В Д А Н Н Я***

# НА КВАЛІФІКАЦІЙНУ РОБОТУ

Войцехову Миколі Олександровичу

(Прізвище, ім’я, по-батькові студента)

1. Тема роботи Розроблення Android-додатку для коучсерфінгу

керівник роботи Ємел’яненко Тетяна Георгіївна, доцент кафедри ЕОМ

(Прізвище, ім’я, по-батькові, науковий ступінь, вчене звання)

затверджені наказом по Університету від « 19 » травня 2023 року №457 C

1. Термін подання роботи 16 червня 2023 р.
2. Вхідні дані до роботи текстові файли з данними для задачі комівояжера та задачі маршрутизації транспортних засобів.
3. Перелік питань, які потрібно розробити
4. Провести огляд існуючих підходів до розв’язання задачі комівояжера та задачі маршрутизації транспортних засобів.
5. Розробити бібліотеку класів для розв’язання зазначених задач методом мурашиної колонії.
6. Розробити графічний інтерфейс для візуалізації отриманих розв’язків
7. Провести дослідження роботи програмного забезпечення на відомих наборах даних
8. Керівник, консультант з окремих (спеціальних) розділів роботи

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Ініціали прізвище та посада  керівника, консультанта | Підпис, дата | |
| завдання видав | завдання прийняв |
| 1 | Т.Г. Ємел’яненко, доцент кафедри ЕОМ |  |  |
| 2 | Т.Г. Ємел’яненко, доцент кафедри ЕОМ |  |  |
| 3 | Т.Г. Ємел’яненко, доцент кафедри ЕОМ |  |  |
| 4 | Т.Г. Ємел’яненко, доцент кафедри ЕОМ |  |  |

1. Дата видачі завдання

# КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Зміст етапів роботи | Термін виконання етапів роботи | Примітка |
| 1. | Проведення огляду існуючих підходів до розв’язання задачі комівояжера та задачі маршрутизації транспортних  засобів |  |  |
| 2. | Детальне ознайомлення з алгоритмом мурашиної колонії |  |  |
| 3. | Розроблення алгоритму мурашиної колонії для задачі комівояжера та задачі маршрутизації транспортних засобів |  |  |
| 4. | Розроблення графічного інтерфейсу |  |  |
| 5. | Проведення обчислювальних експериментів на тестових наборах даних |  |  |
| 6. | Оформлення кваліфікаційної роботи |  |  |

|  |  |  |
| --- | --- | --- |
| Студент | (підпис) | Микола ВОЙЦЕХОВ  (Власне ім’я ПРІЗВИЩЕ) |
| Керівник роботи | (підпис) | Тетяна ЄМЕЛ’ЯНЕНКО  (Власне ім’я ПРІЗВИЩЕ) |

**РЕФЕРАТ**

**Кваліфікація робота:** 46 c., 11 рис., 1 табл., 9 джерел.

**Об’єкт дослідження:** метаевристичні методи розв’язання задачі комівояжера та асиметричної задачі маршрутизації транспортних засобів.

**Мета розробки:** розробити програмне забезпечення для розв’язання задачі комівояжера та асиметричної задачі маршрутизації транспортних засобів з обмеженою місткістю методом мурашиної колонії.

**Одержані висновки та їх новизна:** розроблено програмне забезпечення, що реалізує алгоритми мурашиної колонії для розв’язання задачі комівояжера і задачі маршрутизації транспортних засобів методом мурашиної колонії; проведено тестування розроблених алгоритмів на тестових наборах даних; виявлені переваги і недоліки наведеного алгоритму, знайдені шляхи покращення алгоритму.

**Реалізація:** програмне забезпечення написано на С# використовуючи платформу .NET 7.0 у середовищі розробки Microsoft Visual Studio 2022.

**Результати дослідження можуть бути використані** для диспетчеризації транспортування товарів.

**Ключові слова:** ЗАДАЧА КОМІВОЯЖЕРА, ПРОБЛЕМА МАРШРУТИЗАЦІЇ ТРАНСПОРТНИХ ЗАСОБІВ З ОБМЕЖЕНОЮ ЄМНІСТЮ, МЕТОД МУРАШИНОЇ КОЛОНІЇ.

# ANNOTATION

The graduation research of the 4-year student V. Soloviov (Oles Honchar Dnipro National University, Faculty of Applied Mathematics, Department of Mathematical Support of Computers), focuses on the development of software solutions for two well-known optimization problems: the Traveling Salesman Problem (TSP) and the Asymmetric Capacitated Vehicle Routing Problem (ACVRP). Specifically, the research aims to utilize the Ant Colony Optimization (ACO) algorithm to solve these problems effectively.

The primary objective of the research is to adapt an ACO algorithm to handle the complexities associated with the ACVRP and TSP. Both problems are known to be computationally challenging and have real-world applications in various fields such as logistics, transportation, and supply chain management.

The ACVRP involves finding optimal routes for multiple vehicles with different capacities to serve a set of customers, considering the capacity limitations of each vehicle and the asymmetric nature of the problem (where the travel distance between two points may vary depending on the direction taken). On the other hand, the TSP focuses on determining the shortest possible route that visits a set of cities exactly once and returns to the starting point, with the goal of minimizing the overall distance traveled.

By adapting the ACO algorithm for these two problems, V. Soloviov's research aims to provide an efficient solution. The performance of the developed algorithm will be evaluated and compared with existing approaches for solving the ACVRP and TSP. This comparison will likely involve benchmarking the algorithm on established problem instances and assessing factors such as solution quality, computation time, and scalability.

The outcomes of this work can have practical implications for industries that deal with routing and logistics optimization, offering more efficient and effective solutions to their complex problems.

Pages 46, Bibliography 9, pictures 11, tables 1.

# ЗМІСТ

[ПОСТАНОВКА ЗАДАЧІ 8](#_bookmark0)

1. [ТЕОРЕТИЧНІ ВІДОМОСТІ 9](#_bookmark1)
   1. [Евристичні алгоритми 9](#_bookmark2)
   2. [Метаевристичні алгоритми 10](#_bookmark3)
   3. [Обґрунтування вибору методу розв’язання задачі 11](#_bookmark4)
2. [АЛГОРИТМ МУРАШИНОЇ КОЛОНІЇ ДЛЯ ЗАДАЧІ КОМІВОЯЖЕРА. 13](#_bookmark5)
   1. [Принцип роботи алгоритму 13](#_bookmark6)
   2. [Недоліки і обмеження 14](#_bookmark7)
   3. [Основні етапи роботи ACO 15](#_bookmark8)
      1. [Формула вибору шляху мурахою 16](#_bookmark9)
      2. [Оновлення феромонів 17](#_bookmark10)
      3. [Алгоритм Пріма 18](#_bookmark11)
3. [ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 19](#_bookmark12)
   1. [Структура проєкту 19](#_bookmark13)
   2. [Класи для задачі комівояжера 21](#_bookmark14)
   3. [Класи для задачі маршрутизації транспортних засобів 26](#_bookmark15)
   4. [Опис деяких допоміжних класів 31](#_bookmark16)
   5. [Графічний інтерфейс 35](#_bookmark17)
4. [АНАЛІЗ РЕЗУЛЬТАТІВ 37](#_bookmark18)
   1. [Параметри 37](#_bookmark19)
   2. [Результати роботи алгоритму для задачі комівояжера 38](#_bookmark20)
   3. [Результати роботи алгоритму для задачі маршрутизації транспортних](#_bookmark21) [засобів 41](#_bookmark21)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 44](#_bookmark22)

[Додаток А](#_bookmark23). [Приклад вихідних даних для задачі комівояжера 45](#_bookmark24)

[Додаток Б.](#_bookmark25) [Приклад вихідних даних для задачі маршрутизації транспортних](#_bookmark26) [засобів 46](#_bookmark26)

**ВСТУП**

У сучасному світі, мобільні технології стають невід'ємною частиною нашого повсякденного життя, вже майже кожна людина з народження контактує з ними. Багато інновацій прийшли до нас за останні декілька років, які спростили наше життя. Серед таких інновацій особливе місце займає коучсерфінг – це соціальна практика, що дозволяє мандрівникам знаходити тимчасове безкоштовне житло та підтримку від місцевих жителів, щоб більше зануритися у місцеві традиції. Розробка мобільного додатку для Android, що сприятиме ефективному зв'язку та взаємодії між користувачами у цій сфері, є актуальною та перспективною задачею.

Цей дипломний проект має на меті створити надійний, інтуїтивно зрозумілий додаток для коучсерфінгу, який дозволить користувачам легко знаходити господарів і гостей, обмінюватись відгуками. Основна увага приділяється розробці зручного користувацького інтерфейсу, щоб кожна людина змогла його використовувати.

Важливість розробки такого додатку також підкреслюється швидким розвитком мобільних технологій та зростанням популярності культури незалежних подорожей. Використання цифрових платформ для сприяння культурному обміну і розширення можливостей для мандрівників стає не просто зручністю, а необхідністю. Тому цей проект відіграє ключову роль у розвитку цифрових сервісів у галузі туризму.

Проект сприятиме не тільки технічному, але й соціальному прогресу, дозволяючи людям з усього світу знаходити нових друзів, досліджувати нові культури та робити свої подорожі більш приємними та економічно вигідними. Ця робота є кроком до створення більш відкритого та з'єднаного світу, де кожен може стати частиною глобальної спільноти мандрівників.

# ПОСТАНОВКА ЗАДАЧІ

Основною метою є розробка мобільного додатку, який надаватиме користувачам зручний інтерфейс, широкі функціональні можливості для пошуку житла та спілкування, а також високий рівень безпеки під час взаємодії між гостями та господарями.

Додаток повинен містити наступний функціонал:

1. Система профілю(реєстрація та авторизація);
2. Система рейтингу – після дати закінчення оренди, у обох сторін з’являється можливість виставити оцінку;
3. Система скарг­ ­– якщо на користувача за останній час прийшло багато скарг(більше 5 скарг за останній місяць) то профіль буде заблоковано. Тільки адміністратор зможе розблокувати;
4. Система оренди – з можливістю наглядно на мапі подивитися, де знаходиться житло;
5. Пошук профіля та об’яв( та їх сортування);
6. Система адміністратора;
7. Система сповіщень ­– орендодавець та орендар, за день та в день оренди будуть отримувати сповіщення.

# ТЕОРЕТИЧНІ ВІДОМОСТІ

Коучсерфінг, як соціальна практика, набуває все більшої популярності, проте існуючі платформи для коучсерфінгу не завжди відповідають зростаючим вимогам користувачів щодо зручності, безпеки та функціональності. Відтак, розробка нового мобільного додатку для коучсерфінгу, який буде враховувати сучасні тенденції та потреби користувачів, є актуальною і важливою задачею.

Основною метою даної дипломної роботи є створення мобільного додатку, що надаватиме користувачам зручний інтерфейс, широкий функціонал для пошуку житла і спілкування, а також високий рівень безпеки під час взаємодії між гостями і господарями. Це передбачає розробку системи профілів, рейтингу, скарг, оренди, пошуку та сортування оголошень, адміністратора, а також системи сповіщень.

## Огляд існуючих рішень

Couchsurfing є найбільш відомою та широко використовуваною платформою для коучсерфінгу, яка з'єднує мандрівників з місцевими жителями, які готові безкоштовно прийняти їх у своїх домівках. Сайт та мобільний додаток дозволяють користувачам шукати господарів, встановлювати з ними контакт та планувати свої подорожі.

Преваги**:**

* 1. Величезна глобальна спільнота, що легко дозволяє знайти господаря або гостя майже у будь-якому куточку світу.
  2. Різноманітні можливості для взаємодії та обміну культурним досвідом.
  3. Функції безпеки, включаючи верифікацію профілів, систему відгуків та рекомендацій.

Недоліки**:**

* 1. Необхідність платити за отримання статусу верифікованого користувача.
  2. Можливі складнощі з навігацією та використанням інтерфейсу сайту або додатку.
  3. Іноді зустрічаються скарги на проблеми з безпекою та конфіденційністю.

BeWelcome — це некомерційна платформа для коучсерфінгу, яка націлена на сприяння культурному обміну та гостинності по всьому світу. Вона зосереджена на створенні довірливих взаємовідносин між користувачами та пропонує зручні інструменти для пошуку місць проживання та спілкування.

Переваги**:**

* 1. Безкоштовне користування всіма функціями, включаючи верифікацію.
  2. Сильне почуття спільноти та підтримки серед користувачів.
  3. Відкритість та некомерційний характер платформи.

Недоліки**:**

* + 1. Менша база користувачів у порівнянні з Couchsurfing.
    2. Обмежені функціональні можливості та інструменти на сайті.

Trustroots створена з метою об'єднання людей, які поділяють цінності довіри, відкритості та взаємодопомоги. Це платформа спрямована на мандрівників, які шукають більш глибокого занурення у місцеві культури та спільноти.

Преваги**:**

* 1. Фокус на створенні міцних взаємин та довіри між користувачами.
  2. Простий та легкий у використанні інтерфейс.
  3. Повністю безкоштовний доступ до всіх функцій.

Недоліки**:**

* 1. Відносно мала кількість користувачів.
  2. Менше географічне покриття, особливо в менш популярних регіонах.
  3. Обмежені можливості для верифікації користувачів.

Кожна з цих платформ має свої унікальні особливості та відповідає певним потребам користувачів. Вибір платформи залежить від індивідуальних уподобань, цілей подорожі та бажаного рівня взаємодії з місцевими жителями та іншими мандрівниками.

## Огляд існуючих технологій та засобів мобільної розробки

**Android SDK (Software Development Kit)** є основним інструментарієм для розробки додатків під Android ОС. Він включає набір розробницьких інструментів, таких як дебагер, бібліотеки, емулятор пристрою, документацію, приклади коду, а також специфічні API для використання функцій Android пристроїв. Розробники можуть використовувати Android SDK через Android Studio, офіційне інтегроване середовище розробки (IDE) від Google, що спрощує процес розробки і тестування додатків.

Переваги:

* 1. Широкі можливості для розробки і тестування додатків.
  2. Підтримка від Google та великої спільноти розробників.
  3. Регулярні оновлення та вдосконалення інструментів.

Недоліки:

* 1. Вимагає часу на вивчення та адаптацію до постійно оновлюваних інструментів.
  2. Може виявитися складним для новачків у мобільній розробці.

**Java** — це універсальна, об'єктно-орієнтована мова програмування, яка використовується для розробки Android додатків. Є однією з найпопулярніших мов програмування для Android, завдяки своїй портативності, безпеці та легкості у використанні. Java дозволяє розробникам писати код, який може виконуватися на різних платформах без змін, що робить її ідеальною для мобільної розробки.

**Переваги:**

* 1. Велика спільнота та багато ресурсів для навчання.
  2. Висока переносимість коду між різними платформами.
  3. Сильна підтримка з боку розробників інструментів та бібліотек.

Недоліки:

* 1. В порівнянні з Kotlin, може мати більш високу складність і меншу виразність коду.
  2. Може вимагати більше ресурсів системи та часу на виконання.

Flutter — це відкритий фреймворк від Google для створення високоякісних нативних інтерфейсів на iOS та Android з єдиного кодової бази. Flutter використовує мову програмування Dart і забезпечує високу продуктивність додатків завдяки компіляції в нативний код.

Переваги:

* 1. Єдина кодова база для iOS та Android, що значно скорочує час розробки.
  2. Велика кількість готових до використання віджетів та висока продуктивність додатків.
  3. Гаряча перезавантаження (hot reload) для швидкого тестування змін у коді.

Недоліки:

* 1. Вимагає вивчення мови програмування Dart, яка менш відома порівняно з Java або Kotlin.
  2. Можуть виникати складнощі при інтеграції з деякими нативними бібліотеками або при використанні специфічних платформних можливостей.

## Обґрунтування вибору обраних технологій та засобів розробки

Flutter є відкритим фреймворком від Google для створення високоякісних нативних інтерфейсів на iOS і Android з єдиного коду. Він використовує мову програмування Dart і надає розробникам потужний набір інструментів та віджетів для створення інтуїтивно зрозумілих та візуально привабливих мобільних додатків.

Переваги:

Швидка розробка: Гаряче перезавантаження дозволяє миттєво побачити результати змін у коді.

Кросплатформність: Єдиний кодовий базис для розробки додатків під Android та iOS.

Висока продуктивність: Компіляція у нативний код забезпечує високу продуктивність додатків.

Гнучкий UI: Великий вибір віджетів та можливість створення складних UI елементів.

Недоліки:

Вивчення нової мови: Необхідність вивчення Dart, хоча вона і легка для освоєння.

Розмір додатку: Додатки на Flutter можуть бути більшими за розміром порівняно з нативними.

Обмежений доступ до деяких нативних функцій: Може вимагати додаткової роботи для інтеграції специфічних нативних функцій[2].

PHP — це популярна загальноцільова скриптова мова програмування, яка активно використовується для розробки веб-сайтів. PHP легко інтегрується з HTML і використовується на серверній стороні, що робить його ідеальним для розробки динамічних веб-сайтів і веб-додатків.

Переваги:

Широка підтримка: Підтримується більшістю хостингових провайдерів.

Гнучкість: Легко інтегрується з різними базами даних, такими як MySQL, PostgreSQL та іншими.

Велика спільнота: Багато ресурсів для навчання, фреймворків та готових рішень.

Простота вивчення: Легко почати для новачків у програмуванні.

Недоліки:

Безпека: Вимагає ретельного написання коду для уникнення вразливостей.

Продуктивність: У деяких сценаріях може працювати повільніше порівняно з іншими мовами програмування[3].

MySQL — це найпопулярніша система управління базами даних SQL (СУБД), яка використовується у всьому світі для зберігання, обробки та отримання даних. Є відкритим програмним забезпеченням і часто використовується разом з PHP для розробки веб-додатків.

Переваги:

Висока надійність та стабільність: Масштабується для обробки великих обсягів даних.

Широка підтримка: Інтегрується з багатьма мовами програмування та платформами.

Легкість використання: Простий у вивченні синтаксис SQL для управління даними.

Безпека: Включає розширені функції безпеки для захисту даних.

Недоліки:

Обмеження в розподіленій обробці даних: Не найкращий вибір для розподілених систем баз даних порівняно з NoSQL рішеннями.

Складність оптимізації: Для великих та складних баз даних може вимагати додаткових зусиль для оптимізації продуктивності.

# ОПИС РОЗРОБЛЕННОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Аналіз вимог до програмного забезпечення

Під час аналізу вимог до програмного забезпечення коучсерфінга було проведено глибоке обговорення з потенційними користувачами та зацікавленими сторонами з метою зрозуміти їх потреби та очікування від додатку. Основні вимоги до програмного забезпечення визначено наступним чином:

1. Реєстрація/Авторизація: Користувач має можливість створити обліковий запис, для цього йому потрібно ввести дані про себе, пошту та пароль, після чого отримає листа на пошту та повинен ввести код. Для авторизації потрібно ввести пароль та пошту.

2. Відновлення аккаунту: Для відновлення аккаунту користувачу потрібно вказати пошту, після чого отримати листа на пошту та вказати код. Далі буде можливість змінити пароль.

3. Система профілю: Користувач має профіль де відображається основна інформація про нього, а також його рейтинг. Він має можливість вийти з аккаунту або видалити. Також користувачі мають можливість шукати інших людей та переглядати їх профілі.

4. Створення об’яви: Користувач має можливість додати нову об’яву, вказати адресу, кількість людей, дату заселення та виселення, додати фото домівки. З можливістю подальшого редагування та видалення.

5. Перегляд розміщених об’яв: При перегляді об’яв користувач має змогу сортувати, за датою створенням об’яви, за кількістю людей. Також шукати за країною або містом. Для більшої інформації о розташуванні житла, користувач має можливість переглянути на мапі та подивитися погодні умови на час прибування в обраному місці.

6. Система скарг: Користувачі мають можливість шукати облікові записи інших користувачів та залишати на них скарги. Якщо на користувача прийшло більше 5 скарг за останній місяць, то його обліковий запис буде заблоковано. Для подальшого використання потрібно зв’язатися з адміністратором.

7. Система рейтингу: Після завершення дати об’яви для обох сторін стає доступна можливість оцінювання по 5 бальній шкалі.

8. Система сповіщення: За день та в день оренди користувачам буде приходити нагадування про оренду.

9. Система адміністратора: адміністратор має можливість видаляти об’яви та користувачів, переглядати жалоби, розблоковувати користувачів та відправляти їм повідомлення.

## Опис архітектури програмного забезпечення

Додаток розроблено на основі сучасних технологій: Flutter для розробки мобільного інтерфейсу, PHP для обробки серверної логіки та MySQL для зберігання даних. Ця комбінація технологій забезпечує високу продуктивність, масштабованість та гнучкість додатку.

Архітектура клієнтської частини (Flutter)

Flutter дозволяє створювати високопродуктивний та візуально привабливий інтерфейс користувача для мобільних платформ (iOS та Android) з єдиного кодової бази. Архітектура клієнтської частини базується на використанні віджетів, які є основними будівельними блоками для створення користувацького інтерфейсу в Flutter.

Використовується архітектурний патерн BLoC (Business Logic Component), який дозволяє ефективно управляти станом додатку, розділяючи інтерфейс користувача та бізнес-логіку. BLoC спрощує розробку та тестування додатку, забезпечуючи чисту архітектуру та високу масштабованість.

Архітектура серверної частини (PHP)

Серверна частина реалізована на PHP, мові програмування, яка широко використовується для розробки веб-додатків. PHP пропонує широкі можливості для створення динамічних веб-сторінок і взаємодії з базами даних.

## Use Case діаграма

Діаграма варіантів використання – це динамічна діаграма або діаграма поведінки в UML. Діаграми варіантів використання моделюють функціональність системи з використанням діючих осіб та сценаріїв використання. Варіанти використання – це набір дій, функцій, які повинна виконувати система. «Діючі особи» - люди, організації або програмне забезпечення, що діють під певними ролями в системі. Відповідно до кожного актора були визначені їх сценарії взаємодії з мобільним додатком, які відповідають всім визначеним вимогам, що були визначені в аналізі вимог до програмного продукту.

Для розробки діаграми варіантів використання Android-додатку були визначені актори :

− Admin – адміністратор, що представляє особу, що буде слідкувати за правилами платформи, блокувати користувачів, видаляти об’яви, що містять не допустимий контент;

− Customer – користувач додатку.

Після формування переліку акторів наступним етапом необхідно сформувати перелік сценаріїв :

− Реєстрація/Авторизація;

− Відновлення паролю;

− Перегляд профілю;

− Редагування/Видалення профілю;

− Створення об’яв;

− Редагування/Видалення об’яв;

− Відклик на об’яву;

− Блокування/Розблокування користувача;

− Залишення відгуку на об’яву;

− Відправлення скарг на інших користувачів.

На рис. 2.5 представлено діаграма варіантів використання для веб– додатку.



Рисунок 2.1– Use Case діаграма

# ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Структура проєкту

Основна частина програми являє собою бібліотеку класів (рис 3.1), яка може використовуватись для розв'язання задач TSP та VRP. Основний алгоритм суттєво відрізняється для TSP та VRP, тому уніфікувати мурах не виявилось можливим.

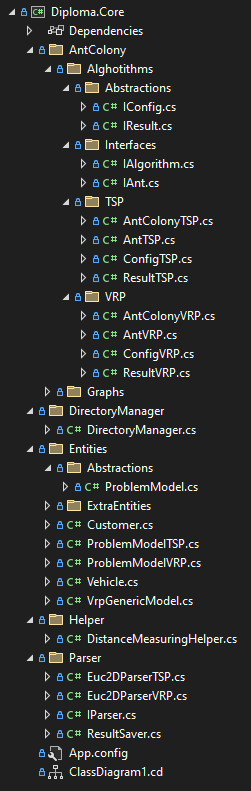


Рисунок 3.1– Загальна структура основного проєкту

Далі наведені діаграми класів, що використовуються для розв'язання TSP (рис. 3.2) та VRP (рис. 3.3).

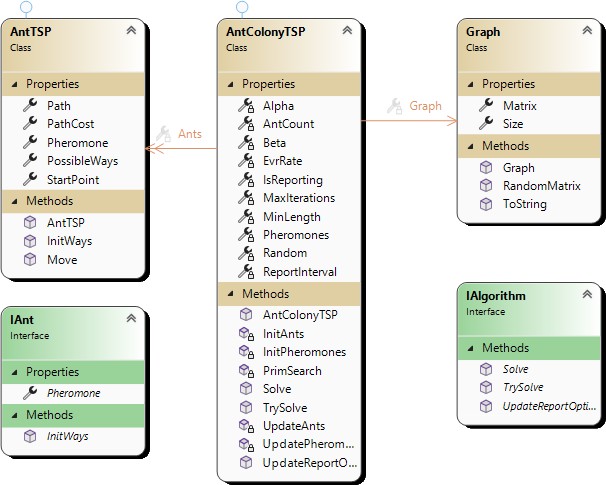


Рисунок 3.2– Фрагмент діаграми класів для розв'язання задачі Traveling

Salesman Problem

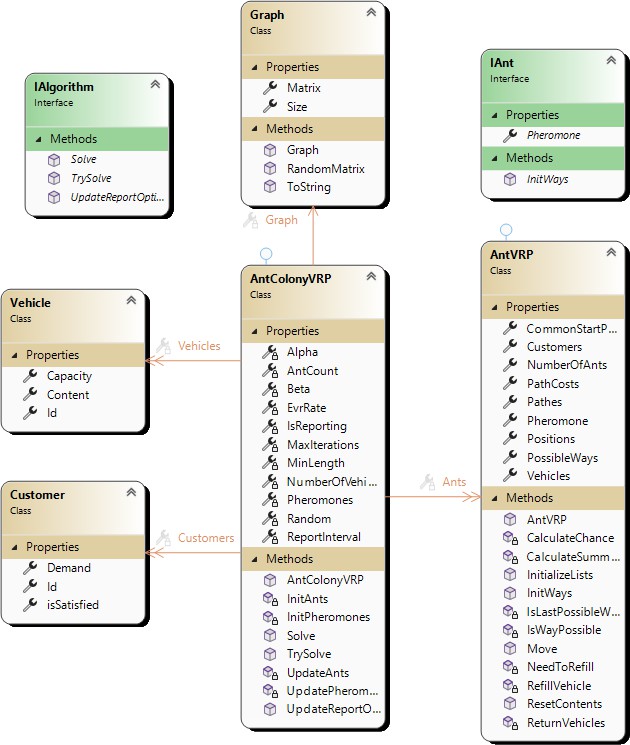


Рисунок 3.3– Фрагмент діаграми класів для розв'язання задачі Vehicle

Routing Problem

## Класи для задачі комівояжера

* 1. Клас ***AntColonyTSP*** реалізує інтерфейс IAlgorithm і представляє алгоритм мурашиного колонії для задачі комівояжера.

## Поля класу:

* + - *Ants*: список мурах (об'єкти типу AntTSP).
    - *Pheromones*: двовимірний масив засобів маркування феромонами між вершинами графа.
    - *Graph*: граф, над яким проводиться обчислення.
    - *MaxIterations*: максимальна кількість ітерацій алгоритму.
    - *Alpha*: параметр альфа для розрахунку ймовірностей переходу мурахи.
    - *Beta*: параметр бета для розрахунку ймовірностей переходу мурахи.
    - *AntCount*: кількість мурах в колонії.
    - *EvrRate*: швидкість випаровування феромонів.
    - *MinLength*: найкоротша відстань, знайдена за допомогою алгоритму Пріма.
    - *Random:* об'єкт Random для генерації випадкових чисел.

## Методи класу:

* + - *UpdateReportOptions(bool isReporting, int reportInterval)*: оновлює параметри звітування алгоритму.
    - *TrySolve(out Result result)*: спробувати розв'язати задачу комівояжера за допомогою алгоритму. Повертає true, якщо розв'язок успішно знайдений, і передає результат через параметр result.
    - *Solve()*: запускає алгоритм мурашиного колонії для задачі комівояжера. Повертає результат у вигляді об'єкту ResultTSP.
    - *PrimSearch()*: виконує алгоритм Пріма для знаходження найкоротшого шляху в графі. Повертає найкоротшу відстань.
    - *UpdatePheromone()*: оновлює рівні феромонів в графі після кожної ітерації.
    - *UpdateAnts()*: оновлює початкові значення мурах перед початком нової ітерації.
    - *InitAnts()*: створює та ініціалізує список мурах.
    - *InitPheromones()*: створює та ініціалізує масив феромонів між вершинами графа.
  1. Клас ***AntTSP***, який реалізує інтерфейс IAnt.

## Поля:

*StartPoint* - початкова точка мурашки. Це ціле число, яке визначає індекс початкової точки мурашки в графі.

*Pheromone* - феромон мурашки. Це незмінне ціле число, яке відображає рівень феромонів мурашки.

*PathCost* - вартість поточного шляху мурашки. Це дійсне число, яке відображає сумарну вартість шляху мурашки в графі. Початково воно встановлено на 0.

*Path* - список, який містить послідовність відвіданих вершин мурашкою. Це список цілих чисел, що відображає послідовність індексів вершин у графі. *PossibleWays* - список, який містить можливі шляхи, які мурашка може вибрати. Це список цілих чисел, які представляють індекси вершин, доступних

для вибору мурашкою.

## Конструктор:

* *AntTSP(int startPoint, int pheromone)*: конструктор класу AntTSP. Приймає початкову точку startPoint та рівень феромонів pheromone. Ініціалізує поля StartPoint, Pheromone, Path та PossibleWays.

## Методи:

* *List<int> InitWays(int size)*: ініціалізує список можливих шляхів PossibleWays. Приймає розмір графу size і повертає список цілих чисел, який містить всі можливі шляхи мурашки в графі, крім початкової точки.
* *void Move(Graph graph, double[,] pheromones, int beta, int alpha)*: виконує рух мурашки в графі. Приймає об'єкт graph типу Graph,

який представляє граф, матрицю pheromones, яка містить рівні феромонів між вершинами графу, і параметри beta та alpha, які використовуються для визначення шансу вибору наступної вершини. Метод вираховує наступну вершину для мурашки, оновлює PathCost згідно вибраного шляху та оновлює PossibleWays, видаляючи вже вибрані вершини.

* 1. Клас **ConfigTSP** є похідним від базового класу Config. У ньому оголошені два конструктори та метод ToString().
     + Конструктор *ConfigTSP()*: це конструктор за замовчуванням, який не приймає жодних аргументів.
     + Конструктор *ConfigTSP(int MaxIterations, int Alpha, int Beta, double Rho, int AntCount)*: цей конструктор приймає аргументи MaxIterations, Alpha, Beta, Rho та AntCount, щоб задати значення полів об'єкта ConfigTSP. Значення аргументів використовуються для ініціалізації відповідних полів класу ConfigTSP.
     + Метод *ToString()*: цей метод перевизначає метод ToString() базового класу Config. Він повертає рядок, що містить рядок з форматованою інформацією про значення полів об'єкта ConfigTSP. Рядок містить значення Alpha, Beta, Rho, AntCount та MaxIterations. Значення полів отримуються з відповідних полів об'єкта ConfigTSP за допомогою ключового слова this.
  2. Клас **ResultTSP** успадковує від класу Result. Цей клас містить наступні поля та методи:
     + *public TimeSpan Time { get; set; }*: властивість, що представляє час, який знадобився для виконання алгоритму.
     + *public double PathCost { get; set; }*: властивість, що представляє вартість (відстань) найкращого шляху.
     + *public List<int> BestPath { get; set; }*: властивість, що містить список цілочисельних значень, які представляють найкращий шлях.
     + *public int BestIteration { get; set; }*: властивість, що представляє номер ітерації, на якій був знайдений найкращий шлях.
     + *public int CurrentIteration { get; set; }*: властивість, що представляє поточну ітерацію алгоритму.
     + *public DateTime StartTime { get; set; }*: властивість, що представляє час початку виконання алгоритму.
     + *public TimeSpan OptimalTime { get; set; }*: властивість, що представляє оптимальний час виконання алгоритму.
     + *public void SetTime()*: метод, який встановлює час виконання (Time) шляхом обчислення різниці між поточним часом (DateTime.Now) та часом початку виконання (StartTime).
     + *public override string ToString()*: перевизначений метод ToString(), який повертає рядкове представлення об'єкту класу. Рядок містить інформацію про поточну ітерацію, номер ітерації, час виконання, поточний шлях, найкращий шлях та їх вартість.
     + *public string ToStringSimplified()*: метод, який повертає спрощене рядкове представлення об'єкту класу. Рядок містить інформацію про номер ітерації, вартість найкращого шляху, оптимальний час виконання та загальний час виконання.

Цей клас і його методи використовуються для представлення результатів розв'язку задачі комівояжера. Він містить інформацію про шлях, вартість шляху, час виконання алгоритму та інші деталі, які допомагають аналізувати та відображати результати розв'язку задачі.

## Класи для задачі маршрутизації транспортних засобів

1. Клас ***AntColonyVRP*** реалізує інтерфейс IAlgorithm і представляє алгоритм мурашиного колонії для розв'язання задачі маршрутизації транспорту. Він має такі поля та методи:

## Поля:

* + *Ants*: список мурах, що використовуються в алгоритмі.
  + *Pheromones*: список матриць феромонів, що відповідають кожному виду транспортного засобу.
  + *Graph*: граф, який представляє мережу міст і відстані між ними.
  + *MaxIterations*: Максимальна кількість ітерацій алгоритму.
  + *Alpha*: параметр альфа, що визначає вплив феромонів на вибір наступного міста.
  + *Beta*: параметр бета, що визначає вплив відстані на вибір наступного міста.
  + *AntCount*: кількість мурах в колонії.
  + *EvrRate*: швидкість випаровування феромонів (евапорація).
  + *MinLength*: мінімальна довжина маршруту.
  + *Random*: випадковий генератор чисел. Поля специфічні для VRP:
  + *NumberOfVehicles*: кількість доступних транспортних засобів.
  + *Customers*: список клієнтів (міст), які необхідно відвідати.
  + *Vehicles*: список транспортних засобів з їхньою місткістю. Поля звітності:
  + *IsReporting*: прапорець, який показує, чи включена звітність.
  + *ReportInterval*: інтервал, з яким відбувається звіт.

## Методи:

* + *UpdateReportOptions(bool isReporting, int reportIneterval)*: оновлює налаштування звіту.
  + *TrySolve(out Result result)*: спробувати вирішити задачу VRP і отримати результат. повертає true, якщо задачу вдалося вирішити, та false в іншому випадку.
  + *Solve()*: вирішити задачу VRP і повернути результат.
  + *UpdatePheromone()*: оновлює рівні феромонів відповідно до знайдених маршрутів мурах.
  + *UpdateAnts()*: оновлює стан мурах перед початком нової ітерації.
  + *InitAnts()*: ініціалізує список мурах з випадковими стартовими містами.
  + *InitPheromones()*: ініціалізує початкові рівні феромонів усіх маршрутів.

Клас ***AntColonyVRP*** реалізує мурашиний алгоритм для розв'язання задачі маршрутизації транспорту. Він містить логіку пошуку оптимального маршруту, оновлення феромонів та управління мурахами. Поля та методи класу допомагають ініціалізувати алгоритм, встановити параметри та отримати результати розв'язку VRP.

1. Клас ***AntVRP*** представляє мурашника для розв'язання проблеми маршрутизації транспорту. Даний клас реалізує інтерфейс IAnt.

## Основні властивості класу:

* + *NumberOfAnts*: кількість мурашників;
  + *CommonStartPoint*: спільна стартова точка для всіх мурашників;
  + *Positions*: список поточних позицій мурашників;
  + *Pheromone*: феромон;
  + *PathCosts*: список вартостей шляхів мурашників;
  + *Pathes*: список шляхів мурашників;
  + *PossibleWays*: список можливих шляхів;
  + *Vehicles*: список транспортних засобів;
  + *Customers*: список клієнтів.

Конструктор класу AntVRP приймає значення феромону, кількість мурашників, список транспортних засобів та список клієнтів. У конструкторі ініціалізуються властивості об'єкта і викликаються методи InitializeLists() і ResetContents().

## Методи класу:

* + *ResetContents():* скидає вміст кожного транспортного засобу на його початкову ємність та встановлює прапорець isSatisfied для кожного клієнта на значення false.
  + *InitializeLists():* ініціалізує списки PathCosts, Pathes і Positions для кожного мурашника.
  + *InitWays(int size):* ініціалізує список можливих шляхів, виключаючи спільну стартову точку.
  + *NeedToRefill(int n):* перевіряє, чи потрібно заправити транспортний засіб n наступного мурашника.
  + *ReturnVehicles(Graph graph):* повертає транспортні засоби до спільної стартової точки шляхом додавання її до кінця шляху кожного мурашника.
  + *CalculateSummary(Graph graph, List<double[,]> pheromones, int beta, int alpha, int n):* обчислює суму значень функції імовірності для кожного можливого шляху для мурашника n.
  + *CalculateChance(Graph graph, List<double[,]> pheromones, int beta, int alpha, int n, int way, double summary):* обчислює ймовірність обрання конкретного шляху для мурашника n.
  + *RefillVehicle(int n):* перезаповнює транспортний засіб n його початковою ємністю.
  + *IsLastPossibleWay():* перевіряє, чи є останній можливий шлях для мурашника.
  + *IsWayPossible(int n, int way):* перевіряє, чи є шлях way доступним для мурашника n.
  + *Move(Graph graph, List<double[,]> pheromones, int beta, int alpha):* виконує переміщення мурашників від клієнта до клієнта з урахуванням значень феромонів та функції імовірності.

Цей клас використовується для моделювання розв'язання проблеми маршрутизації транспорту з обмеженнями за допомогою алгоритму мурашиного колонії у варіанті VRP.

1. Клас ***ConfigVRP***, який успадковується від абстрактного класу Config і використовується для налаштування алгоритму Ant Colony Optimization для задачі VRP.

## Основні властивості класу ConfigVRP:

* + *MaxIterations*: максимальна кількість ітерацій алгоритму
  + *Alpha*: значення параметра Alpha
  + *Beta*: значення параметра Beta
  + *Rho*: значення параметра Rho
  + *AntCount*: кількість мурах

## Клас має два конструктори:

Безпараметричний конструктор ConfigVRP(), який встановлює початкові значення властивостей MaxIterations, Alpha, Beta, Rho та AntCount.

Параметричний конструктор ConfigVRP(int MaxIterations, int Alpha, int Beta, double Rho, int AntCount), який приймає значення для всіх властивостей і встановлює їх значення відповідно до переданих аргументів.

Клас також має перевизначений метод ToString(), який повертає рядок, який містить значення властивостей об'єкту класу ConfigVRP.

Отже, клас ConfigVRP використовується для налаштування параметрів алгоритму Ant Colony Optimization для задачі VRP. Його властивості визначають значення параметрів Alpha, Beta, Rho, кількість мурах і

максимальну кількість ітерацій, а метод ToString() надає рядкове представлення цих значень.

1. Клас ***ResultVRP***, який успадковується від класу Result і реалізує додаткові властивості і методи для розв'язання проблеми маршрутизації транспорту.

## Основні властивості класу ResultVRP:

* + *Time*: час, який знадобився для знаходження розв'язку
  + *PathCost*: вартість знайденого маршруту
  + *PathCosts*: список вартостей маршрутів
  + *BestPathes*: список найкращих маршрутів (представлений списками цілочисельних значень)
  + *BestIteration*: номер ітерації, на якій був знайдений найкращий маршрут
  + *CurrentIteration*: поточний номер ітерації
  + *StartTime*: час початку розрахунку
  + *OptimalTime*: оптимальний час розрахунку

## Методи класу ResultVRP:

* + *SetTime()*: встановлює значення властивості Time, вираховуючи різницю між поточним часом і StartTime.
  + *ToString()*: перевизначений метод, який повертає рядок зі згенерованим відображенням об'єкта ResultVRP. У даному методі формується рядок, який містить інформацію про найкращі маршрути зі списку BestPathes.

Цей код використовується для представлення результатів розв'язання проблеми маршрутизації транспорту, де реалізовано відстеження часу виконання, обчислення вартості маршрутів та збереження найкращих маршрутів.

## Опис деяких допоміжних класів

1. Клас **DirectoryManager**, є статичним і містить різні методи для роботи з директоріями і шляхами файлів.

## Константи:

* + *PATH\_DEFAULT*: містить значення, яке використовується як типовий шлях до вхідних даних.
  + *SAVE\_PATH\_DEFAULT:* містить значення , яке представляє типовий шлях для збереження даних.

## Методи:

* + *GetRelativeDataDirectory()*: повертає абсолютний шлях до директорії з вхідними даними. Він спочатку перевіряє наявність значення "LocalDataFolderPath" у конфігураційних налаштуваннях (AppSettings). Якщо це значення відсутнє, використовується типовий шлях "PATH\_DEFAULT". Потім отримується поточна робоча директорія (currentDirectory), і за допомогою методу "Path.Combine" об'єднується поточна директорія з відносним шляхом (relativePath), що призводить до отримання повного шляху до директорії з вхідними даними. Повний шлях повертається як результуюче значення методу.
  + *GetFilePathFromDataDirectory(string fileName):* повертає абсолютний шлях до файлу, знаходячись в директорії з вхідними даними. Використовується метод "GetRelativeDataDirectory()" для отримання шляху до директорії з вхідними даними, після чого до цього шляху додається ім'я файлу (fileName) з використанням оператора конкатенації. Отриманий повний шлях до файлу повертається як результат методу.
  + *GetSavePath()*: повертає типовий шлях для збереження даних, який заданий константою "SAVE\_PATH\_DEFAULT". Це просто

повертається значення константи "SAVE\_PATH\_DEFAULT" як результуюче значення методу.

Клас **DirectoryManager** надає деякі зручні методи для отримання шляхів до директорій і файлів, які можуть бути використані в іншій частині коду для роботи з файловою системою.

1. Клас **Euc2DParserTSP** реалізує інтерфейс IParser. Він використовується для розбору файлів, що містять дані задачі TSP (Travelling Salesman Problem) у форматі EUC\_2D.

## Методи:

* + *ParseNode(string nodeStr):* цей метод приймає рядок nodeStr, який представляє дані про вузол (точку) задачі TSP у форматі EUC\_2D. Метод розбирає рядок і створює об'єкт класу Node, заповнюючи його відповідними значеннями, що містяться у рядку. Значення для ідентифікатора (Id) витягуються з першого елемента рядка, а значення для координат X і Y витягуються з другого і третього елементів рядка відповідно. Після цього об'єкт Node повертається з методу.
  + *ParseFromFile(string filePath):* цей метод приймає шлях до файлу, який містить дані задачі TSP у форматі EUC\_2D. Метод зчитує файл по заданому шляху та розбирає його для створення об'єкта ProblemModelTSP, який представляє модель задачі TSP. Метод відкриває файл для читання за допомогою StreamReader. Потім він читає перші рядки файлу, які містять загальну інформацію про задачу TSP (назва, коментар, тип, розмірність тощо) та встановлює відповідні значення в об'єкт model. Далі метод читає рядки, що містять дані про вузли (точки) задачі TSP, та використовує метод ParseNode для розбору цих рядків і створення об'єктів Node. Отримані об'єкти Node додаються до списку Destinations в об'єкті model. Читання файлу завершується, якщо зустрічено рядок "EOF" або якщо прочитаний рядок порожній. У разі виникнення помилки при читанні

файлу, виводиться повідомлення про помилку. На завершення метод повертає об'єкт model, що містить розібрані дані задачі TSP.

1. Клас **Euc2DParserVRP** реалізує інтерфейс IParser і виконує обробку та розбір даних з файлу для моделі VRP (Vehicle Routing Problem).

## Методи:

* + *AdustString(string lineStr)*: цей метод приймає рядок тексту і виконує його обробку шляхом заміни послідовностей пробілів на один пробіл, заміни крапок на коми і видалення зайвих пробілів з початку і кінця рядка. Результат обробки повертається як рядок.
  + *BuildMatrix(List<string> rows)*: цей метод приймає список рядків і будує двовимірний масив чисел типу double. Він розбиває кожен рядок на окремі числа, розділені пробілами, і заповнює ними елементи матриці. Повертається отримана матриця.
  + *ReplaceDepot(double[,] matrix, int depotId, int dimention)*: цей метод приймає матрицю, індекс депо (зберігається у змінній depotId) та розмірність матриці (зберігається у змінній dimention). Він замінює рядки та стовпці, що відповідають депо, з першим рядком та першим стовпцем відповідно. Результат повертається як модифікована матриця.
  + *GetCustomer(string line)*: цей метод приймає рядок тексту, який представляє інформацію про клієнта, і повертає об'єкт класу Customer, заповнений даними про клієнта. Рядок спочатку обробляється шляхом заміни послідовностей пробілів на один пробіл, заміни крапок на коми і видалення зайвих пробілів з початку і кінця рядка. Результат обробки використовується для заповнення властивостей об'єкта Customer, таких як Id і Demand, і повертається об'єкт Customer.
  + *ParseFromFile(string filePath)*: цей метод приймає шлях до файлу, який містить дані для моделі VRP, і повертає об'єкт типу

ProblemModel, заповнений відповідними даними. Метод використовує об'єкт StreamReader для читання файлу і обробки кожного рядка даних. Дані зчитуються відповідно до визначених форматів, ініціалізуються відповідні властивості об'єкта ProblemModelVRP, такі як Name, Comment, Type, Dimention, EdgeWeightType, EdgeWeightFormat, DataType, CustomerDemands, Matrix і VehicleCapacities. Результат повертається як об'єкт ProblemModelVRP.

У цьому класі також є приватні поля, такі як rows, capacity, vehicles і depotId, які використовуються для зберігання проміжних даних під час обробки файлу.

1. Клас **DistanceMeasuringHelper** є статичним, що означає, що його методи можна використовувати без створення екземпляра класу.

## Методи:

* + *MeasureIntDistances:* обчислює відстані між точками, використовуючи цілі числа. Він створює новий двовимірний масив result розміром points.Count на points.Count, де points - це список точок з об'єкта problemModel. Для кожної пари точок (i, j) метод обчислює відстань між ними за допомогою формули відстані між двома точками в евклідовому просторі. Отримане значення округлюється до найближчого цілого числа та зберігається у відповідному елементі result.
  + *MeasureDoubleDistances:* виконує аналогічні дії, але результат зберігає у двовимірному масиві типу double, зберігаючи точні значення відстаней.

Обидва методи повертають обчислені відстані у вигляді двовимірного масиву, де кожний елемент [i, j] представляє відстань між точками i і j зі списку points. У коментарях є закоментовані цикли for з викликом Console.WriteLine,

які можна розкоментувати, якщо потрібно вивести відстані на консоль для перевірки.

## Графічний інтерфейс

У процесі роботи було розроблено графічний інтерфейс на платформі WinForms (рис 3.4). У початковий план не входила розробка інтерфейсу, але можливість візуалізації отриманих результатів дало можливість виявити і виправити недоліки створеного алгоритму, спростити роботу із файловою системою для зручного збереження результатів і покращити розуміння приципів роботи алгоритму.

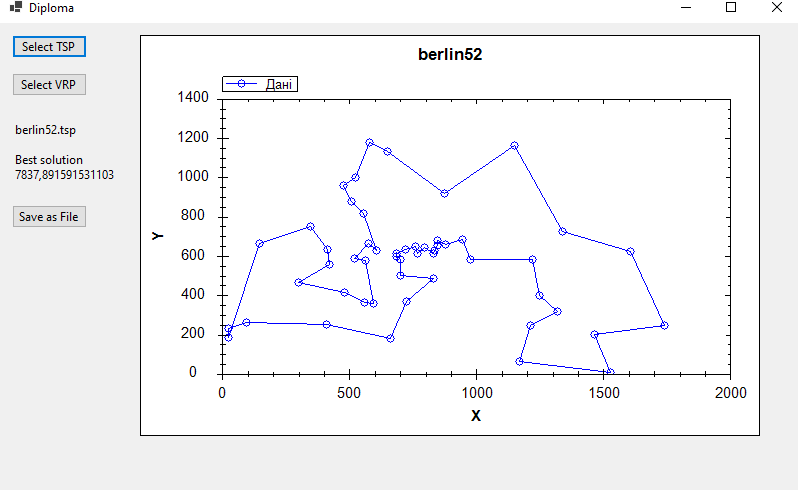
Програма будує графи тільки для задач комівояжера, оскільки асиметричні задачі маршрутизації транспортних засобів із ваговим обмеженням не передбачають координат точок у вхідних даних. Крім того основною метою створення інтерфейсу була можливість покращити алгоритм на початкових етапах його розробки.

Рисунок 3.4 – Графічний інтерфейс

До функціоналу входить можливість обирати набори даних безпосередньо із файлової системи, візуалізувати результат та зберігати файли у текстових файлах для їх застосування у реальних задачах. Програма приймає файли в форматах .tsp та .dat для задач комівояжера і асиметричних задач маршрутизації транспортних засобів із ваговим обмеженням відповідно. За необхідності додання нового формату даних на вхід, це можливо зробити, не порушивши функціонал програми, оскільки це було передбачено на ранніх етапах розробки.

# АНАЛІЗ РЕЗУЛЬТАТІВ

## Параметри

Як було зазначено вище одним із недоліків ACO є необхідність підбирати параметри під кожну окрему задачу. У програмі наявна можливість змінювати всі параметри алгоритму за допомогою класу ConfigTSP. За замовчуванням параметри беруться із файлу конфігурації, але також наявна можливість підбору параметрів в залежності від кількості точок.

Основні відомості про параметри:

* Кількість мурах (***Ant Count***). Це кількість мурах, які будуть рухатися по містах. Загальна рекомендація полягає в тому, щоб мати кількість мурах, що дорівнює приблизно 10-20% від загальної кількості міст.
* Вага феромону (***Alpha***). Це параметр, який впливає на значущість феромону при виборі маршрутів мурахами. Вища вага феромону призводить до більшої впливовості феромону, що дозволяє мурахам швидше знаходити короткі маршрути. Рекомендоване значення для цього параметра може бути в діапазоні 1-5.
* Вага привабливості (***Beta***). Цей параметр впливає на значущість привабливості між містами при виборі маршрутів. Чим вища вага привабливості, тим більше мурахи будуть керуватись привабливістю міст при виборі наступного міста. Рекомендоване значення для цього параметра також може бути в діапазоні 1-5.
* Швидкість випаровування феромону (***Rho***). Цей параметр визначає, яка частка феромону випаровується після кожної ітерації. Вища швидкість випаровування феромону дозволяє швидше забувати менш ефективні маршрути і зосереджуватись на кращих маршрутах. Рекомендоване значення може бути в діапазоні 0.1-0.5.

## Результати роботи алгоритму для задачі комівояжера

Для оцінки якості роботи алгоритму використано відомі тестові приклади з бібліотеки TSPLIB [5] та VLSI [7].

Одним із найкращих розв’язань задачі berlin52 є відстань у 7544,3659 [3]. Отже, наближення до цього значення можна вважати за успішність роботи алгоритму. Для виключення випадковості були проведені по 50 спроб знайти оптимальний шлях для кожної групи параметрів. Далі подані вхідні параметри і середні результати роботи програми. Тестування проводилося на ПК із операційною системою Windows 10 x64, процесором Intel(R) Core(TM) i5-4570 CPU 3.20GHz та 16гб оперативної пам’яті.

Iteration – ітерація, на якій був досягнутий найкращий результат. Optimal time – час досягнення цієї ітерації. Час наведено у секундах.

Таблиця 4.1 — Результати роботи на датасеті berlin52

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Alpha*** | ***Beta*** | ***Rho*** | ***Ant Count*** | ***Max Iterations*** | *Iteration* | *Length* | *Optimal time* | *Total time* |
| 5 | 1 | 0,9 | 40 | 1000 | *10.22* | *9889.97* | *0.06* | *5.89* |
| 1 | 5 | 0,1 | 40 | 1000 | *508.88* | *7838.86* | *3.11* | *6.06* |
| 5 | 5 | 0,2 | 40 | 1000 | *3.46* | *8390.76* | *0.04* | *8.69* |
| 1 | 1 | 0,2 | 40 | 1000 | *559.92* | *8193.52* | *1.39* | *2.46* |
| 1 | 2 | 0,5 | 30 | 1000 | *511.26* | *7802.09* | *2.3* | *4.48* |
| 1 | 2 | 0,5 | 30 | 300 | *177.9* | *7886.58* | *0.82* | *1.36* |
| 1 | 2 | 0,5 | 30 | 1000 | *493.2* | *7821.3* | *2.24* | *4.5* |
| 1 | 2 | 0,5 | 30 | 1000 | *479.6* | *7809.66* | *2.2* | *4.58* |

На основі перевірених даних, можна зробити висновок, що оптимальні параметри для цієї конкретної задачі будуть із 6 і 8 прикладів. Окремо варто відзначити наступні два розв’язки, які дуже наближаються до відомого оптимального розв’язку.

*Iteration: 110; Length: 7591.01; Optimal seconds: 0.51; Total seconds: 4.58*

*Iteration: 277; Length: 7550.19; Optimal seconds: 1.26; Total seconds: 4.52*

Інший важливий висновок – тисяча ітерацій є надто великою. Із часом шанс зміни маршруту зменшується, оскільки ферамони концентруються на відомому найкращому шляху. Як результат ймовірність його подальшої оптимізації мінімізується. Це можна вирішити або зменшенням кількості ітерацій для зменшення навантаження на обчислювальну техніку, або застосуванням модифікацій до методу. У цьому випадку як варіант можна збільшити фактор випадковості після приблизно 500 ітерацій. Таким чином зменшиться застигання шляху ближче до останніх ітерацій. Але в контексті цього експерименту зібраної інформації достатньо.

Задача xqf131 [7]:

Відомий оптимальний розв’язок – 564 (рис 4.4), знайдений розв’язок 632 (рис 4.5)

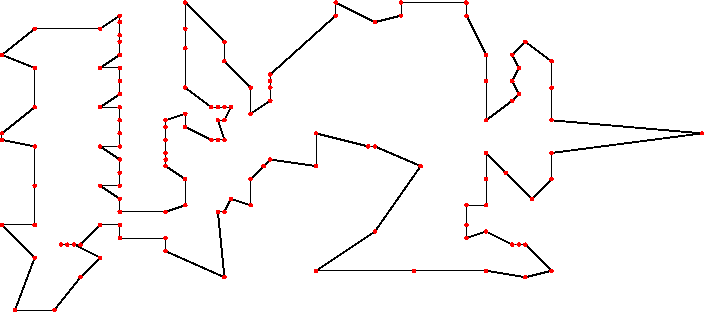


Рисунок 4.4 – Оптимальний розв’язок задачі xqf131

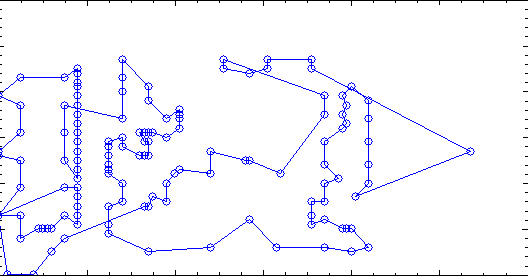


Рисунок 4.5 – Знайдений розв’язок задачі xqf131

Задача bcl380 [7]:

Відомий оптимальний розв’язок – 1621 (рис 4.6), знайдений розв’язок 1866 (рис 4.7)

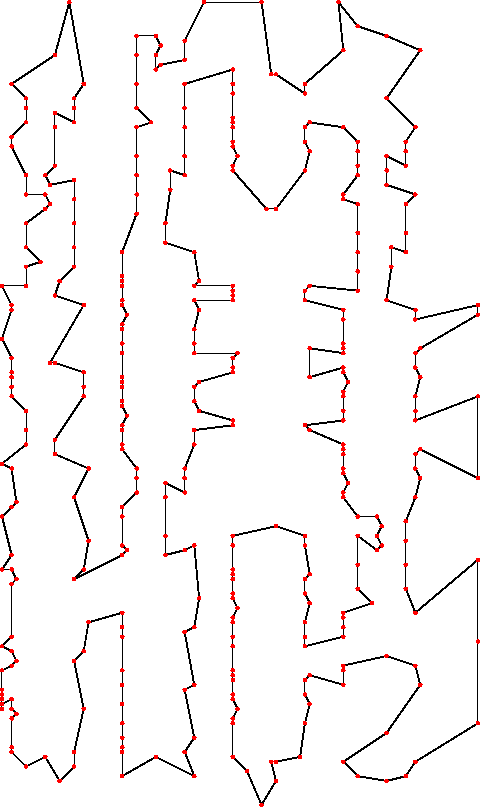


Рисунок 4.6 – Оптимальний відомий розв’язок задачі bcl380

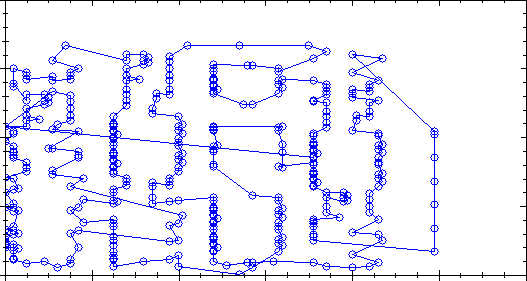


Рисунок 4.7 – Знайдений розв’язок задачі bcl380 Спираючись на ці результати можна вважати роботу алгоритму на

задачах комівояжера задовільною. При подальшому більш ретельному вибору параметрів для кожного набору даних можна досягти результатів дуже близьких до оптимальних (як було зазначено вище із berlin52).

## Результати роботи алгоритму для задачі маршрутизації транспортних засобів

Оскільки для асиметричної версії цієї задачі дані надаються тільки у матриці відстаней, було ухвалено рішення не робити візуалізацію маршрутів. Для тестування взяті набори даних із FTV1994 [8].

1. Задача A065-03f:

Відомий оптимальний розв’язок – 1974 [9], знайдений розв’язок 2583. Тут проявляється недолік методу мурашиної колонії, пов’язаний із фокусуванням на феромонових слідах, що у задачі із кількома транспортними засобами заважає побудові нових маршрутів, якщо інша колонія вже залишила значний слід до якоїсь точки. Так замість конкуренції між колоніями за найкращі шляхи, колонії формують свій список точок для відвідування і майже не обмінюються ними. Це призводить до швидкого згасання

покращення результатів роботи із кожною ітерацією. Проте результат не є незадовільним. Для порівняння середній випадковий шлях буде становити приблизно 9000 умовних одиниць. Таким чином можна зробити висновок, що алгоритм працює задовільно.

1. Задача A034-02f:

Відомий оптимальний розв’язок – 1406 [9], знайдений розв’язок 1648. Тут середня довжина випадкового маршруту складатиме приблизно 4239. Ситуація аналогічна із попередньою задачею. Збільшення кількості ітерацій не дає значного позитивного ефекту, а от збільшення фактору випадковості, особливо на пізніх етапах алгоритму, дає помітний результат при побудові маршруту.

1. Задача A048-03f:

Відомий оптимальний розв’язок – 1891 [9], знайдений розв’язок 2443. Тут середня довжина випадкового маршруту складатиме приблизно 6692.

Отже, результати роботи алгоритму є в цілому задовільними. Алгоритм проявив себе менш ефективно у розв’язанні задачі асиметричної проблеми маршрутизації транспортних засобів із обмеженою ємністю, ніж у задачі комівояжера. Це пов’язано із досить швидким визначенням шляхів у кожної колонії, яка репрезентує кожен окремий транспортний засіб. Проте у порівнянні із випадковими довжинами шляхів алгоритм видає результат у 2-4 рази кращий залежно від конкретного набору даних. Однозначною перевагою є швидкість роботи алгоритму і можливість знаходити близькі до оптимальних розв’язки за малу кількість ітерацій.

# ВИСНОВКИ

В результаті роботи вивчені різні підходи до побудови маршрутів для транспортування товарів від складу до замовників. Зокрема, було досліджено сучасні метаевристичні методи, серед яких особливу увагу приділено методу мурашиної колонії.

На основі отриманих знань розроблено програмне забезпечення для диспетчеризації транспортування товарів, використовуючи метод мурашиної колонії. Створено простий графічний інтерфейс, що показує побудовані маршрути у задачах комівояжера. Розроблено функціонал експортування результатів у файли для їх подальшого впровадження в реальних задачах оптимізації маршрутів.

Проведення експериментів на існуючих наборах даних дозволило оцінити ефективність та придатність розробленого програмного забезпечення. Аналіз отриманих результатів показав, що використання методу мурашиної колонії дозволяє будувати прийнятні маршрути, але досягти найкращих відомих результатів не вдалось.

Виявлені можливості подальшого розвитку та вдосконалення програмного забезпечення, такі як впровадження додаткових метаевристичних методів або розширення функціональності програми. Ці можливості відкривають перспективи для подальшого вдосконалення системи диспетчеризації транспортування товарів та підвищення її ефективності в майбутньому.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Blum, C., Roli A. *Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys (CSUR). 2003.* 308 p.
2. Л. Ф. Гуляницький, О. Ю. Мулеса. Прикладні методи комбінаторної оптимізації. 2016. 142 с.
3. Otman A., Tajani C., Abouchabaka J. Hybridizing *PSM and RSM Operator for Solving NP-Complete Problems: Application to Travelling Salesman Problem.* 2012. 378 p.
4. Afsana K., Afrida Anzum Aesha, Juthi S. *New Algorithmic Approach to Finding Minimum Spanning Tree.* 2018. 6 p.
5. Symmetric traveling salesman problem data. URL: [http://comopt.ifi.uni-](http://comopt.ifi.uni-/) heidelberg.de/software/TSPLIB95/tsp/ (дата звернення: 01.06.2023)
6. Optimal solutions for symmetric TSPs. URL: [http://comopt.ifi.uni-](http://comopt.ifi.uni-/) heidelberg.de/software/TSPLIB95/STSP.html (дата звернення: 01.06.2023)
7. Optimal solutions for symmetric TSPs VLSI Problem collection. URL: https://[www.math.uwaterloo.ca/tsp/vlsi/index.html](http://www.math.uwaterloo.ca/tsp/vlsi/index.html) (дата звернення: 01.06.2023)
8. FTV1994 ACVRP problem data. <http://www.vrp-rep.org/datasets/item/2017-> 0002.html (дата звернення: 01.06.2023)
9. Optimal solutions for ACVRP FTV1994 problem data. https://[www.hindawi.com/journals/mpe/2022/2435002/tab13/](http://www.hindawi.com/journals/mpe/2022/2435002/tab13/) (дата звернення: 01.06.2023)

## Додаток А

**Приклад вихідних даних для задачі комівояжера**

Total number of iterations is 1000 Best way was achieved at iteration 540

Algorithm ended in 133,1550514 seconds Best path cost is 3751,714088313742

Best path is: 198-->144-->145-->142-->143-->199-->200-->201-->202-->203--

>204-->205-->206-->211-->212-->215-->214-->213-->218-->217-->216--

>219-->220-->221-->222-->223-->224-->225-->226-->227-->228-->229--

>250-->249-->246-->243-->242-->239-->240-->241-->1-->279-->2-->278--

>277-->247-->248-->255-->254-->251-->208-->207-->252-->253-->256--

>257-->258-->259-->260-->261-->262-->263-->264-->265-->266-->267--

>268-->269-->133-->134-->135-->136-->137-->138-->139-->140-->141--

>146-->147-->148-->149-->178-->179-->180-->181-->182-->183-->184--

>185-->189-->190-->191-->193-->194-->195-->196-->197-->192-->188--

>187-->186-->170-->0-->4-->5-->6-->7-->9-->10-->11-->12-->13-->14--

>15-->270-->271-->272-->273-->274-->275-->276-->3-->8-->126-->125--

>28-->27-->26-->25-->21-->24-->22-->23-->16-->17-->18-->131-->130--

>129-->20-->127-->128-->153-->154-->152-->155-->151-->150-->177--

>176-->175-->173-->172-->105-->104-->106-->169-->168-->100-->99--

>98-->97-->96-->95-->94-->93-->92-->91-->90-->89-->79-->78-->77--

>76-->74-->73-->72-->71-->70-->69-->66-->65-->64-->84-->85-->115--

>114-->113-->110-->107-->103-->102-->101-->171-->167-->166-->165--

>164-->163-->162-->161-->160-->174-->159-->158-->157-->156-->118--

>119-->120-->121-->122-->123-->124-->29-->30-->31-->32-->33-->34--

>35-->36-->37-->38-->39-->40-->41-->42-->59-->60-->117-->116-->58--

>43-->44-->45-->46-->53-->52-->47-->48-->49-->50-->51-->54-->55--

>56-->57-->67-->68-->63-->62-->61-->83-->86-->112-->111-->87-->82--

>81-->80-->88-->108-->109-->75-->132-->19-->210-->209-->232-->231--

>236-->235-->234-->233-->230-->245-->244-->237-->238-->0-->198

## Додаток Б

**Приклад вихідних даних для задачі маршрутизації транспортних засобів**

Total path costs - 2886 :

1067 - 0-->38-->3-->14-->47-->36-->41-->15-->13-->46-->45-->11-->42-->43-

->44-->12-->35-->20-->19-->48-->21-->61-->0

574 - 0-->2-->4-->39-->5-->7-->8-->40-->9-->10-->60-->64-->18-->17-->16--

>1-->37-->50-->55-->0

1245 - 0-->34-->56-->29-->25-->63-->22-->59-->23-->49-->24-->51-->30--

>53-->31-->62-->52-->32-->54-->28-->27-->26-->0-->57-->58-->6-->33-->0