МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ АЕРОНАВІГАЦІЇ, ЕЛЕКТРОНІКИ ТА ТЕЛЕКОМУНІКАЦІЙ

КАФЕДРА АЕРОКОСМІЧНИХ СИСТЕМ УПРАВЛІННЯ

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач випускної кафедри

_____ О.М. Тачиніна

«___» _____ 2020 р.

ДИПЛОМНА РОБОТА

(Пояснювальна записка)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

ЗА СПЕЦІАЛЬНІСТЮ 151 «АВТОМАТИЗАЦІЯ ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ»

Тема: «Дослідження і розробка методів надійної навігації безпілотних літальних апаратів»

Виконавець: студент групи ФАЕТ – 601, Калмиков Вадим Віталійович

Керівник: доктор техн. наук, професор, Гаєв Євген Олександрович

Консультант розділу «Охорона праці»: _____ (підпис) _____ (П.І.Б)

Консультант розділу «Охорона навколишнього середовища»: _____ (підпис) _____ (П.І.Б)

Нормоконтролер: _____ (підпис) _____ (П.І.Б)

КИЇВ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ АЕРОНАВІГАЦІЇ, ЕЛЕКТРОНІКИ ТА ТЕЛЕКОМУНІКАЦІЙ

Кафедра аерокосмічних систем управління

Спеціальність: 151«Автоматизація та комп'ютерно-інтегровані технології»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____О.М. Тачиніна

«_____»_____2020 р.

ЗАВДАННЯ
на виконання дипломної роботи

Калмикова Вадима Віталійовича

1. Тема дипломної роботи: "Дослідження і розробка методів надійної навігації безпілотних літальних апаратів" затверджена наказом ректора від «08» жовтня 2020 р. 1944/ст.
2. Термін виконання роботи: з 05.10.2020 по 27.12.2020.
3. Вихідні дані до роботи: Використати сучасні комп'ютерні засоби для створення алгоритму маршрутизації у системі безпілотних літальних апаратів та симулювати політ оптимальної кількості дронів за визначеним маршрутом, з метою доставки вантажу. За основу взяти NP-повну Capacitated Vehicle Routing Problem (CVRP) "задачу вагової маршрутизації транспортних засобів", імплементовану під використання її в авіації. Мова програмування – Python. Кінцева мета – симуляція роботи системи в программі Misson Planner, керована алгоритмом рекурентної нейронної мережі на основі привертання уваги (Recurrent Attention Neural Network) зі вчителем (тренована попередніми результатами інших штучних алгоритмів). Провести оцінку отриманої системи, що задовільняє критерію якості, достатньому для подальшого використання на прикладі реального безпілотного апарату.
4. Зміст пояснювальної записки:

Розділ 1: Постановка задачі знаходження оптимального маршруту у системі БПЛА.

Розділ 2: Відомі методи вирішення задачі

Розділ 3. Розробка власного алгоритму та його програмування.

Розділ 4. Симуляція результатів роботи алгоритму для системи БПЛА.

Розділ 5: Охорона навколишнього середовища

Розділ 6. Охорона праці та безпека життєдіяльності

5. Перелік обов'язкового графічного (ілюстрованого) матеріалу: ілюстрації у вигляді GPS карти України, з попередньо відібраної місцевості, авіаційними сполученнями, приклади роботи створених python-програм та результатів їх роботи, таблиці випробування алгоритмів.

6. Календарний план-графік

№ пор	Завдання	Термін виконання	Примітка
1	Огляд літературних джерел.	03.09.20 – 30.09.20	
2	Розробка PC-програм	31.09.20 – 20.10.20	
3	Написання першого та другого розділів диплому.	20.10.20 - 25.10.20	
4	Написання третього розділу.	25.10.20 - 31.10.20	
5	Написання четвертого розділу.	31.10.20 – 10.11.20	
6	Написання розділу з охорони навколишнього середовища	10.11.20 - 15.11.20	
7	Написання розділу з охорони праці	15.11.20 - 20.11.20	
8	Аналіз отриманих результатів.	20.11.20 – 10.12.20	
9	Фінальні виправлення і узгодження. Підготовка презентації і виступу, отримання рецензій.	30.11.20 – 22.12.20	

7. Дата видачі завдання: 3 вересня 2020 р.

Керівник дипломної роботи _____ Є. Гаєв
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ В.Калмиков
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Магістерська дисертація за освітньо-професійною програмою автоматизація та комп'ютерно-інтегровані технології підготовки рівня «магістр» на тему "Дослідження і розробка методів надійної навігації безпілотних літальних апаратів": 130 сторінки, 45 рисунків, 6 таблиць, 72 посилання, 62 авторські програми.

В роботі розроблено алгоритм оптимізації побудови маршрутів для систем БПЛА з урахуванням вантажу, що може нести дрон, та висотних коридорів. Вирішено задачу підвищення ефективності навігації, а саме маршрутизації, системи дронів за допомогою найсучаснішіх методів оптимізації, що включають в себе використання рекурентних нейронних мереж на основі привертання уваги (Recurrent Attention Neural Network) зі вчителем та генетичних алгоритмів, для їх навчання.

Отриманий алгоритм оптимізації був використаний для симуляції польоту системи БПЛА. Результати дослідження були підтверджені за допомогою моделювання в графічному середовищі Misson Planner.

Результати магістерської атестаційної роботи попередньо були частково продемонстровані у публікаціях [1] і конференціях "Політ". За основу взята попередня дипломна робота **"**Minimization of aviation routes by artificial intelligence methods" [2 - 4].

Ключові слова: безпілотний літальний апарат, система навігації, проблема подорожуючого торговця, рекурентна нейронна мережа, Attention Neural Network, імітація відпалу, якість системи, Python.

ABSTRACT

Master's diploma thesis on the educational-professional program automation and computer-integrated technologies of the level "master" on the subject "Research and development of robust navigation methods for Unmanned Aerial Vehicles (UAV)": 130 pages, 45 graphs, 6 tables , 72 references, 62 author's programs.

The paper develops an algorithm for optimizing the construction of routes for UAV systems taking into account the load that can carry a drone and high-altitude corridors. The problem of the navigation efficiency improving, namely routing, of the drone systems with the help of modern optimization methods, including the use of recurrent neural networks based on attention (Recurrent Attention Neural Network) with the teacher and genetic algorithms for their training.

The obtained optimization algorithm was used to simulate the flight of the UAV system. The results of the study were confirmed by experiments, calculations and modeling in the graphical environment of Misson Planner.

The results of the master's attestation work were previously partially demonstrated in publications [1] and conferences "Flight". The previous diploma work “Minimization of aviation routes by artificial intelligence methods” was taken as a basis [2 - 4].

Keywords: unmanned aerial vehicle, navigation system, traveling merchant problem, Recurrent Neural Network, Attention Neural Network, Simulated annealing, system quality, Python.

Скорочення та умовності

У роботі можуть зустрічатися наступні абревіатури та скорочення:

ACO – Ants Colony Optimization (Оптимізація мурашиної колонії)

CVRP – Capacitated Vehicle Routing Problem (Ємнісна проблема маршрутизації транспортних засобів)

GPS – Global Positioning System (Глобальна система навігації)

HNN – Hopfield neural network (Нейронна мережа Хопфілда)

NN – Neural network (Нейронна мережа)

NP-повна – це клас складності, що представляє собою набір всіх рішенн, для яких не існує доказів, які можуть бути перевірені за поліноміальний час.

RANN – Recurrent Attention Neural Network, Рекурентна нейронна мережа методом “привертання уваги”.

RTL — Return to Launch mode

SA – Simulated Annealing (Імітація відпалу)

TSP – Travelling Salesman Problem (Проблема подорожуючого торговця)

VRP – Vehicle Routing Problem (Проблема маршрутизації транспортних засобів)

БПЛА – безпілотний літальний апарат

САУ – система автоматичного управління.

ЗМІСТ

ВСТУП

В наші дні розвиток різного роду мобільних роботів набув стрімкого темпу, різко популярним стало використання безпілотних літальних апаратів (БПЛА) в різноманітних сферах людської діяльності. Насамперед це пов’язано з перевагами, які притаманні дронам. Завдяки компактним розмірам, надійній конструкції, маневреності, простоті в управлінні, за малої ваги при значній масі корисного навантаження, БПЛА здатні виконувати найрізноманітніші задачі.

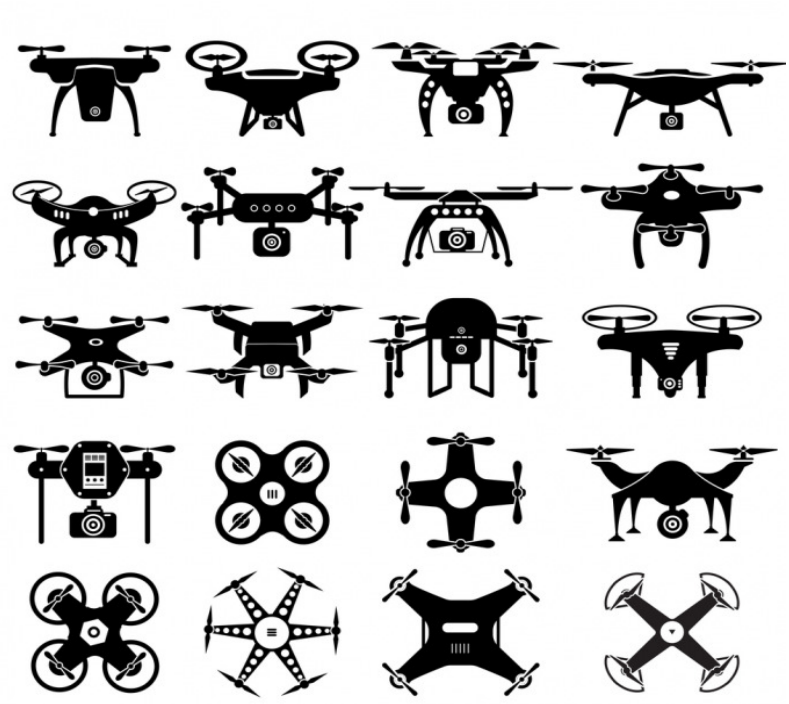


Рис. 1. Різноманітні види БПЛА [5]

Вони успішно використовуються в таких сферах як: аерофотозйомка та картографування, оперативне прогнозування та оцінка негативного впливу за надзвичайних ситуацій, різного роду моніторинг, в тому числі і об’єктів промисловості та природничих комплексів, доставки різного роду товарів, у розважальних цілях і так далі.

Однак можливості використання БПЛА обмежені тим, що на сьогодні керування польотом здійснюється в напівавтоматичному режимі за командами оператора, або в дистанційному режимі, використовуючи пульт керування.

Існує багато варіантів рішень в плані навігації БПЛА, але в кожного рішення є як свої переваги так і недоліки. Задля знаходження оптимальних маршрутів польоту БПЛА дуже широко використовуються алгоритми штучного інтелекту і в їх числі, так звані, нейронні мережі. Варіантів нейронних мереж існує безліч і всі вони так чи інакше підходять тільки для певних варіацій задач (задачі класифікації, кероване навчання, зниження розмірності, виявлення аномалій та діагнозування). В їх числі можна окремо виділити задачі для пошуку оптимальних рішень, так званої, задачі подорожуючого торговця (TSP)[6-7]. В своїй роботі я більш детально розбираю можливості Attention нейронних мереж, тобто мереж на основі "привертання уваги". В роботі розглядається побудова алгоритму багатошарової рекурентної самонавчальної нейронної мережі. Результатом роботи очікується покращення існуючих варіантів розв'язків задач оптимізації та зменшення витрат часу, кількості БПЛА, інших ресурсів на проведення доставки вантажів дроном або системою дронів.

Мета магістерської атестаційної роботи – синтез оптимального способу маршрутизації системи безпілотних літальних апаратів з урахуванням ваги.

Об'єкт дослідження – процес оптимізації повітряних маршрутів для системи безпілотних літальних апаратів.

Предмет дослідження – система навігації (маршрутизації) безпілотних літальних апаратів.

Методи дослідження – метод мурашиної колонії, метод пошуку найближчого маршруту через симуляцію відпалу, метод порівняльного аналізу результатів, метод нейронної мережі на основі механізму уваги, метод симуляції реального польоту системи БПЛА (квадрокоптерів).

Актуальність дипломної роботи. Сучасні системи управління безпілотними літальними апаратами в основному цифрові; розробка, обслуговування та експлуатація таких систем тісно пов'язані з навігаційними

системами GPS. До нині у нас немає однозначно гарного рішення відносно навігаційних (маршрутизаційних) систем для безпілотних літальних апаратів. Наявні системи не дають достатнього рівня впевненості в результаті. Саме тому я поставив перед собою задачу створити алгоритм оптимізації маршрутів для системи БПЛА з використанням нейронної мережі, щоб в подальшому, за бажанням, його можна було використати на реальному підприємстві або в наукових цілях.

Наукова новизна отриманих результатів. Було створено власну модифікацію нейронної мережі на мові Python апробовані нові закордонні методи, які можуть бути використані реальними логістичними підприємствами та в інших проблемах, де нині намагаються застосувати штучний інтелект.

Практичне значення отриманих результатів полягає в наступному: запропонований алгоритм оптимізації маршрутів системи БПЛА дозволяє підвищити ефективність маршрутизації системи дронів, зменшити час на доставку, зменшити кількість дронів до оптимальної, зменшити витрату ресурсів при використанні дронів. Як результат, створити умови більш ефективного та доцільного використання системи БПЛА.

Апробація результатів дисертації. Результати проведених у магістерський роботі досліджень частково були опубліковані в журналі “Proceedings of the National Aviation University” та на науково-професійних конференціях студентів і молодих вчених НАУ «Політ» [1-4].

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ЗНАХОДЖЕННЯ ОПТИМАЛЬНОГО МАРШРУТУ СИСТЕМИ БПЛА.

1.1 Роль оптимізації в навігації повітряних маршрутів

Навігація [8] – це галузь знань і технологій про керування транспортним засобом для спрямування його до цілі. Використовується на воді, на землі, в повітрі, а також останніми роками і в космосі. У XX ст., внаслідок розвитку науки і техніки, появи повітряних суден, космічних кораблів — нових об'єктів навігації, виникли нові значення терміну. Тепер, в загальному значенні, навігація — процес керування деяким об'єктом (в тому числі інформаційним), який має властиві йому методи пересування і орієнтації в певному просторі. Наука навігації на сьогоднішній час дуже розвинена, налічує тисячі книжок і посібників. Мені важливо правильно знайти своє місце у цій галузі. В моєму випадку, робота орієнтована лише на розділ навігації, а саме маршрутизацію системи БПЛА, та вибір оптимального шляху проходження об'єкта в просторі.

1.2 Постановка задачі оптимізації маршрутів БПЛА

Оптимізація — це процес знаходження найвигіднішого способу розв’язання проблеми. За своєю природою людина завжди прагне знайти краще рішення та поліпшити наявні в будь-якій проблемі. Проміж інших можна відокремити такі відомі проблеми оптимізації, як задача подорожуючого торговця [6,7,9], задача пакування рюкзака[10] та задача виплавки різного роду елементів із листа сталі[12],

найвигіднішим способом. Я виділив нечіткі терміни «краще» і «поліпшити», бо «цільова функція» може бути різною. Далі про неї буде мова.

У сучасному світі однією з найважливіших питань є питання логістики і все, що з цим пов’язано.

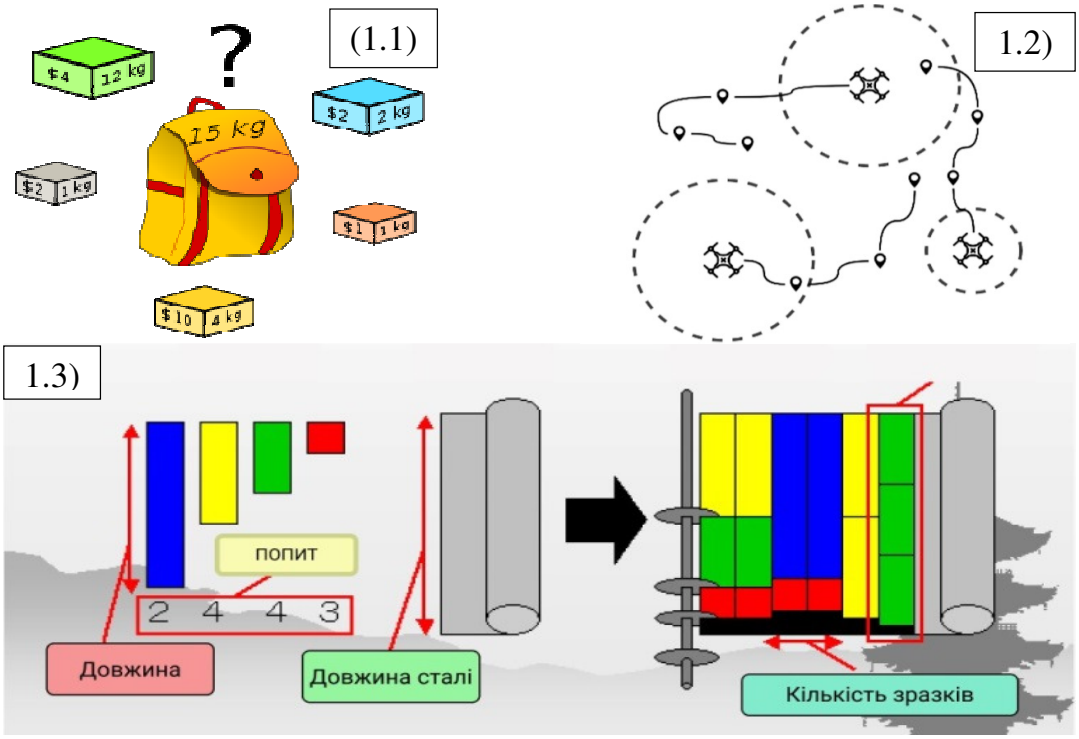


Рис. 1.1 Проблема пакування портфелю [10]

Рис. 1.2.Приклад проблеми подорожуючого торговця для системи дронів [11]

Рис. 1.3 Приклад проблеми ефективної нарізки сталі [12]

Тобто управління матеріальними, інформаційними та людськими ресурсами з метою їх оптимізації. У математиці ж задача оптимізації зводиться до знаходження екстремумів заданої функції в деякому неперервному просторі. Розв’язок такої задачі може включати багато методів, у тому числі для оптимізації використовують так звані евристичні алгоритми (евристики), або алгоритми штучного інтелекту — це алгоритми, що описують практичний підхід, який не є точним або оптимальним, але достатнім для розв’язку поставленої задачі в заданих межах, будь то часу або коштів.

Як відомо, екстремум функції – це найбільше чи найменше її значення на заданій множині. Розрізняють локальні та глобальні екстремуми. Локальний – це екстремум в деякому достатньо малому околі даної точки, глобальний – в усій розглядуваній області значень.

В загальному випадку задача оптимізації зводиться до знаходження точки (або точок) глобального екстремуму або декількох екстремумів заданої функції.

Нехай задано функцію $f(x)$, визначену на множині X із n-вимірного евклідового простору . Необхідно знайти точки мінімуму (або максимуму) значень функції $f(x)$ в X, де $f(x)$ досягає найбільшого чи найменшого значень. Інколи таких точок декілька, тобто задача не має однозначного розв'язання, що можна було б перевірити за поліноміальний час, і тоді існує кілька «найкращих» з точки зору практики стратегій.

Тобто:

$f(x) \rightarrow \min$, $x \in X$. де $f(x)$ — цільова функція, X — допустима множина, кожна точка x цієї множини — допустима точка задачі. Для того щоб правильно поставити задачу оптимізації необхідно задати:

1. Допустиму множину — множину X.

2. Цільову функцію $f(x)$.

3. Критерій пошуку (максимум або мінімум).

Найпростішим і найточнішим є використання так званого методу "грубого перебору" за якого ми перебираємо всі можливі варіанти рішень і обираємо найкращий. Оскільки більшість задач оптимізації є NP-повними – це означає, що неможливо знайти найкраще рішення за «поліноміальний час», тобто перебір рішень для 12 точок вже буде тривати близько години, а для 13 взагалі декілька тижнів. Тобто такий розв'язок вже немає практичного сенсу, з точки зору бізнесу.

Цим методом доволі корисно перевіряти рішення до 10 точок, що і було продемонстровано в попередніх роботах [1 - 4], де я порівнював відразу 5 евристичних алгоритмів.

В своїх попередніх роботах, я ставив перед собою задачу вирішення самого примітивного сценарію знаходження оптимального шляху між точками на карті, тобто, вирішував базову задачу подорожуючого торговця, більш відому як Travelling Salesman Problem(TSP). В загальному формулюванні вона полягає у знаходженні найвигіднішого маршруту, що проходить через вказані

міста хоча б по одному разу. В умовах завдання вказуються критерій вигідності маршруту (найкоротший, найдешевший, сукупний критерій тощо) і відповідні матриці відстаней, вартості тощо.

Розв'язки такого роду задачі є тривіальним, тобто не несуть за собою складності розрахунків, для кількості точок на карті, що не перевищує 10, і не має великого значення для простого польоту, однак все змінюється зі збільшенням кількості точок для відвідування та загальної кількості дронів в системі.

Задля вирішення проблем навігації та маршрутизації таких систем використовують більш складні алгоритми, котрі враховують відразу багато факторів взаємодії. Це можуть бути як натуральні перешкоди у вигляді лісів, гір так і штучні типу будівель, доріг, інших літальних апаратів. Одним із можливих підходів вирішення проблеми навігації таких систем є принцип синергетики [13].

На сьогодні вже є багато різних методів та алгоритмів оптимізації, але оскільки задача сама по собі є NP-повною, то завжди буде актуальним пошук найкращих методів її вирішення. До такого роду евристичних алгоритмів можна віднести:

1) Алгоритм оптимізації мурашиної колонії [14]
2) Алгоритм імітації відпалу [15]
3) K-opt алгоритм [16]
4) Concorde TSP solver [17]

Більш детально сутність цих алгоритмів буде викладено в 2 розділі.

Із них найбільш вдалим можна вважати метод Concorde, оскільки за його допомогою була вирішена проблема подорожуючого торговця для всіх міст нашої планети [17]. На цей алгоритм зазвичай рівняються інші комерційні та дослідні роботи в сфері оптимізації.

В наш час більшість доступних алгоритмів є комерційно невигідними і доступні тільки в межах наукових робіт. Серед їх числа такі відомі рішення як

Google OR tools [18]. Таким чином вирішення проблем бізнесу є також важливим фактором подальших дослідів в цьому напрямку.

З огляду на результати попередньої роботи, в якій я робив порівняльний аналіз для тієї самої кількості точок 10 та 19 аеропортів, всі евристичні алгоритми не є ідеальними рішеннями навіть для 10 точок, оскільки здатні впадати в локальні мінімуми. В минулій роботі я проаналізував роботу 5 алгоритмів штучного інтелекту на прикладі вирішення задачі комівояжера для аеропортів України, побудувавши їх програмні реалізації в середовищі MATLAB. Метод мурашиної колонії, як і більшість схожих евристичних алгоритмів, показав досить гарний результат відносно помилки обчислень серед, попередньо знайденого локального мінімуму, та швидкодії розрахунків.

Перші спроби створити нейронну мережу були досить невдалими, оскільки мережа дуже часто вироджувалась, тобто не могла зрозуміти що від неї хочуть побачити на виході, і відповідно не могла видати оптимального рішення за короткий час. Це був мій перший досвід роботи з самонавчальними мережами, після написання простого одношарового, а потім і багатошарового перцептрону.

Алгоритм імітації відпалу показав себе найкращим чином серед інших евристичних алгоритмів він працює доволі швидко і виявився найбільш точним. Саме результати довжини повітряних маршрутів цього алгоритму я брав за еталон під час порівняння з іншими локальними мінімумами. Я вважаю, що зважаючи на його точність буде доцільно його і надалі використовувати під час навчання моєї багатошарової нейронної мережі.

В даній магістерській роботі я обрав розробку ускладненої версії цієї проблеми, з урахуванням можливих постановок проблеми, з обмеженнями у вигляді наявності системи об’єктів, вагових обмежень на доставку вантажу та за найменшої кількості БПЛА, потрібних для виконання цього завдання. Також під час симуляції польоту та навчання нейронної мережі були враховані повітряні коридори, щоб БПЛА не зіштовхувались один з одним.

Обмеження кількості безпілотних літальних апаратів обумовлена потребами бізнесу, оскільки основна мета – це завжди мінімізація витрат, та максимізація прибутків компанії. Хотілося б аби розроблений мною алгоритм оптимізації міг бути використаний у різного роду службах онлайн-доставки вантажів, в пошуково-рятувальних операціях з використанням системи дронів, операціях різного роду досліджень ландшафту та екологічного нагляду за певною місцевістю, зрошування місцевості інсектицидами.

1.3 Огляд середовищ моделювання польоту систем БПЛА

На сьогодні є багато різних варіантів моделювання польоту, досліджень пошуку траєкторій, в більшості своїй програми написані під моделювання саме польоту літаків. У більшості вони є комерційними, закритими від сторонніх фірмами-розробниками, але в наш час вже з’явились і програми з відкритим кодом, що дозволяють виконувати складні моделювання як окремих БПЛА так і цілком систем із багатьох безпілотних літальних апаратів. Для написання роботи я розглянув декілька із них, про це і піде мова далі.

Симулятори польоту безпілотника можуть коштувати від 5 до 250 доларів. Ціна варіюється від того, наскільки надійним ви хочете, щоб ваш симулятор безпілотника був.

Якщо використовувати їх задля простого дослідження, то не потрібно інвестувати в дорогий симулятор. З іншого боку, якщо потрібно навчитися літати в конкретних сценаріях для комерційних операцій безпілотників, витратити більше на професійний тренажер, може мати сенс тому, що це інвестиції в бізнес, які згодом можуть багатократно окупитись. Програми з управління дроном називаються «наземними станціями», оскільки управління відбувається дистанційно, і на «Наземну станцію» приходить інформація про стан дрона з його борту. Одним з таких ПО є «Qground Control» рисунок 1.4.

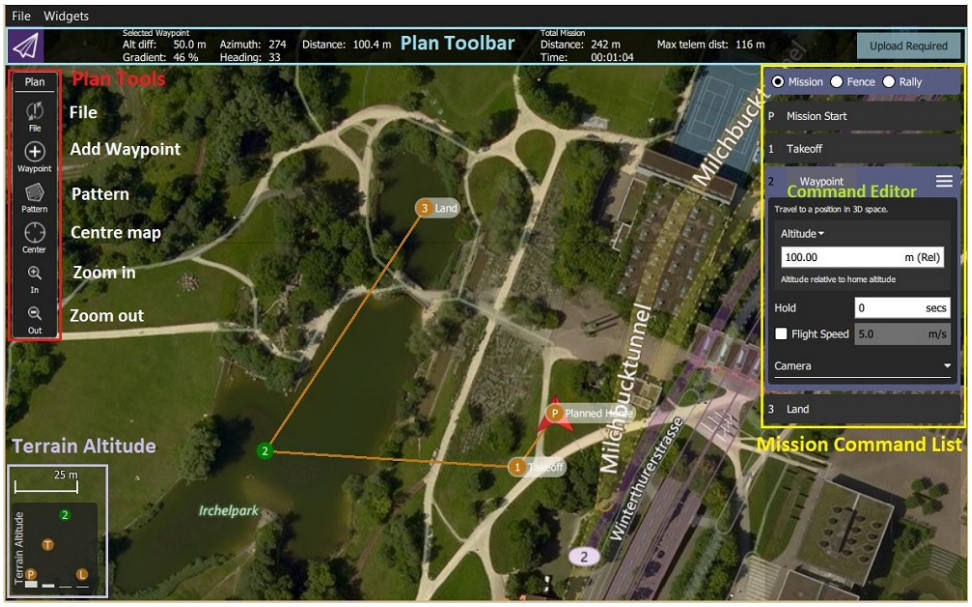


Рис. 1.4. Приклад роботи програми-симулятора Qground Control [19]

У даній програмі можна проводити наладку і конфігурацію як самого дрона, так і його польотної місії. Дане ПО є вельми гнучким, оскільки підтримується операційними системами Windows, MacOS, Android, IOS, тобто є можливість встановлювати їх на такі мобільні пристрої, як смартфони і планшети.

Але ПО такого типу дозволяє керувати лише реальними дронами, на котрі

Рис. 1.5. Приклад «польоту дрону» в середовищі Mission Planner [20]

потрібно витратити від 4 тисяч гривень, що не є частиною мети моєї роботи, тому я вирішив використати інший підхід і обрав програму моделювання польоту, що дозволяє доволі точно відтворювати політ БПЛА без його фізичної наявності. Все, що потрібно для роботи такого ПО як Mission Planner, уже є його вбудованим функціоналом.

Цю програму можна використовувати не тільки для простих симуляцій польоту, але і для більш складних наукових робіт та розрахунків. Сумарно таке програмне забезпечення може коштувати міліони доларів, але воно відкрите для всіх бажаючих, що також є великим плюсом на користь його використання. Ще однією важливою особливістю, котра дозволяє керувати автоматично польотом дрону є використання Python, як мови програмування. За допомогою вже написаних бібліотек:

Бібліотека DroneKit-Python[21] дозволяє розробникам створювати додатки, які запускаються на бортовому комп'ютері супутника і спілкуються з бортовим контролером ArduPilot[22] по каналу зв'язку з найменшою затримкою. Бортові додатки можуть значно поліпшити роботу автопілота, підвищивши інтелектуальність поведінки транспортного засобу і виконувати завдання, що вимагають великих обчислювальних і часових витрат (наприклад, комп'ютерний зір, планування траєкторій або 3D-моделювання). DroneKit-Python також може бути використаний для додатків наземних станцій, які спілкуються з транспортними засобами за вищої затримки RF-зв'язку.

API зв'язується з транспортними засобами по MAVLink[23]. Він забезпечує програмний доступ до телеметрії підключеного транспортного засобу, інформацію про стан і параметри, а також дозволяє керувати місією і надає безпосередній контроль над рухом транспортних засобів і операцій.

1.4 Режими симуляції польоту БПЛА

В більшості БПЛА підтримуються так звані режими польоту “flight modes”, або сценарії польоту. Вони регламентують поведінку дрона в тих чи інших ситуаціях. Деякі з них, як наприклад режим повернення додому Return to

Launch (RTL)[24], слугують невід'ємною частиною кожного польоту, оскільки дозволяють, в разі аварійної ситуації, або іншої непередбачуваної ситуації коректно повернути БПЛА на початкове положення.

В своїй роботі я використав наступні режими польоту:

Режим повернення додому або Return to Launch (RTL). При активації даної функції квадрокоптер летить «додому», тобто відшукує шлях до Наземної

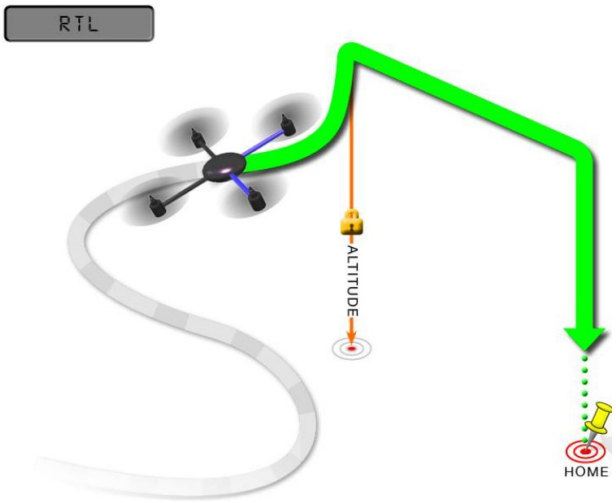


Рис. 1.6. Режим повернення додому [24]

станції і виконує всі операції посадки. Орієнтиром служить сигнал від апаратури управління. Так як багато моделей квадрокоптера не оснащені ні компасом, ні GPS (Global Positioning System), то часто це є єдиним способом визначити в який бік летіти. Але через слабкий сигнал бюджетних пультів режим працює не завжди надійно.

У тих же моделях в яких присутній GPS режим Return to Launch буде керувати квадрокоптера повернутися в початкове положення, тобто де був знятий з охорони (arming). Таким чином вихідне положення (home position) завжди має бути фактичним становищем GPS при зльоті апарату.

При виборі режиму RTL квадрокоптер повернеться в початкове домашнє положення. Спочатку він підніметься на висоту зазначену в параметрі RTL_ALT перш ніж почне рух в сторону будинку (або буде підтримувати поточну висоту, якщо вона перевищує вказаний параметр RTL_ALT).

У цьому режимі коптер запам'ятовує початкову точку спостереження (це також може бути пульт, телефон з GPS, спеціальний браслет або просто об'єкт, що рухається) і рухається за нею на зазначеній в програмних налаштуваннях відстані та висоті. На даний момент багато виробників ставлять цю функцію

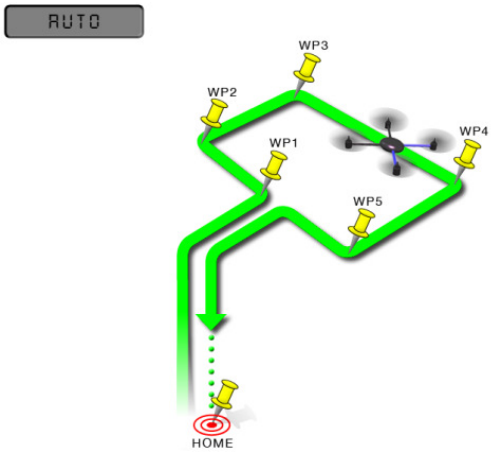


Рисунок 1.7 – Політ по точкам GPS [25]

однієї з найважливіших пріоритетів, але все ж в професійних коптерах часто ця функція відсутня в цілях безпеки оскільки ризик зіткнення БПЛА з перешкодою або з самим пілотом дуже великий.

Обліт по точкам GPS званий режимом AUTO або політ по точкам. В автоматичному режимі квадрокоптер буде слідувати заздалегідь запрограмованому сценарію, що зберігається в пам'яті автопілота, яка складається з навігаційних команд (тобто точок) і команди "зробити" (тобто команди, які не впливають на розташування квадрокоптера) на рисунку 1.7 представлений приклад маршруту польоту дрона в автоматичному режимі по точкам GPS. В своїй роботі під час моделювання я використовував деякі з режимів, в тому числі і так званий керований режим Guided mode (Рис1.8)

Перевага цього режиму в тому, що під його дією є можливість відправляти сигнали управління напряму в наземну станцію Mission Planer. А вже саме середовище симуляції буде виконувати взліт, сам політ та посадку БПЛА. Для цього я буду використовувати мою програму, в котрій прорахую маршрут БПЛА в, попередньо тренованій нейронній мережі, та подам координати для

польоту в симулятор БПЛА. Тим самим імітуючи реальну ситуацію з доставкою грузів в різні точки та повернення на місце дислокації. Для цього я обрав мову програмування Python.

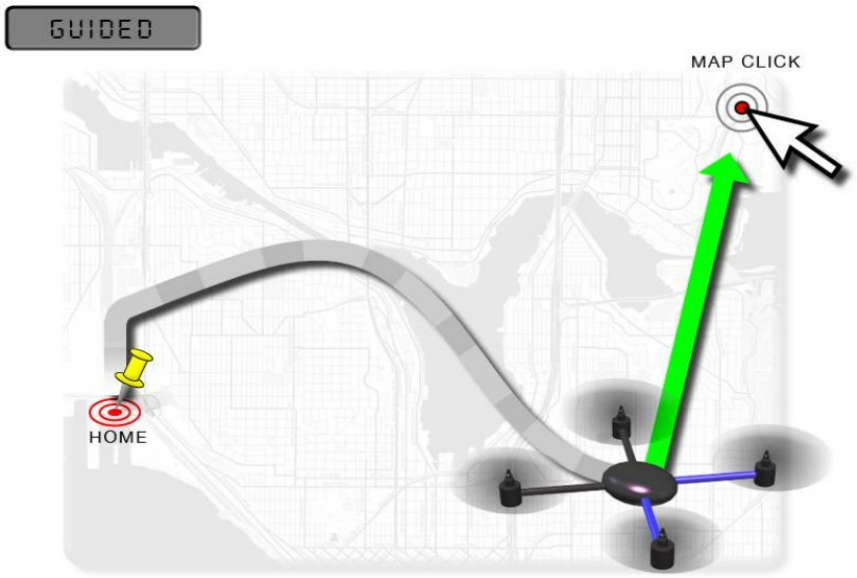


Рис. 1.8 Приклад роботи квадрокоптера в керованому режимі [26]

Автономна система керування польотом квадрокоптера з можливістю обліту перепон та комплексної навігації має наступну схематичну модель рисунок 1.9.

Рис. 1.9 Автономна система керування польотом квадрокоптера.
Рисунок було зроблено мною в програмі Figma. [27]

ВИСНОВОК ДО РОЗДІЛУ 1

У цьому першому розділі приведено короткий огляд різних евристичних алгоритмів, серед яких можна виділити алгоритм Concorde TSP Solver та алгоритм імітації відпалу. Поставлено низку задач розробки алгоритму побудови маршрутів для систем БПЛА з урахуванням оптимізації маршруту, вантажу, що може нести дрон, та висотних коридорів. Все назване – актуальні практичні задачі, які нині обговорюються переважно у закордонній літературі.

Узагальнено результати попередніх робіт автора [1-4]. З урахуванням недоліків попередніх робіт, вирішено їх врахувати в даній магістерській роботі і розробити рекурентну нейронну мережу на основі алгоритму «привертання уваги» (Recurrent Attention Neural Network) зі «вчителем» та генетичних алгоритмів, для її навчання, котра може бути використана в подальшому для побудови оптимальних маршрутів як окремих БПЛА так і їх систем. Тим самим сприяти підвищенню ефективності навігації, а саме маршрутизації, системи дронів за допомогою найсучасніших методів оптимізації.

Розглянути режими польоту (сценарії), що можна зробити при симуляції польоту в Mission Planner. Для цього доцільно використати бібліотеку Python, а саме DroneKit-Python.

РОЗДІЛ 2. АЛГОРИТМИ ПОШУКУ ОПТИМАЛЬНИХ ТРАЕКТОРІЙ ПОЛЬОТУ.

2.1 Оптимізаційні алгоритми на основі штучного інтелекту

Генетичний алгоритм (ГА). Це еволюційний алгоритм пошуку, основним принципом якого є використання механізмів, що є аналогічними до біологічної еволюції, для розв'язання задач оптимізації. Для цього

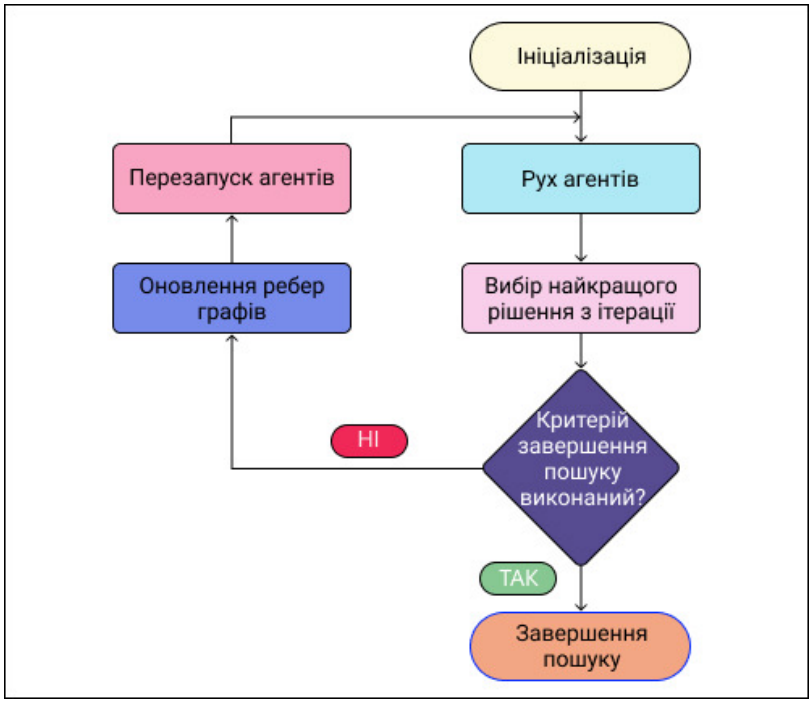


Рис. 2.1 Загальний принцип роботи генетичних алгоритмів

Рисунок побудовано автором в програмі Figma[27].

виконується послідовний підбір, комбінування і варіації шуканих параметрів. Особливістю генетичного алгоритму є використання так званого оператора «схрещення», який виконує операцію рекомбінації (тобто складає нове рішення з існуючих), роль якої аналогічна ролі схрещення при розмноженні в живій природі.

Принцип мурашиного алгоритму оптимізації полягає в моделюванні реальної поведінки мурах в природі. Початково мурахи блукають в просторі довільним чином, і по знаходженні їжі повертаються до своєї колонії, залишаючи по собі феромонний слід. Після цього якщо інші мурахи знаходять такий шлях, вони схильні до припинення своїх блукань, тобто вони починають слідувати позначеним шляхом, посилюючи його під час повернення у разі знайдення їжі. Проте, з часом, феромонові сліди

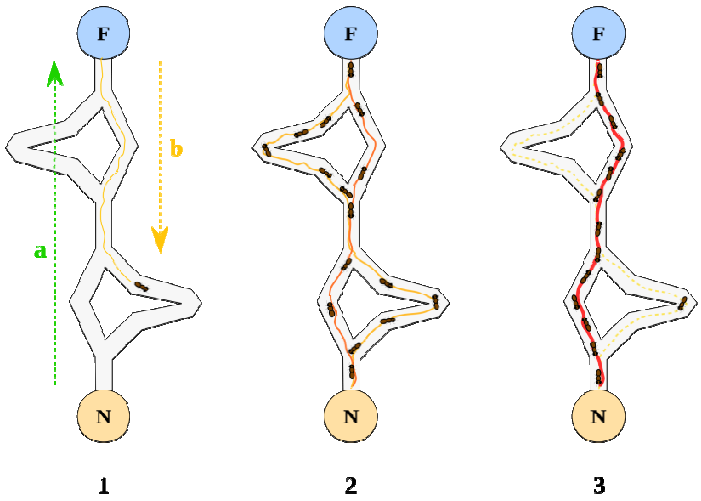


Рис. 2.2 Приклад роботи алгоритму мурашиної оптимізації

випаровуються, одже привабливість шляхів зменшується. Тобто, очевидно, чим більше часу потрібно мурасі, щоб подолати дорогу, тим більше часу мають феромонові сліди, щоб випаруватись. Натомість, коротка дорога проходиться швидше (відповідно і частіше), отже щільність феромонів стає більшою на короткому шляху. Випаровування феромонів також надає перевагу уникнення локально найкращих шляхів. Якби випаровування не відбувалось взагалі, шляхи обрані першим мурахою ставали б вкрай привабливими для наступних, тобто розвідка можливих шляхів була б обмежена. Таким чином, коли мураха знаходить вдалий (тобто коротший з наявних) шлях з колонії до джерела їжі, інші мурахи більш ймовірно слідуватимуть йому, і такий позитивний зворотний зв'язок в підсумку призведе до обрання цього кращого шляху всіма мурахами. Отже, ідея

мурашиного алгоритму полягає в наслідуванні поведінки колонії мурах, що прогулюються графом, який представляє проблему для розв'язання.

Алгоритм імітації відпалу [29] - алгоритм, в якому процес пошуку глобального екстремуму імітує фізичний процес відпалу, який полягає у наступному:

У рідкій фазі частинки (атоми та молекули) розташовуються випадковим чином, а в основному стані твердого тіла усі частинки розташовані в добре структуровані гратки, для яких відповідна їх енергія мінімальна. Ми отримуємо основний стан твердого тіла тільки тоді, коли максимальне

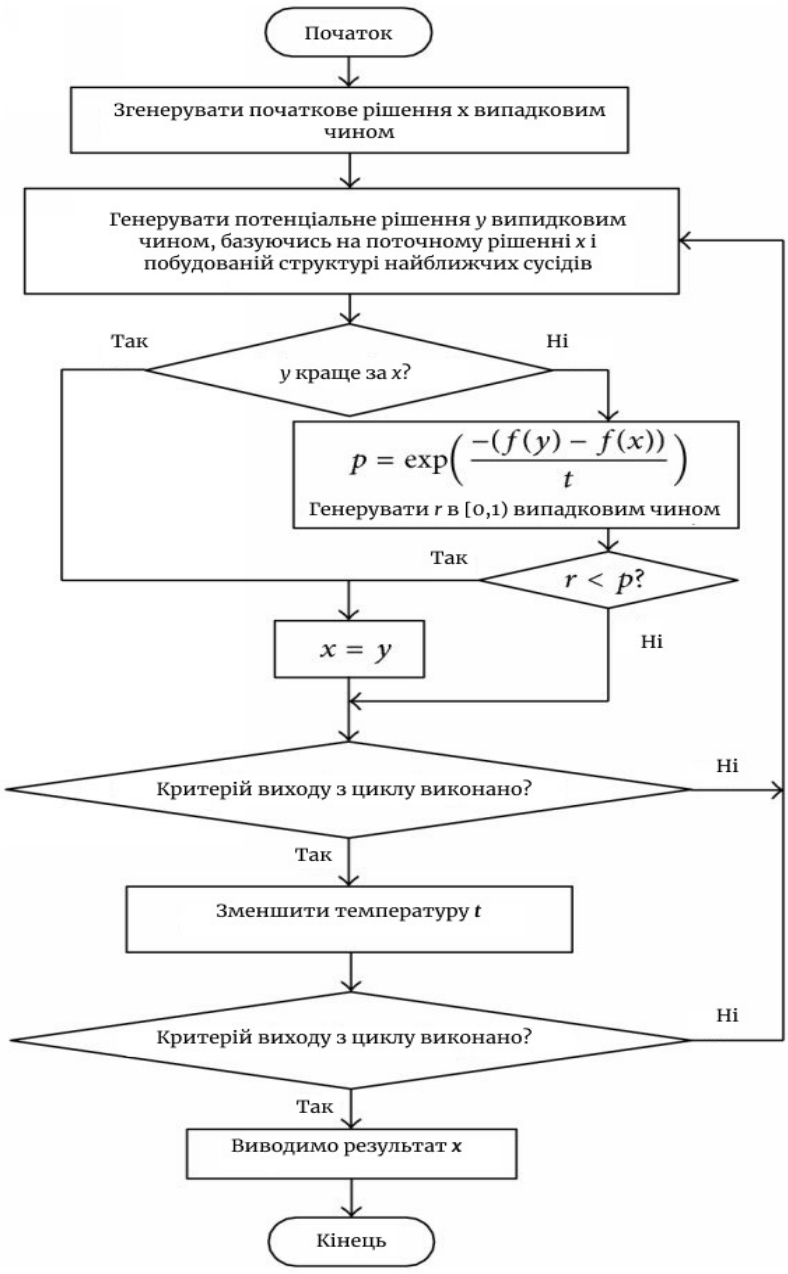


Рис 2.3 Блок-схема роботи алгоритму імітування відпалу [29]

значення температури досить високе і охолодження здійснюється досить повільно. В іншому випадку, тіло застигне в деякому метастабільному стані, а не в істинному.

Отже, даний алгоритм ґрунтується на імітації фізичного процесу, який відбувається при кристалізації речовини з рідкого стану в твердий, у тому числі при відпалі металів. Передбачається, що атоми вже вишикувалися в кристалічну решітку, але ще допустимі переходи окремих атомів з однієї комірки в іншу. Такий процес протікає при поступовому зниженні температури. Перехід частинки з однієї комірки в іншу відбувається з деякою ймовірністю, причому вона зменшується з пониженням температури. Стійка кристалічна решітка відповідає мінімуму енергії атомів, тому атом або переходить в стан з меншим рівнем енергії, або залишається на місці.

2.2 Навігація систем об’єктів

Задача пошуку оптимальних маршрутів для системи об’єктів зводиться до знаходження рішень спочатку для кожного окремого об’єкту а потім і для всієї системи. На сьогоднішній день прикладом такого способу навігації є навігація в логістичних компаніях та службах таксі.

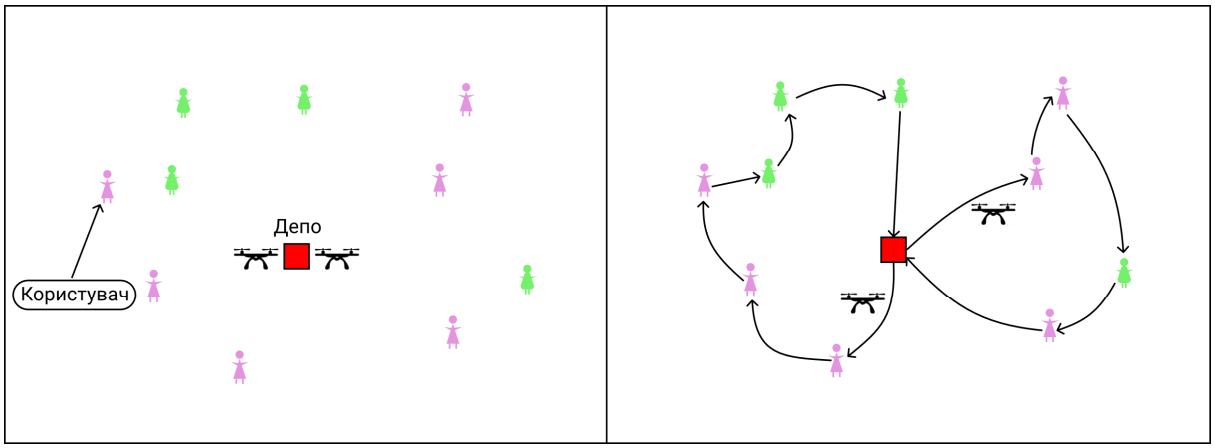


Рис. 2.4 Загальний вигляд задачі пошуку маршрутів системи об’єктів

Рисунок побудовано автором в програмі Figma[27].

Ця задача може бути сформульована різними способами:

1) Проблема маршрутизації транспортних засобів (Vehicle routing problem)

Це проблема комбінаторної оптимізації та цілочисельного програмування, яка задається питанням: "Який оптимальний набір маршрутів для автопарку, який необхідно пройти, щоб доставити за цими маршрутами клієнтів". Вона узагальнює добре відому проблему подорожуючого торговця.

Визначення оптимального рішення для VRP [30] є NP-повним, тому розмір завдань, які можуть бути вирішені, оптимально, за допомогою математичного програмування або комбінаторної оптимізації, може бути обмежений. Тому комерційні вирішувачі схильні використовувати евристику в зв'язку з розмірами і частотою реальних VRP, з якими вони постійно зіштовхуються.

Дорожню мережу можна описати за допомогою графіка, де дуги - це дороги, а вершини - це перехрестя між ними. Дуги можуть бути спрямовані або не направлено в зв'язку з можливою наявністю вулиць в одну сторону або різними витратами в кожному напрямку. Кожна дуга має супутню вартість, яка зазвичай є її довжину або час пробігу, яке може залежати від типу транспортного засобу.

Для того щоб знати глобальну вартість кожного маршруту, необхідно знати вартість проїзду і час у дорозі між кожним клієнтом і депо. Для цього наш оригінальний графік трансформується в такий, де вершини є клієнтами і депо, а дуги - дороги між ними. Вартість кожної дуги - найменша вартість між двома точками первісної дорожньої мережі. Це легко зробити, так як проблеми з найкоротшим шляхом відносно легко вирішуються. Це перетворює розріджений вихідний графік в повний. Для кожної пари вершин ***i*** та ***j*** існує дуга (***i, j***) повного графіка, вартість якої записується як вартість найкоротшого шляху від ***i*** до ***j***. Час в дорозі - це сума часу в дорозі дуг на найкоротшому шляху від і до j на вихідному дорожньому графіку.

Існує безліч способів вирішення проблем з маршрутизацією транспортних засобів вручну. Наприклад, оптимальна маршрутизація - це

велика проблема ефективності навантажувачів на великих складах. Деякі з ручних методів дозволяють вибрати найбільш ефективний маршрут. Проте, процентна різниця між оптимальним ручним методом маршрутизації і реальним оптимальним маршрутом становить в середньому 13%.

2) Multi Agent Travelling Salesman Problem [31]

Існує ряд реальних проблем, які вимагають від декількох агентів відвідувати цікаві для них області, виконувати завдання і подорожувати між ними. До них зазвичай відносяться такі проблеми, як спостереження, розвідка або пошуково-рятувальні роботи.

Загальне формулювання проблеми Multi-Agent Travel Salesman:

Спочатку визначимо індекси ***i*** та ***j*** позначимо завдання з безлічі T завдань від 1 до N, безліч A агентів від 1 до M і матрицю c_{ija} для позначення вартості агента при переході від завдання i до задачі j. Додатково ми визначимо трехіндексную двійкову змінну рішення:

Спочатку визначимо індекси ***i*** та ***j*** позначимо завдання з набору T завдань від 1 до N, набору A агентів від 1 до M і матриці c_{ija} для позначення вартості ***a*** агента, що переміщається з завдання ***i*** в ***j*** завдання. Додатково визначаємо трьохіндексну двійкову змінну рішення:

$$x_{ija} = \begin{cases} 1 & \text{якщо агент } a \text{ відвідує точку } j \text{ після } i, \\ 0 & \text{в іншому випадку} \end{cases}$$

$$\min_{x_{ija}} \sum_{i-1}^{N} \sum_{j-1}^{N} \sum_{a-1}^{M} c_{ija} x_{ija}$$

$$\sum_{i=1}^{N} x_{ipa} - \sum_{j=1}^{N} x_{pja} = 0, a \in A, p \in T$$

$$\sum_{j=1}^{N} x_{1ja} = 1, \forall a \in A$$

$$u_i - u_j + N \sum_{a=1}^{M} x_{ija} \leq N - 1, \forall i \neq j \neq 1$$

$$x_{ija} \in 0,1 \forall i, j, a$$

Мета полягає в тому, щоб звести до мінімуму загальні витрати всіх агентів, які прямують поставленими завданнями. Наступні обмеження гарантують, що кожна задача відвідується тільки один раз, в той час як обмеження щодо збереження потоку свідчать, що як тільки агент відвідує завдання, він також повинен відступити від неї. Обмеження по "одному

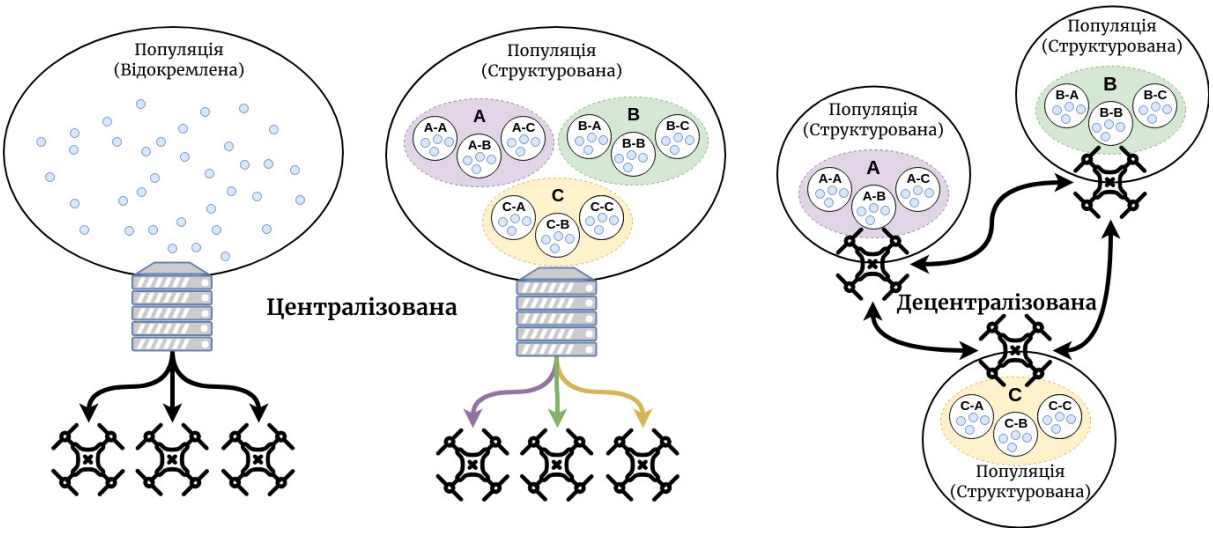


Рис. 2.5 Приклад вирішення Multi-Agent Travel Salesman Problem [31]

агенту" гарантують, що кожен агент використовується тільки один раз, а обмеження щодо усунення підмаршрутів використовуються, коли мова йде про u додаткових невід'ємних допоміжних змінних рішення, із u_i зазначенням відповідної задачі і-ої, відомої як "потенціали вузла".

2.3 Теорія нейронних мереж

Під нейронною мережею розуміють об'єднання нейронів між собою за допомогою зв'язків, що називаються синапсами. Нейрон можна розглядати як певну функцію, що отримує вхідні дані та видає якийсь результат. Це так званий “чорний ящик”.

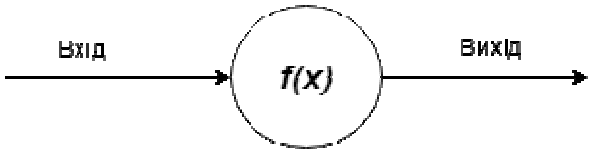


Рис.2.6. [32] Будова штучного нейрону

Кожен синапс має свою вагу. Це працює таким чином, що чим більший коефіцієнт ваги, тим більш значний вплив нейрон буде здійснювати на інші нейрони. Словом, основними елементами мережі нейронів є нейрони, які об'єднуються в шари певним чином.

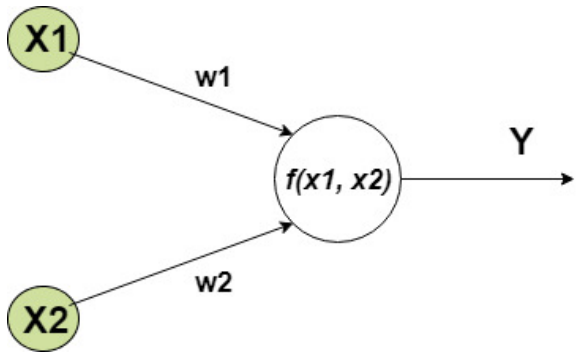


Рис. 2.7. Приклад роботи нейрону [32]

Нейрон може мати один або відразу декілька входів, кожен зі своєю вагою. Цими входами можуть бути також виходи інших нейронів.

X1 та X2 — вхідні дані;

w1 та w2 — вага синапсів;

f(x1, x2) — функція нейрона, що перетворює вхідні дані;

Y — вихідне значення.

Вхідне значення нейрона (також називають сукупний ввід) він обчислюється за наступною формулою:

$$net = \sum_{i=1}^{n} x_i w_i + b$$

де n — кількість входів,

x — вхід,

w — вага,

b — зміщення

Щоб це порахувати нам потрібно для кожного вхідного зразка перемножити вхідні дані на вагу, а потім всі добутки додати. Отримане значення і буде вхідним значенням для нейрона. Далі, це значення необхідно піддати дії функції, яку ще називають функцією активації. Вихідне значення

такої функції і буде виходом нейрона. Цю операцію потрібно повторити для всіх нейронів у нейронній мережі. Називається це - прямим проходом, або (forward propagation).

Для оцінки якості роботи моделі використовується зворотний прохід (back propagation), що складається з декількох кроків, які потрібно повторити для кожного нейрона у кожному шарі.

У кожній нейронній мережі існує один вхідний шар, хоч один прихований та один вихідний шар. Коли нейрони кожного шару з'єднуються з кожним нейроном наступного шару, то вони утворюють багатошаровий перцептрон

(MLP — multilayer perceptron). Якщо мережа має більше ніж один прихований шар, то її прийнято називати глибокою.

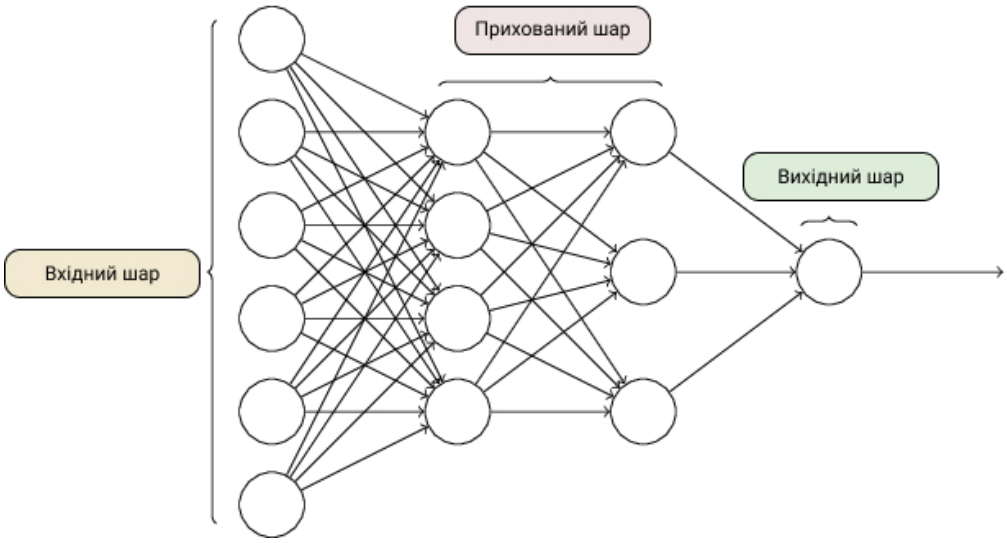


Рис. 2.8. Багатошаровий перцептрон [32]

2.4 Розробка власної нейронної мережі

Свою нейронну мережу я вирішив побудувати на основі передової технології використання механізму привертання уваги в так званих (Attention Recurrent Neural Networks) [33].

Рис. 2.9. Принцип роботи згорткової нейронної мережі під час розпізнавання картинок
Проілюстровано автором за допомогою Figma[27]

Attention це - спосіб повідомити мережі, на що варто звернути більше уваги, тобто повідомити ймовірність того чи іншого результату в залежності від стану нейронів, що надходять на вхід. Реалізований в Keras [34] шар Attention сам виявляє, на основі навчальної вибірки, фактори звернення уваги, на які знижує помилку мережі. Виявлення важливих факторів здійснюється через метод зворотнього поширення помилки (back propagation), подібно до того як це робиться для згорткових мереж (Convolutional Neural Networks). Згорткові нейронні мережі себе добре зарекомендували при розпізнаванні текстів та патернів поведінки в зображеннях. Принцип їх роботи наочно описано на рисунку 2.9.

Підхід Attention застосовується також для роботи з текстом, а також звуком і часовими рядами. Для обробки тексту широко використовуються рекурентні нейронні мережі (RNN, LSTM, GRU). Attention може або доповнювати їх, або замінювати їх, переводячи мережу до більш простих і швидких архітектур.

Одне з найвідоміших застосувань Attention це застосування його для того, щоб відмовитися від рекуррентної мережі і перейти до повнозв'язної моделі. Рекурентні мережі мають серією недоліків: неможливість здійснювати навчання на GPU, швидко наступає перенавчання, тобто система починає вироджуватись і будувати неоптимальні прогнози з часом. За допомогою механізму Attention ми можемо вибудувати мережу здатну до вивчення послідовностей на базі повнозв'язної мережі, навчити її на GPU.

Головною перевагою Attention моделей є те, що вони більш складні з точки зору розрахунків, але ми можемо зменшувати похибку рішення до безкінечності. Тим самим якість отриманих результатів буде залежати тільки від потужностей комп'ютера та відеокарти на якій ми навчаємо мережу.

За допомогою сторонніх Python бібліотек типу attention-keras [36] та TensorFlow [35] побудову таких мереж можна дуже швидко організувати декількома рядками коду:

```
from attention_keras.layers.attention import AttentionLayer
attn_layer = AttentionLayer(name='attention_layer')
attn_out, attn_states = attn_layer([encoder_outputs, decoder_outputs])
```

Механізм нейронної уваги оснащує нейронну мережу здатністю зосереджуватися на підмножині її входів (або особливостей): він вибирає конкретні входи. Нехай $\mathbf{x} \in R^d$ вектор входу, тоді $\mathbf{z} \in R^d$, вектор вивчення особливостей, $\mathrm{a} \in [0,1]^k$ вектор уваги, $g \in R^k$ побіжна увага і $\mathrm{f}_\phi(\mathrm{x})$ мережа уваги з параметрами ϕ. Як правило, увага реалізується як:

$$\mathrm{a} = \mathrm{f}_\phi(\mathrm{x}),$$

$$g = a \odot \mathrm{z}$$

де ⊙ - поелементне множення, в той же час z - вихід іншої нейромережі f_θ (x) з параметрами θ. Є два типи уваги: м'яка та жорстка. Якщо говорити

про м'якому увагу, вона примножує риси з (м'якою) маскою значень між нулем і одиницею, а при жорсткій увазі, коли ці значення обмежені точно нулем або одиницею, а саме $a \in \{0,1\}^k$. В останньому випадку маска жорсткої уваги може використовуватися для безпосередньої індексації вектора ознаки: $\tilde{g} = z\,[a]$ (в нотації Матлаб), який змінює свою розмірність, а тепер $\tilde{g} \in R^m$ при $m \leq k$.

Недавні дослідження показують, що машинне навчання має потенціал для вивчення кращих евристичних методів, ніж розроблені людиною.

Глибока нейронна мережа використовується для характеристики вхідного екземпляру і в подальшому поступової побудови можливого рішення. Для вирішення задачі маршрутизації декількох літальних апаратів я буду використовувати наступну модель (Рис 2.1)

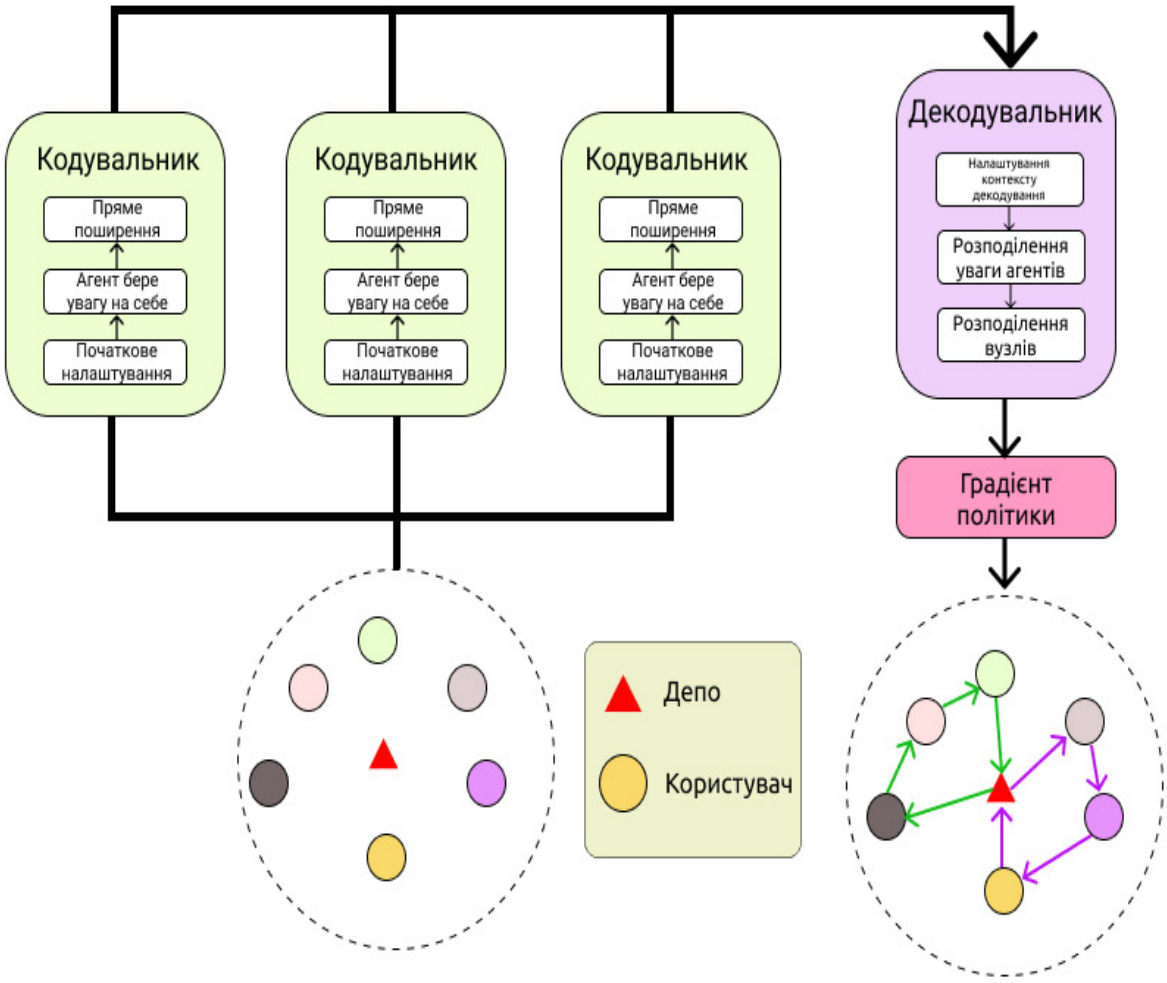


Рис. 2.10 Структура динамічної моделі нейронної мережі типу кодувальник-декодувальник

Малюнок створено автором за допомогою Figma [27]

У цій моделі стан БПЛА представлений функціями вузла, які фіксуються з часом. Однак справа в тому, що стан БПЛА змінюється відповідно до рішення, оскільки модель зроблена на різних етапах побудови, то і функції вузла повинні бути відповідно оновлені.

Тому в своїй роботі я вирішив використати модель з динамічним розподіленням уваги агентів та за архітектурою динамічного кодера-декодера, що дозволяє моделі динамічно досліджувати особливості вузла та ефективно використовувати інформацію про приховану структуру на різних етапах побудови.

Структурні особливості вхідного екземпляру у вигляді графу витягуються кодером. Потім структура будується поступово за допомогою декодера.

Зокрема, на кожному етапі побудови, декодер передбачає розподіл по вузлах, потім один вузол вибирається і додається до кінця часткового рішення.

Основні ідеї:

- Використовувати навчання з підкріпленням для створення агента(нейрону), який може вивчати евристику та надавати оптимальні рішення.
- Використати Graph Attention Networks (GAT) [37], щоб створити відповідні вбудовування графіків для агента.
- Політика агента відповідно до навчання з підкріпленням регулюється декодером.

Підхід до моделі динамічного привертання уваги:

Після повернення БПЛА до депо, решта вузлів можуть розглядатись як новий (менший) екземпляр (граф), який потрібно вирішити.

Суть алгоритму в тому, щоб оновити побудову решти вузлів за допомогою кодера після повернення агента до депо.

Реалізація:

1. Примусити агента (навчання з підкріпленням) чекати інших агентів.
2. Коли кожен агент знаходиться в депо, застосовуємо кодер з маскою до всього графу.

Динамічна модель на основі механізму “привертання уваги” навчається за допомогою градієнта політики а потім алгоритму REINFORCE з базовим рівнем (baseline), тобто основною мережею, в яку будуть потім додані тільки найкращі рішення.

Рис. 2.11 Схема процесу навчання моделі

Рисунок створено автором в Figma [27]

Базова лінія - це копія моделі з фіксованими вагами однієї з попередніх епох. Цей етап потрібен для змішування попередньо тренованих шарів з новими та створення своєї, окремої логіки вибору оптимальних значень нейронів. Потім ми оновлюємо базовий рівень наприкінці епохи, якщо різниця у витратах на модель-кандидат та базовим рівень є статистично значущою.

Базова лінія використовує окремий набір даних для цієї перевірки. Цей набір даних оновлюється після кожного знайденого варіанту покращення

базової лінії частинними її рішеннями. За ідеал беруться вже попередньо відібрані коректні рішення.

Частково наступне математичне підґрунтя було взято з ресурсів [38,39].

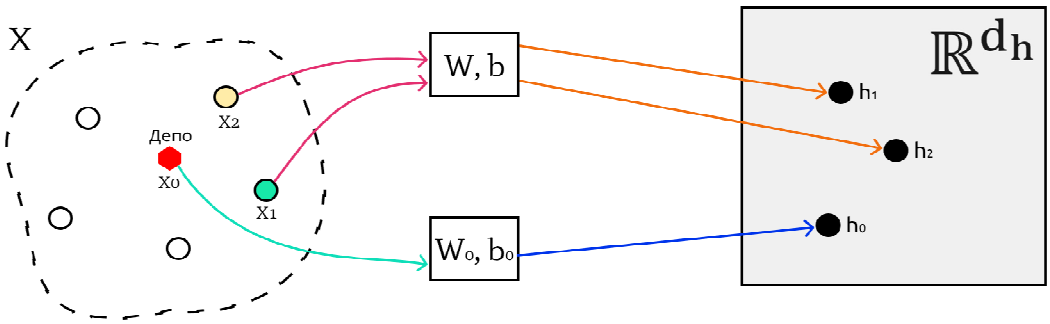


Рис.2.12. 1Процес кодування групи (batch) нейронів

Рисунок побудовано автором в програмі Figma[27].

Вірогідність рішення для маршруту $\pi = \{\pi_1, ..., \pi_T\}$ і графу X:

$$p_\theta(\pi|X) = \prod_{t=1}^{T} p_\theta(\pi t|X, \pi_{1:t-1})$$

$$h^{(0)} : \mathbb{R}^3, \rightarrow \mathbb{R}^{d_h}, d_h = 128$$

Для кожного вузла лінійно проектуємо $x_i = (s_{1i}, s_{2i}, \text{попит}_i) \in \mathbb{R}^3$ to $\mathbb{R}^{128}$

Параметри, що піддаються навчанню $W \in \mathbb{R}^{3 \times d_h}$, $b \in \mathbb{R}^{d_h}$. Також розділяємо параметри W_0 , b_0 що використовуються для депо:

$$h_i^{(0)} = \begin{cases} W \cdot x_i + b & i \neq 0 \\ W_0 \cdot x_i + b_0 & i = 0 \end{cases}$$

1. Використовуємо нейронне передавання повідомлення, Self-Attention механізм(використовуємо вже побудовані успішні графи задля навчання моделі) – це потрібно щоб прибрати необхідність подавати на вхід купу, вже пройдених іншим алгоритмом, маршрутів.

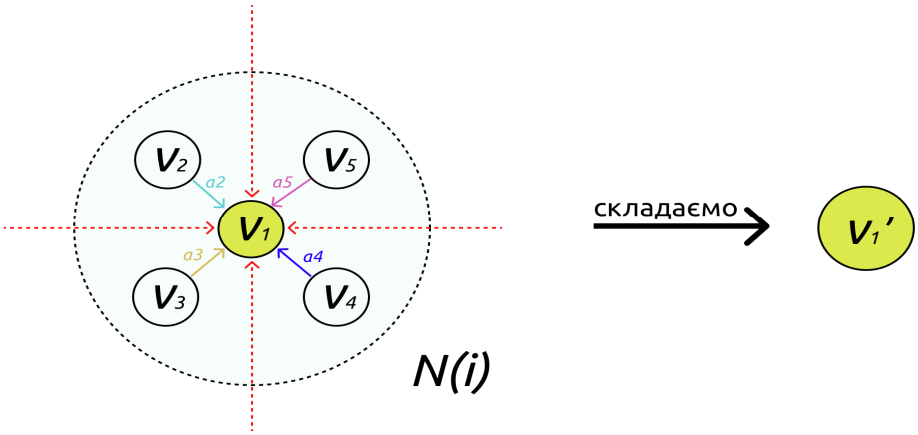


Рис. 2.13. Процес обчислення ваги за механізмом привертання уваги

Рисунок створено автором у Figma [27]

2. Вбудовування (прихований стан) для кожного вузла $\mathbf{v}_i$ отримується зваженою сумою особливостей усіх вузлів $v_{j \in N(i)}$ в деякій околиці N (i).
3. Ваги a_i обчислюються за механізмом уваги, представляючи важливість (подібність) кожного сусіда для конкретного вузла.

Лінійно проектуємо початкові вбудовування вузлів

$\mathrm{H} \in \mathbb{R}^{batch \times \mathrm{n} \times \mathrm{d}_h}$ (запит, ключ, значення):

$$Q = HW^Q, K = HW^K, V = HW^V, W^Q W^K W^V \in \mathbb{R}^{d_h \times d}$$

Розбиваємо на M задач і обчислюємо матрицю сумісності $A \in \mathbb{R}^{batch \times M \times n \times n}$ для вузлів графу:

$$A = softmax\left(\frac{U}{\sqrt{d/M}}\right) = softmax\left(\frac{QK^T}{\sqrt{d/M}}\right)$$

Рахуємо вагові повідомлення для кожної з задач:

$$H' = AV \in \mathbb{R}^{batch \times M \times n \times d/M}$$

Конкатинуємо задачі і проєктуємо $W^O \in \mathbb{R}^{d \times d}$:

$$MHA = Concat(H'_1, \ldots, H'_M)W^O \subset \mathbb{R}^{batch \times n \times d}$$

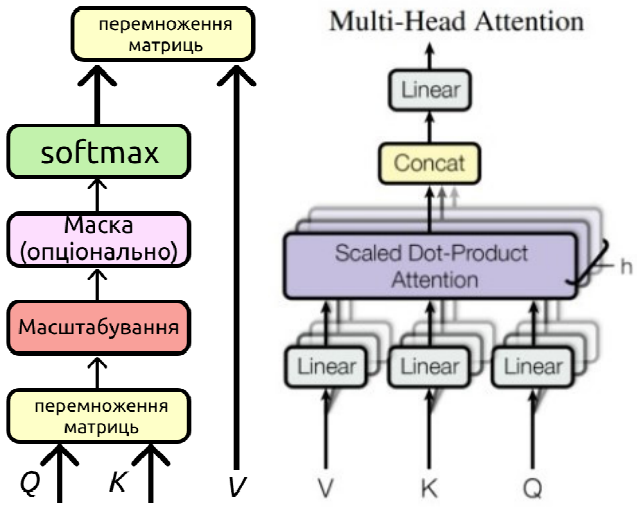


Рис. 2.14. , Рис. 2.15 [38] Принцип роботи багатозадачної моделі уваги

Для кожного вузла я застосовую повністю підключену мережу прямого проходу (Feed-forward, FF) за допомогою зв'язків пропускання (skip-connections), $l \in \{1, ..., N\}$:

$$\hat{h}_i^{(l)} = tanh\left(h_i^{(l-1)} + MHA_i^{(l)}\left(h_0^{(l-1)}, \ldots, h_n^{(l-1)}\right)\right),$$

$$FF\left(\hat{h}_i^{(l)}\right) = W_1^F ReLu\left(W_0^F \hat{h}_i^{(l)} + b_0^F\right) + b_1^F,$$

$$h_i^{(l)} = tanh\left(\hat{h}_i^{(l)} + FF\left(\hat{h}_i^{(l)}\right)\right)$$

Нарешті, після N шарів ми отримуємо кінцеві вбудовування вузлів:

$$h_i^N = ENCODE_i^N(h_0^0, \ldots, h_n^0)$$

На кожному кроці побудови t ∈ 1, ..., T об'єднуємо середнє вбудовування графу по всіх вузлах, вбудовування раніше вибраного вузла та залишкової ємності БПЛА:

$$\hat{h}_c = \begin{cases} [h_t;\ h_0^N;\ D_t] & t = 1 \\ [h_t;\ h_{\pi_{t-1}}^N;\ D_t] & t > 1 \end{cases}$$

Політика регулюється двома послідовними рівнями уваги в декодері.

1. Вектор запиту з контекстного вектора: $\boldsymbol{q} = W^Q \hat{h}_c$ Ключі та Значення від вбудовування вузлів.
2. Додаємо маску до уваги: вузли маски, які вже були відвідані або мають занадто великий попит.
   - Багатошаровий рівень уваги:

$$\boldsymbol{q} - W^Q \tilde{\boldsymbol{h}}_c, k_i = W^K \boldsymbol{h}_i, \boldsymbol{v}_i - W^V \boldsymbol{h}_i$$

$$\boldsymbol{u}_j = \begin{cases} \boldsymbol{q} \cdot \boldsymbol{k}_j^T & \boldsymbol{d}_j \leq \boldsymbol{D}_t, \boldsymbol{x}_j \notin \pi_{1:t-1} \\ -\infty & \text{в іншому випадку} \end{cases}$$

   - Одношаровий рівень уваги (лише сумісність) для ймовірностей

$$k_{tanh\,i} - \boldsymbol{W}^{K_{tanh}} \boldsymbol{h}_i,$$

$$\boldsymbol{u}_j = \begin{cases} \boldsymbol{C} \cdot \boldsymbol{tanh}\,(\boldsymbol{q} \cdot \boldsymbol{k}_{tanh_j}^{\boldsymbol{T}})\boldsymbol{d}_j \leq \boldsymbol{D}_t, \boldsymbol{x}_j \notin \pi_{1:t-1} \\ -\infty \qquad \text{в іншому випадку} \end{cases}$$

$$\mathrm{p}_\theta(\pi t|\mathrm{X}, \pi_{1:t-1}) = \frac{e^{u_j}}{\sum_{j'} e^{u_{j'}}}$$

1. Після того, як транспортний засіб повернеться в депо, інші вузли можна розглядати як новий (менший) екземпляр (граф), який має бути вирішений.

2. Оновлення вбудовування залишку вузлів за допомогою кодувальника, коли агент повертається в депо.

$$h_i^t = \begin{cases} ENCODE_t^N(h_0^0, \ldots, h_n^0) \; \pi_{t-1} = x_0 \\ h_i^{t-1} \qquad \text{в іншому випадку} \end{cases}$$

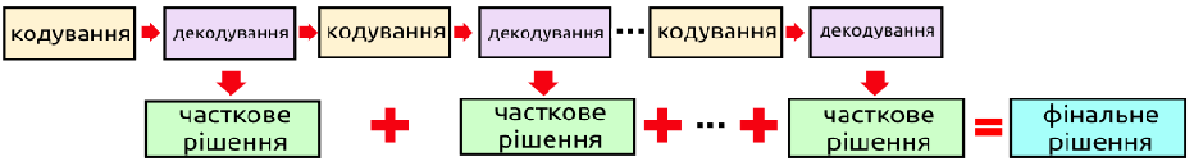


Рис. 2.16. Процес знаходження рішення нейронної моделі методом складання часткових рішень і формування базової лінії (baseline)

Рисунок побудовано автором в програмі Figma[27].

Реалізація:

Примусити агента RL чекати інших, як тільки він досягне x_0, тобто депо.

Коли всі знаходяться в депо, застосуйте кодер з маскою до всієї партії.

Ми тренуємо модель, використовуючи градієнт політики:

Градієнт очікуваної вартості маршруту:

$$\nabla_\theta J(\theta) \sim \mathbb{E}_p [(L^p(X,\pi) - b(X)) \nabla_\theta \log(p_\theta(\pi|X))],$$

де умовна ймовірність рішення:

$$p_\theta(\pi|X) = \prod_{t-1}^{T} p_\theta(\pi t|X, \pi_{1:t-1})$$

і b – базова лінія

- Базова лінія - це копія моделі з фіксованими вагами з однієї з попередніх епох.
- Використовуємо розминку для ранніх епох: змішуємо експоненціальну ковзну середню (контрольовану β = const) вартості моделі за минулі епохи з базовою моделлю.
- Розминка контролюється α ∈ [0; 1].
- Оновлюємо базовий рівень наприкінці епохи, якщо різниця у витратах для моделі-кандидата та базової лінії є статистично значущою (t-тест).

- Для цього порівняння базова лінія (Baseline) використовує окремий набір даних. Цей набір даних оновлюється після кожного відновлення базової лінії.

Потім оцінюємо вартість моделі Монте-Карло: генеруємо епізод $S_1, A_1, \ldots, S_T, A_T$, дотримуючись $p_\theta(\cdot \mid \cdot)$ в режимі вибірки (стохастична політика).

Потім проходимо всі кроки, щоб отримати вартість усього епізоду.

Оцінюємо базову лінію в "жадібному" режимі (вибираємо вузол з найбільшою ймовірністю - детермінована політика).

Оцінюємо градієнт відповідно до формули градієнта політики та оновлюємо ваги нейронної мережі.

Потім слідуємо наступному алгоритму:

1. Створюємо новий навчальний набір даних (1 280 000 випадкових графічних екземплярів) на початку кожної епохи, окрім першої, котру генеруємо з попередньо заготованого набору вирішиних маршрутів (розігрів).
2. Створюємо і зберегаємо набір перевірок (10000 екземплярів графів) із фіксованим значенням довільної змінної для першої епохи. В кінці кожної епохи перевіряємо модель у "жадібному" режимі на предмет оптимальних рішень.

Для будь якої нейронної мережі дуже важливим етапом є вибір гіперпараметрів. В моєму випадку я дослідним шляхом знайшов оптимальний набір:

Таблиця 2.1.

Параметр	Значення
Розмір вбудовування (batch size)	128
Зразки	10000
Кількість епох розминки	1
Відсікання норми градієнта	1.0
Розмір партії перевірки	1000
Розмір валідаційної множини для перевірки	10000
Кількість задач у MHA (Multi Head Attention)	8
Відсікання Tanh (C)	10
Нейрони прихованого шару для прямого проходу	512
Експоненціальна бета-версія розминки	0,8

2.5 Засоби програмування та візуалізації результатів

Для своєї роботи я вирішив використати Python тому, що у нього інтуїтивно зрозумілий синтаксис та вже написано багато бібліотек для роботи з нейронними мережами. Прикладом таким бібліотек є TensorFlow[35], Keras[34], Pytorch[40]. На данний момент TensorFlow від компанії Google має дуже хорошу документацію та безліч різних прикладів для створення власних мереж, використовуючи декілька рядків коду. Тому можна приділити більше уваги побудові структурної моделі власної мережі. Під час пошуку наявних варіантів реалізації я наштовхнувся на ще одне рішення від Google - GoogleOR tools[41]. В ньому були приклади побудови оптимізаційних алгоритмів і можна було відібрати деякі ідеї стосовно реалізації звідти. Оскільки вони компілюють всі рішення через свій

фреймворк на більш низькорівневі мови (C, C++), їх алгоритми працюють набагато швидше ніж у конкурентів.

Задля візуалізації часткових випадків маршрутів є лише декілька, доцільних в моєму випадку варіантів, це – бібіліотека візуалізації matplotlib та plotly. Я вирішив використати plotly, оскільки можливості візуалізації в Python доволі обмежені, а ця бібліотека використовує API браузера, в якому налаштовує локальний сервер і вже туди будує візуалізацію.

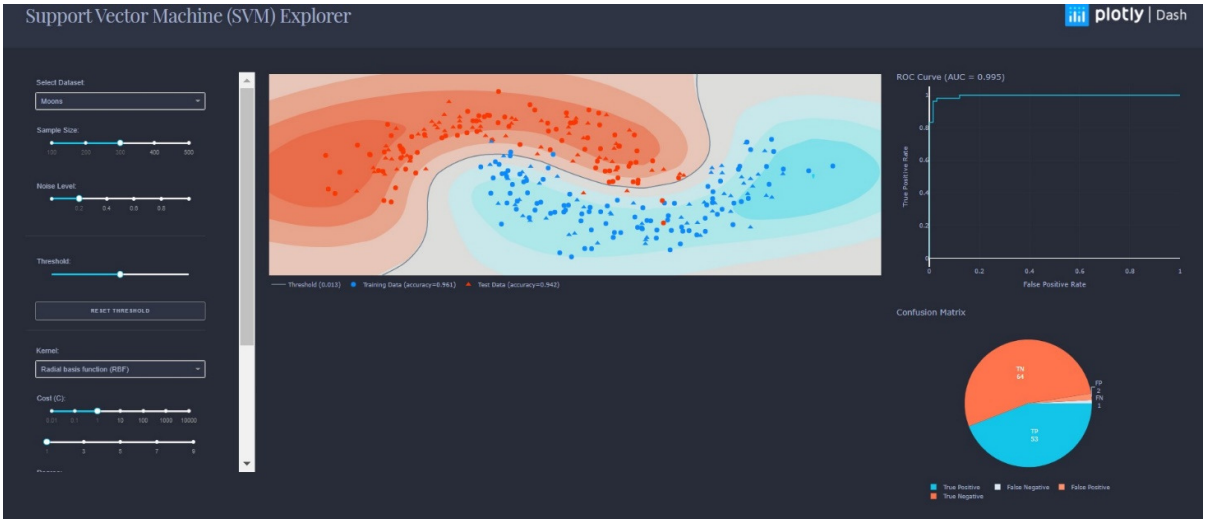


Рис. 2.17 Приклад візуалізації задачі кластеризації в Plotly [42]

Для симуляції польоту, як я вже попередньо вказував, я вирішив використати Mission Planner. Але для того, щоб в керованому режимі відсилати команди управління в програма-симулятор у нас повинні бути бібліотеки-посередники. Оскільки управління дроном це низькорівневий процес і всі команди туди йдуть уже у вигляді бінарного коду, постає задача компіляції більш високорівневих команд. Для цього я використав дві бібліотеку Dronekit [43-44], а саме:

Dronekit-sitl(Software in the loop) – це найпростіший і найшвидший спосіб запуску SITL на Windows, Linux або Mac OS X. Він встановлюється за допомогою інструмента pip Python на всіх платформах і працює шляхом завантаження та запуску попередньо побудованого бінарного коду, що притаманний певному літальному апарату.

З’єднання з Mission Planner відбувається за допомогою створення окремого серверу з данними, і потім уже з нього, підключившись до Mission Planner командою

```
dronekit-sitl copter
```

ми можемо контролювати параметри транспортного засобу. Для цього уже використовується інший файл з командою, вказаною нижче.

```
mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 127.0.0.1:14550 --out 127.0.0.1:14551
```

Тим самим ми визначаємо до якого серверу підключаємось і на які порти будемо передавати команди керування.

Виконуючи в подальшому програму-скрипт

```
ANN_goto.py --connect udp:127.0.0.1:14550
```

Ми передаємо команди керування через порти, що були відкриті попередньою командою. Це все відбувається в трьох окремих терміналах. Це ще одна з причин доцільності використання Python для вирішення такого роду задач.

ВИСНОВОК ДО РОЗДІЛУ 2

Описано та проілюстровано різні евристичні алгоритми, серед яких можна виділити алгоритм Concorde, алгоритм оптимізації мурашиної колонії та алгоритм імітації відпалу. Роботу двох останніх я перевірив особисто.

Обговорено питання маршрутизації системи об’єктів та принципіальну різницю між звичайними розрахунками і тими, що видають штучні алгоритми.

Розглянуто загальні принципи побудови нейронних мереж. Описано будову елементів нейронних мереж а саме: поняття нейрону, побудова простого персептрону, побудова багатошарової нейронної мережі, згорткові нейронні мережі та принципи їх дії, нейронні мережі в основі яких лежить метод привертання уваги. Описано математичну модель рекурентної нейронної мережі на основі привертання уваги зі вчителем, що буде використана в наступних розділах роботи.

Окрема увага приділена середовищу розробки та моделювання на основі мови Python. Описані основні бібліотеки, що будуть використані в роботі а також методи їх взаємодії між собою для ілюстрування часткових випадків маршрутизації системи БПЛА, а також симулювання польоту в реальних умовах системи дронів в межах програми Mission Planner.

Все це готує мій розгляд згаданих питань у наступних розділах роботи.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ.

Після того, як було сформульовано основні теоретичні положення, можна розпочати його програмну реалізацію. При цьому доцільно орієнтуватися на практичні постановки задач, які вже сьогодні виникають в сучасній економіці та бізнесі, як її частині. А саме, цей розділ буде присвячено розв'язанню наступних проблем: мінімізація маршруту окремого дрона, мінімізація маршруту системи дронів, оптимізація ускладнених завдань дрону: висотні коридори під час польоту системи дронів; мінімізація кількості БПЛА, необхідних для достатньо ефективного виконання завдання; урахування вагових особливостей завдань користувачів з огляду на вантажопідйомність дронів (залежить від конструкції [5], але на сьогодні не перевищує 30кг. для загальних цілей).

3.1 Тренування мережі

Тренування мережі є невід'ємною частиною правильного функціонування і головною відмінністю від інших алгоритмів. Нейронні мережі за своєю суттю відрізняються методами навчання. Їх можна класифікувати таким чином: нейронні мережі з вчителем; нейронні мережі без вчителя, або їх ще називають "самонавчальні"; комбіновані нейронні мережі. В роботі я використовую комбіновану нейронну мережу на основі так званого методу "привертання уваги" (Attention mechanism).

В цьому розділі буде розглянуто підґрунтя процесу навчання нейронної мережі і їх конкретні приклади реалізації у вигляді Python-функцій.

Окремо буде розглянуто з процес нормалізації даних мережі і різні перехідні процеси, що використовуються під час навчання. Серед них, як один із важливих елементів, буде розглянуто використання гіперпараметрів.

3.1.1. Робота з даними

В роботі були використано декілька форматів для роботи з даними. Запис та зчитування параметрів побудови мережі та гіперпараметрів навчання велось через функцію конфігурації та стандартну бібліотеку серіалізації – pickle. В коді це виглядає у вигляді наступної функції:

```
def dump_pkl(args, verbose=True, param_log=True):
    cfg = Config(**vars(args))
    with open(cfg.pkl_path, 'wb') as f:
        pickle.dump(cfg, f)
        print('--- save pickle file in %s ---\n' % cfg.pkl_path)
        if verbose:
            print(''.join('%s: %s\n' % item for item in vars(cfg).items()))
        if param_log:
            path = '%sparam_%s_%s.csv' % (cfg.log_dir, cfg.task, cfg.dump_date)
            with open(path, 'w') as f:
                f.write(''.join('%s,%s\n' % item for item in vars(cfg).items()))
```

Для зчитування використовувалась відповідно функція завантаження:

```
def load_pkl(pkl_path, verbose=True):
    print("Path?  = " + str(os.path.isfile(pkl_path)))
    if not os.path.isfile(pkl_path):
        raise FileNotFoundError('pkl_path')
    print(pkl_path)
    with open(pkl_path, 'rb') as f:
        cfg = pickle.load(f)
        if verbose:
            print(''.join('%s: %s\n' % item for item in vars(cfg).items()))
        os.environ['CUDA_VISIBLE_DEVICE'] = cfg.cuda_dv
    return cfg
```

Як можна бачити з коду, при зчитуванні параметрів додатково використовується перевірка на наявність відеокарти, з допомогою якої, можна більш швидко виконувати розподілення навчальних даних на групи (batches). В цьому, по суті, і полягає перевага нейронних мереж на основі “привертання уваги”.

Для того, щоб можна було паралельно запускати декілька варіантів навчання з різними параметрами при запуску я використовував вбудований модуль argparse. Цей модуль дозволяє запускати незалежні термінали з різними параметрами і все це відбувається за допомогою так званих прапорів.

В коді це виглядає наступним чином:

```
def file_parser():
    parser = argparse.ArgumentParser()
    parser.add_argument('-p', '--path', metavar='P', type=str,
                        default='Parameters/VRP20_train.pkl', help='file path, pkl or h5 only')
    args = parser.parse_args()
    return args
```

Тут можна бачити, що окремим аргументом при зчитуванні параметрів додається шлях до .pkl-файлу з налаштуваннями.

Для збереження проміжних результатів роботи мережі, в загальному випадку епох (стадій навчання), я використовував вбудований в TensorFlow функціонал по збереженню вагових коефіцієнтів в .h5 файли.

В коді це виражено таким чином:

```
model.save_weights('{}{}_epoch{}.h5'.format(cfg.weight_dir, cfg.task, epoch),
save_format='h5')
```

По суті тут вказується лише назва та розширення для збереження.

3.1.2. Налаштування гіперпараметрів навчання

Гіперпараметри – це параметри, що характеризують процес навчання моделі і є важливою частиною тренування, оскільки через них можна емпіричним шляхом підібрати такі значення, що тренування буде максимально ефективним та похибка навчання стане мінімальною. Деякі з них уже були перелічені в 2 розділі Таблиця 1. В цьому розділі я більш детально їх опишу з практичної точки зору. В категорію гіперпараметрів також можна віднести деякі структурні зміни моделі (вигляд функції активації на різних етапах навчання, вибір алгоритму зворотного поширення помилки і т.д.).

Гіперпараметри ніяк не впливають на ефективність нейронної моделі, а лише на швидкість та якість навчального процесу. До них можна віднести параметри, що вказані в таблиці 3.1.

Таблиця 3.1

Параметр	Коментар
Розмірність групи входу (batch)	Зазвичай дорівнює розміру вбудування (масиву даних входу), але може різнитись в залежності від задач.
Кількість зразків (batch * batch_steps)	Розмір зразкової вибірки помножений на кількість цих виборок.
Кількість зразків ініціації головної мережі (створених евристичними алгоритмами як відправна точка)	Ініціалізуємо нашу мережу початковими даними, куди потім будемо додавати часткові рішення та обновляти загальне.
Загальна кількість епох (стадій навчання)	Кількість ітерацій навчання з використанням алгоритмів зменшення помилки.
Розмір вбудування(розмір вектору вхідних даних)	Впливає на розмірність використання вхідних параметрів. Чим ближче до вектору входу, тим краще.
Кількість розгалужень механізму уваги	Впливає на розгалуженість в механізму уваги.
Коефіцієнт прилипання до найближчого значення (Tanh clipping)	Покращує “дослідження” мережі. Покращує вихід шару мережі в бік найближчого значення.
Кількість розгалужень даних в кодувальнику	Впливає на розгалуженість розрахунків з використанням відеокарти.
Коефіцієнт впливу епох розігріву (warm-up beta)	Середня експоненціальна ковзна. Потрібна для ініціалізації початкових епох.
Кількість епох розігріву	Потрібні для початкової ініціалізації входів та ваг зв’язків.
Коефіцієнт навчання.	Головним чином впливає на швидкість та правдоподібність навчання.

3.1.3. Навчання моделі

Навчання моделі починається з головного методу програми train.py (Додаток А). Спочатку завантажуємо конфігурацію навчання і потім запускаємо навчання.

```
if __name__ == '__main__':
    # cfg = load_pkl("./Parameters/VRP20_train.pkl")
    cfg = load_pkl("./Parameters/VRP20_test.pkl")
    train(cfg)
```

Функція навчання виглядає наступним чином:

```
def train(cfg, log_path=None):
    model = AttentionModel(cfg.embed_dim, cfg.n_encode_layers, cfg.n_heads,
cfg.tanh_clipping)
    baseline = RolloutBaseline(model, cfg.task, cfg.weight_dir, cfg.n_rollout_samples,
                               cfg.embed_dim, cfg.n_customer, cfg.warmup_beta,
cfg.wp_epochs)
    optimizer = tf.keras.optimizers.Adam(learning_rate=cfg.lr)
    epoch_loss_avg = tf.keras.metrics.Mean()
    epoch_cost_avg = tf.keras.metrics.Mean()
```

На вхід подаємо конфігурацію та шлях логування (поетапна фіксація дій нейронної моделі з ціллю подальшого виправлення помилок). В самій функції навчання ініціалізуємо модель уваги, головну модель, функцію активації, виставляємо відслідковування загальних втрат кожної з епох та втрат вартості маршруту.

```
    for epoch in range(cfg.epochs):
        dataset = generate_data(cfg.n_samples, cfg.n_customer)

        base_line = baseline.eval_all(dataset)
        base_line = tf.reshape(base_line, (
            -1, cfg.batch)) if base_line is not None else None

        for batch, inputs in enumerate(dataset.batch(cfg.batch)):

            loss, mean_loss, grads = grad_func(model, inputs, base_line, batch)

            grads, _ = tf.clip_by_global_norm(grads, 1.0)
            optimizer.apply_gradients(zip(grads, model.trainable_variables))  #
    optimizer.step

            ave_epoch_loss.update_state(loss)
            ave_cost_loss.update_state(mean_loss)

            if batch % (cfg.batch_verbose) == 0:
                t2 = time()
                print('Epoch %d (batch = %d): Loss: %1.3f cost: %1.3f, %dmin%dsec' % (
                    epoch, batch, ave_epoch_loss.result().numpy(),
    ave_cost_loss.result().numpy(), (t2 - t1) // 60,
                    (t2 - t1) % 60))
        baseline.epoch_callback(model, epoch)
        model.save_weights('{}{}_epoch{}.h5'.format(cfg.weight_dir, cfg.task, epoch),
    save_format='h5')

        ave_epoch_loss.reset_states()
        ave_cost_loss.reset_states()
```

Далі в циклі в залежності від кількості епох навчаємо модель. Спочатку ініціалізуємо входи моделі. Ініціалізуємо головну модель з фінальними ваговими коефіцієнтами. Робимо перестановки даних, щоб модель не запам'ятовувала тільки прямий порядок, котрий був вказаний в початковій вибірці а робила висновки із загального тренду. Далі для кожного з входів формуємо підгрупи (batches). Потім вираховуємо методом градієнтного спуску поточну втрату, середню втрату по вибірці, градієнти вибірки. Функція обрахування градієнтного спуску виглядає наступним чином:

```
def grad_func(model, inputs, base_line, batch):
    with tf.GradientTape() as tape:
        loss, mean_loss = rein_loss(model, inputs, base_line, batch)
    return loss, mean_loss, tape.gradient(loss, model.trainable_variables)
```

Під час обрахування використовуються вбудовані функції бібліотеки TensorFlow, які ми беремо готовими і тут не наводимо.

Далі ми оновлюємо втрати мережі за епоху в порівнянні з найкращим варіантом та середнім з попередніх епох, друкуємо результати підрахунків. Після цього викликаємо функцію зворотного зв'язку і у випадку покращення головної моделі злиттям меншого графа в загальний за зміною в загальній довжині маршруту та кількості БПЛА, оновлюємо головну модель. Далі зберігаємо результати епохи у вигляді ваг мережі в .h5 файл та обнуляємо показники втрат.

3.2 Пояснення до підпрограм

Для того щоб вся програма структура моделі нейронної мережі була відокремлена та читабельна з можливістю подальшого використання, я розбив головну задачу на окремі функції. Таким чином у мене структура мережі, формування параметрів тренування та побудови і сам процес тренування з візуалізацією представляють окремі класи та методи в стилі ООП (Об'єктно Орієнтованого Програмування). В наступних підрозділах піде мова про окремі частини програми.

3.2.1. Розбиття задачі на частини

Для того, щоб можна було підійти до оптимізації головного маршруту, потрібно вибрати метод кластеризації(класифікації) точок за певними параметрами, котрими в загальному випадку і будуть слугувати фактори ваги, часу та відстані, котру ми хочемо врахувати в роботі.

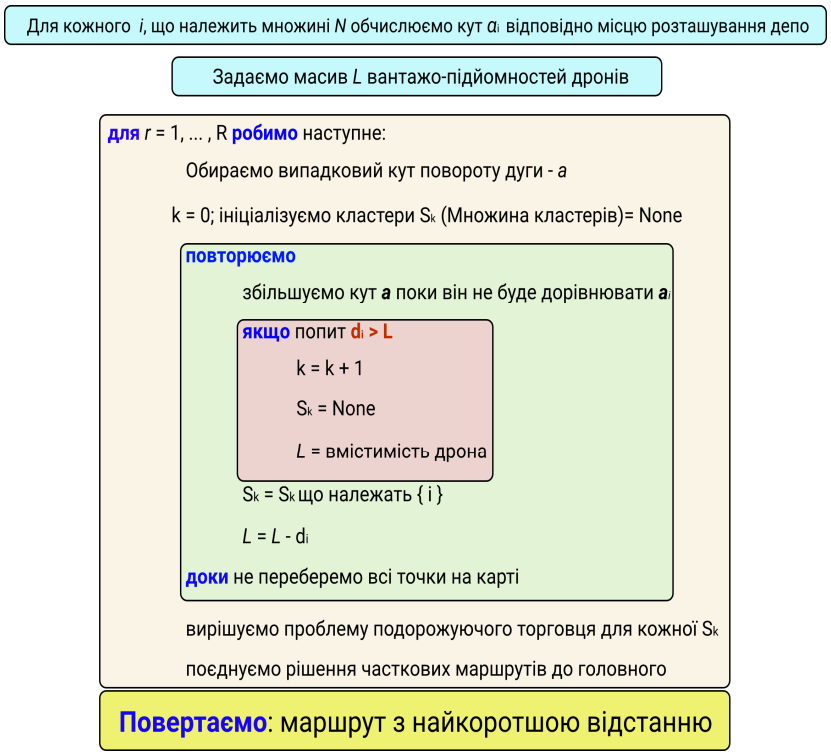


Рис. 3.1 Загальний принцип роботи алгоритму кластеризації та знаходження оптимального результату
Рисунок проілюстровано автором за допомогою Figma [27]

N - набір клієнтів у вигляді точок на карті, тут 0 координатою завжди буде депо. Основна ідея цього алгоритму полягає в тому, що спочатку розглядається окремий маршрут для кожного вузла клієнтів, а потім зменшуємо загальну вартість усього маршруту за рахунок ітераційного злиття маршрутів.

Тут R ми називаємо глибиною рандомізації. Ми вибираємо випадковим чином із $r \in \{1, \cdot \cdot \cdot, R\}$ найкраще можливе злиття. Для кожного r ми розв'язуємо задачу пошуку маршруту між точками і повертаємо рішення

(дугу під певним кутом, що дозволяє відділити кластер точок) з найменшою загальною відстанню.

При виконанні алгоритму нам потрібно кластеризувати (класифікувати) точки на карті за якоюсь ознакою. Одним із таких методів є поворот дуги на певний кут, що виходить з депо, дуга групує вузли на кілька кластерів, що забезпечують при цьому, цілісність кожного окремого кластера не порушуючи цілісності всієї множини рішень. Кожен кластер відповідає окремій задачі подорожуючого торговця, яка може бути вирішена за допомогою точного або приблизного алгоритму. В моєму випадку був використаний алгоритм «імітації відпалу», що показав себе найкраще в попередній роботі[1-4].

Після рішення окремих підзадач алгоритмом імітації відпалу, всі рішення поєднуємо воєдино і отримуємо фінальний оптимальний маршрут.

3.2.2. Структура головної моделі

Для того, щоб створити початкову структуру і в подальшому поновлювати головну модель був написаний програмний клас, який ініціалізується наступним чином:

```
class RolloutBaseline:
    #  Initialize our class with initial parameters
    def __init__(self, model, task, weight_dir, n_rollout_samples=100,
embedded_dimension=128, num_of_customers=20, warmup_beta=0.8,
warmup_epochs=1, from_checkpoint=False, path_to_checkpoint=None,
epoch=0):
```

Сама функція оновлення моделі виглядає ось так:

```
def _update_baseline(self, model, epoch):
    if self.from_checkpoint and self.alpha == 0:
        print('Baseline model loaded')
        self.model = load_model(self.path_to_checkpoint,
embedded_dimension=self.embedded_dimension,
num_of_customers=self.num_of_customers)
    else:
        print('Baseline model copied')
        self.model = copy_model(model,
```

```
embedded_dimension=self.embedded_dimension,
num_of_customers=self.num_of_customers)
    self.dataset = generate_data(n_samples=self.n_rollout_samples,
num_of_customers=self.num_of_customers)

    print(f'Evaluating baseline model on baseline dataset (epoch = {epoch})')
    self.bl_vals = rollout(self.model, self.dataset)
    self.mean = tf.reduce_mean(self.bl_vals)
    self.current_epoch = epoch
```

Суть цієї функції в тому, щоб завантажити модель у випадку, коли вказана точка відновлення, або створити нову модель. Для обрахунків експоненціальної середньої ковзної (лише для епох розминки, тобто епох перших шарів мережі, котра була сформована випадковим чином, або за допомогою алгоритму імітації відпалу) використовувався такий метод:

```
def ema_eval(self, cost):
    if self.mean_value is None:  # first iteration
        self.mean_value = tf.reduce_mean(cost)
    else:
        self.mean_value = self.beta * self.mean_value + (1. - self.beta) *
tf.reduce_mean(cost)
    return self.mean_value
```

Розрахунок для аналізу точок даних шляхом створення серії середніх значень різних підмножин повного набору даних. Його також називають рухомим середнім або ковзаючим середнім і є видом фільтра кінцевої імпульсної характеристики.

Для оцінки втрат поточної головної моделі протягом однієї навчальної епохи, яка може бути використана для епох розігріву була використана функція:

```
def eval(self, batch, cost):
    if self.alpha == 0:
        return self.ema_eval(cost)
    if self.alpha < 1:
        v_ema = self.ema_eval(cost)
    else:
        v_ema = 0.0
    v_b, _ = self.model(batch, decode_type='sampling')
    v_b = tf.stop_gradient(v_b)
    v_ema = tf.stop_gradient(v_ema)
    return self.alpha * v_b + (1 - self.alpha) * v_ema
```

Також для обрахунку втрат всіх моделей було використано:

```
def eval_all(self, dataset):
    if self.alpha < 1:
        return None
    val_costs = rollout(self.model, dataset, batch=512)
    return val_costs
```

Однією з важливих функцій можна вважати функцію для створення зворотного зв'язку, котра визначає різницю між поточною нейронною мережею та потенційно кращим варіантом для загального рішення з огляду на фінальну довжину маршруту:

```
def epoch_callback(self, model, epoch):
    self.current_epoch = epoch
    print(f'Evaluating candidate model on baseline dataset (callback epoch =
{self.current_epoch})')
    candidate_vals = rollout(model, self.dataset)  # costs for training model on
baseline dataset
    candidate_mean = tf.reduce_mean(candidate_vals)
    print(f'Epoch {self.current_epoch} candidate mean {candidate_mean}, baseline
mean {self.mean}')
    if candidate_mean < self.mean:
        t, p = ttest_rel(candidate_vals, self.bl_vals)
        p_val = p / 2
        print(f'p-value: {p_val}')
        if p_val < 0.05:
            print('Update baseline')
            self._update_baseline(model, self.current_epoch)
    if self.alpha < 1.0:
        self.alpha = (self.current_epoch + 1) / float(self.warmup_epochs)
        print(f'alpha was updated to {self.alpha}')
```

3.2.3. Модель уваги

Модель уваги є одним із головних етапів визначення рішень. Це справедливо як для головної мережі так і частинних випадків. В коді це було реалізовано за допомогою вбудованого функціоналу TensorFlow:

```
class AttentionModel(tf.keras.models.Model):
    def __init__(self, embed_dim=128, n_encode_layers=3, n_heads=8,
tanh_clipping=10., FF_hidden=512):
        super().__init__()
        self.Encoder = GraphAttentionEncoder(embed_dim, n_heads,
n_encode_layers, FF_hidden)
        self.Decoder = DecoderCell(embed_dim, n_heads, tanh_clipping)
```

Тут в моделі були задані гіперпараметри, кодувальник та декодувальник, про які мова піде далі.

Для обрахунку вірогідностей покращення маршруту та втрат було написано наступну функцію:

```
    def call(self, x, training=True, return_pi=False, decode_type='greedy'):
        encoder_output = self.Encoder(x, training=training)
        decoder_output = self.Decoder(x, encoder_output, return_pi=return_pi,
decode_type=decode_type)
        if return_pi:
            cost, ll, pi = decoder_output
            return cost, ll, pi
        cost, ll = decoder_output
        return cost, ll
```

3.2.4. Кодувальник

Згідно з моделлю, що я описував в 2 розділі на Рис. 2.10 було створено модель кодувальника. В коді це виглядає так:

```
    class EncoderLayer(tf.keras.layers.Layer):
    def __init__(self, n_heads=8, FF_hidden=512, activation='relu', **kwargs):
        super().__init__(**kwargs)
        self.n_heads = n_heads
        self.FF_hidden = FF_hidden
        self.activation = activation
        self.BN1 = tf.keras.layers.BatchNormalization(trainable=True)
        self.BN2 = tf.keras.layers.BatchNormalization(trainable=True)

    def build(self, input_shape):
        self.MHA_sublayer = ResidualBlock_BN(
            SelfAttention(MultiHeadAttention(n_heads=self.n_heads,
embed_dim=input_shape[2], need_W=True)), self.BN1)
        self.FF_sublayer = ResidualBlock_BN(
            tf.keras.models.Sequential([
                tf.keras.layers.Dense(self.FF_hidden, use_bias=True,
activation=self.activation),
                tf.keras.layers.Dense(input_shape[2], use_bias=True)]), self.BN2)
        super().build(input_shape)

    def call(self, x, mask=None, training=True):
```

```
        return self.FF_sublayer(self.MHA_sublayer(x, mask=mask, training=training),
training=training)
```

Автокодери стали успішною основою для навчання “без нагляду”. Однак звичайні автоматичні кодери не здатні використовувати явні відносини в структурованих даних. Запропонована нижче архітектура не повинна знати структуру графа з маршрутом заздалегідь, і, отже, вона може бути застосована до індуктивного навчання (індуктивне навчання, також відоме як навчання за допомогою відкриття, - це процес, коли учень відкриває правила, спостерігаючи приклади).

```
        class GraphAttentionEncoder(tf.keras.models.Model):
    def __init__(self, embed_dim=128, n_heads=8, n_layers=3, FF_hidden=512):
        super().__init__()
        self.init_W_depot = tf.keras.layers.Dense(embed_dim, use_bias=True)
        self.init_W = tf.keras.layers.Dense(embed_dim, use_bias=True)
        self.encoder_layers = [EncoderLayer(n_heads, FF_hidden) for _ in
range(n_layers)]
    @tf.function
    def call(self, x, mask=None, training=True):
        x = tf.concat([self.init_W_depot(x[0])[:, None, :],
                       self.init_W(tf.concat([x[1], x[2][:, :, None]], axis=-1))], axis=1)
        for layer in self.encoder_layers:
            x = layer(x, mask, training)
        return (x, tf.reduce_mean(x, axis=1))
```

3.2.5. Декодувальник

Суть декодувальника в тому, щоб інтерпритувати вихід моделі кодувальника та за допомогою політики (жадібної або відбору найкращих рішень) вибрати найоптимальніший загальний маршрут серед групи частинних рішень підмереж. В коді ми ініціалізуємо це так:

```
        class DecoderCell(tf.keras.models.Model):
    def __init__(self, embed_dim=128, n_heads=8, clip=10., **kwargs):
        super().__init__(**kwargs)
```

```
        self.Wk1 = tf.keras.layers.Dense(embed_dim, use_bias=False)
        self.Wv = tf.keras.layers.Dense(embed_dim, use_bias=False)
        self.Wk2 = tf.keras.layers.Dense(embed_dim, use_bias=False)
        self.Wq_fixed = tf.keras.layers.Dense(embed_dim, use_bias=False)
        self.Wout = tf.keras.layers.Dense(embed_dim, use_bias=False)
        self.Wq_step = tf.keras.layers.Dense(embed_dim, use_bias=False)

        self.MHA = MultiHeadAttention(n_heads=n_heads, embed_dim=embed_dim,
need_W=False)
        self.SHA = DotProductAttention(clip=clip, return_logits=True,
head_depth=embed_dim)
        # SHA ==> Single Head Attention, because this layer n_heads = 1 which means
no need to spilt heads
        self.env = Env
```

Головним методом в цьому класі є метод виклику декодера і він виглядає ось так:

```
        def call(self, x, encoder_output, return_pi=False, decode_type='sampling'):
    node_embeddings, graph_embedding = encoder_output
    Q_fixed, K1, V, K2 = self.compute_static(node_embeddings, graph_embedding)
    env = Env(x, node_embeddings)
    mask, step_context, D = env._create_t1()
    selecter = {'greedy': TopKSampler(), 'sampling':
CategoricalSampler()}.get(decode_type, None)
    log_ps = tf.TensorArray(dtype=tf.float32, size=0, dynamic_size=True,
element_shape=(env.batch, env.n_nodes))
    tours = tf.TensorArray(dtype=tf.int32, size=0, dynamic_size=True,
element_shape=(env.batch,))
    for i in tf.range(env.n_nodes * 2):
        logits = self._compute_mha(Q_fixed, step_context, K1, V, K2, mask)
        log_p = tf.nn.log_softmax(logits, axis=-1)
        next_node = selecter(log_p)
        mask, step_context, D = env._get_step(next_node, D)

        tours = tours.write(i, tf.squeeze(next_node, axis=1))
```

```
        log_ps = log_ps.write(i, log_p)
    pi = tf.transpose(tours.stack(), perm=(1, 0))
    ll = env.get_log_likelihood(tf.transpose(log_ps.stack(), perm=(1, 0, 2)), pi)
    cost = env.get_costs(pi)
    if return_pi:
        return cost, ll, pi
    return cost, ll
```

Як можемо бачити з коду, то на вхід подаємо вихід кодувальника, або серії кодувальників та вказуємо тип декодування. Серед них можна виділити “відбір” або sampling та “жадібний” greedy. Їх відмінність в тому, що під час виконання методу відбору ми перебираємо всі довжини шляху з огляду на додаткові параметри у вигляді кількості БПЛА, часу, загальної довжини маршруту, довжини маршруту окремих БПЛА. Як результат роботи функція повертає кінцевий шлях у вигляді:

pi = [0, 1, 5, 2, 3, 4, 0]

Вірогідність перспективності рішення для покращенні головної моделі а також функція вірогідності, що в загальному випадку вимірює придатність статистичної моделі до вибірки даних для заданих значень невідомих параметрів.

3.2.6. Функція побудови маршрутів

Для візуалізації частинних рішень було використано бібліотеку plotly і сама функція має вигляд:

```
def plot_route(data, pi, costs, title, index_in_batch=0)
```

На вхід подаємо данні, маршрут, заголовок та індекс моделі в множині рішень.

Результатом роботи отримуємо побудований в вікні браузера шлях на основі рішення нейронної мережі:

```
#  Поєднання даних для побудови
data = [trace_points, trace_depo] + path_traces
print('path: ', pi_)
```

```
    fig = go.Figure(data=data, layout=layout)
fig.show()
```

3.2.7. Збереження та форматування розв'язків моделі

Після того, як модель побудувала оптимальне рішення, нам потрібно його зберегти в файл, для його подальшого перетворення та побудови. В коді я це реалізував наступним чином:

```
#  Save data of the routes to pickle file
with open('data.pkl', 'wb') as f:
   pickle.dump(array_to_serialize, f, pickle.HIGHEST_PROTOCOL)
```

Тут я використовую вбудований модуль для серіалізації об'єктів – pickle.

Наступним етапом в побудові є перетворення отриманих результатів в коректний для подальшого використання вигляд. Саме для цього, вже після побудови маршруту, я використовую такі функції:

```
def get_separate_coordinates(path):
   with open(path, 'rb') as f:
      location_coords = pickle.load(f)
   drones = []
   tmp_arr = []

   for i in location_coords:
      if i[0] == 0 and len(tmp_arr) >= 2:
         tmp_arr.append(i[1])
         drones.append(tmp_arr)
         tmp_arr = []
      tmp_arr.append(i[1])

   return drones
```

Тут ми форматуємо отримане мережею рішення до вигляду, що може бути перетворено в точки на карті Mission Planner. Задля такого форматування для окремих дронів було описано метод, що для кожного БПЛА створює окремий відформатований файл з місією, що буде зрозуміла Mission Planner і яку в подальшому будемо використовувати для окремих дронів.

```
def generate_waypoints(coord_arr: Union[tuple, list]):
   for drone_num, drone_coords in enumerate(coord_arr, 1):
      with open(f"drone{drone_num}.waypoints", "w+", ) as f:
         f.write("QGC WPL 110\n")
```

```
            for index, coords in enumerate(drone_coords):
                if index == 0:
                    f.write(
                        f"{index}\t1\t0\t16\t0\t0\t0\t0\t{round(coords[0], 7)}\t{round(coords[1],
7)}\t100.000000\t1\n")
                else:
                    f.write(f"{index}\t0\t3\t16\t0\t0\t0\t0\t{round(coords[0], 7)}\t{round(coords[1],
7)}\t100.000000\t1\n")
```

3.3 Валідація результатів

Використання будь-якого алгоритму без етапу перевірки його доцільності і працездатності, з точки зору існуючих рішень не має сенсу. Задля перевірки отриманих результатів я буду використовувати вже відомі оптимальні розв'язки з бази даних [45]. Нижче буде приведено порівняння рішення моєї нейронної мережі з Augerat et al.(1995) розв'язком для 53 міст та 7 дронів.

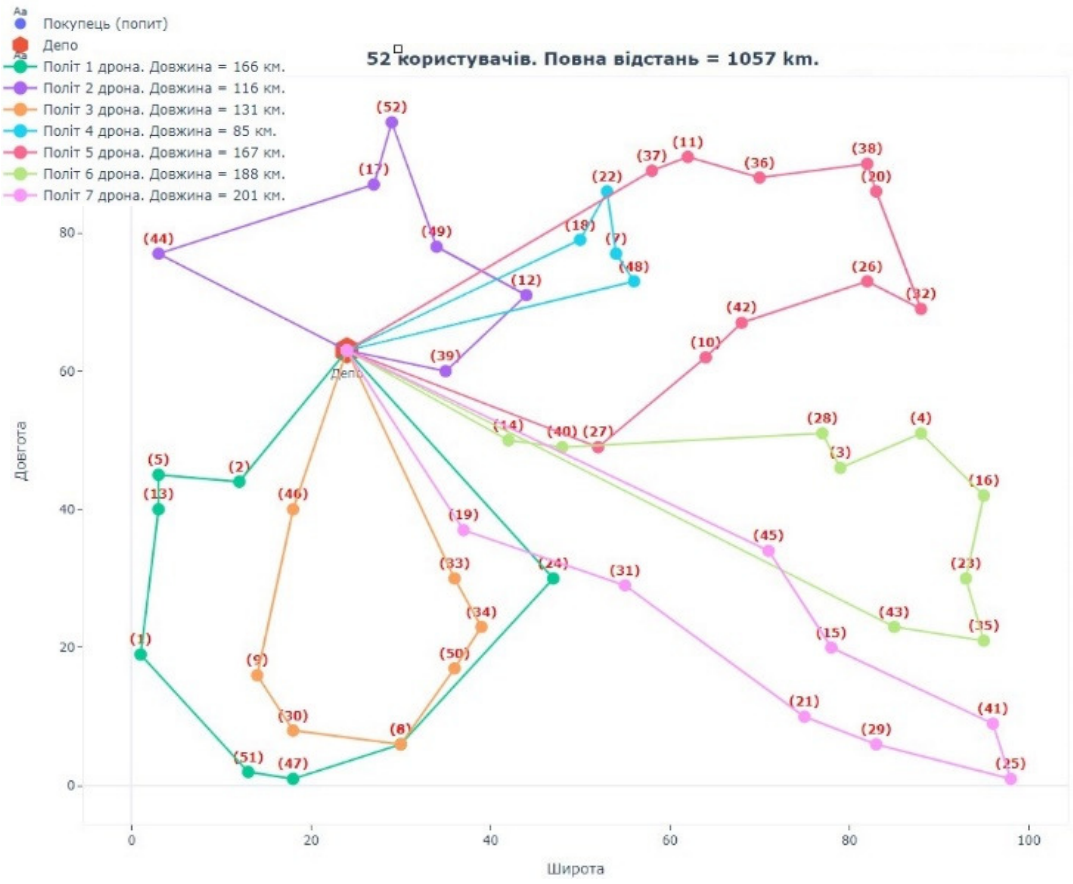


Рис. 3.2 Рішення нейронної мережі для 52 користувачів.

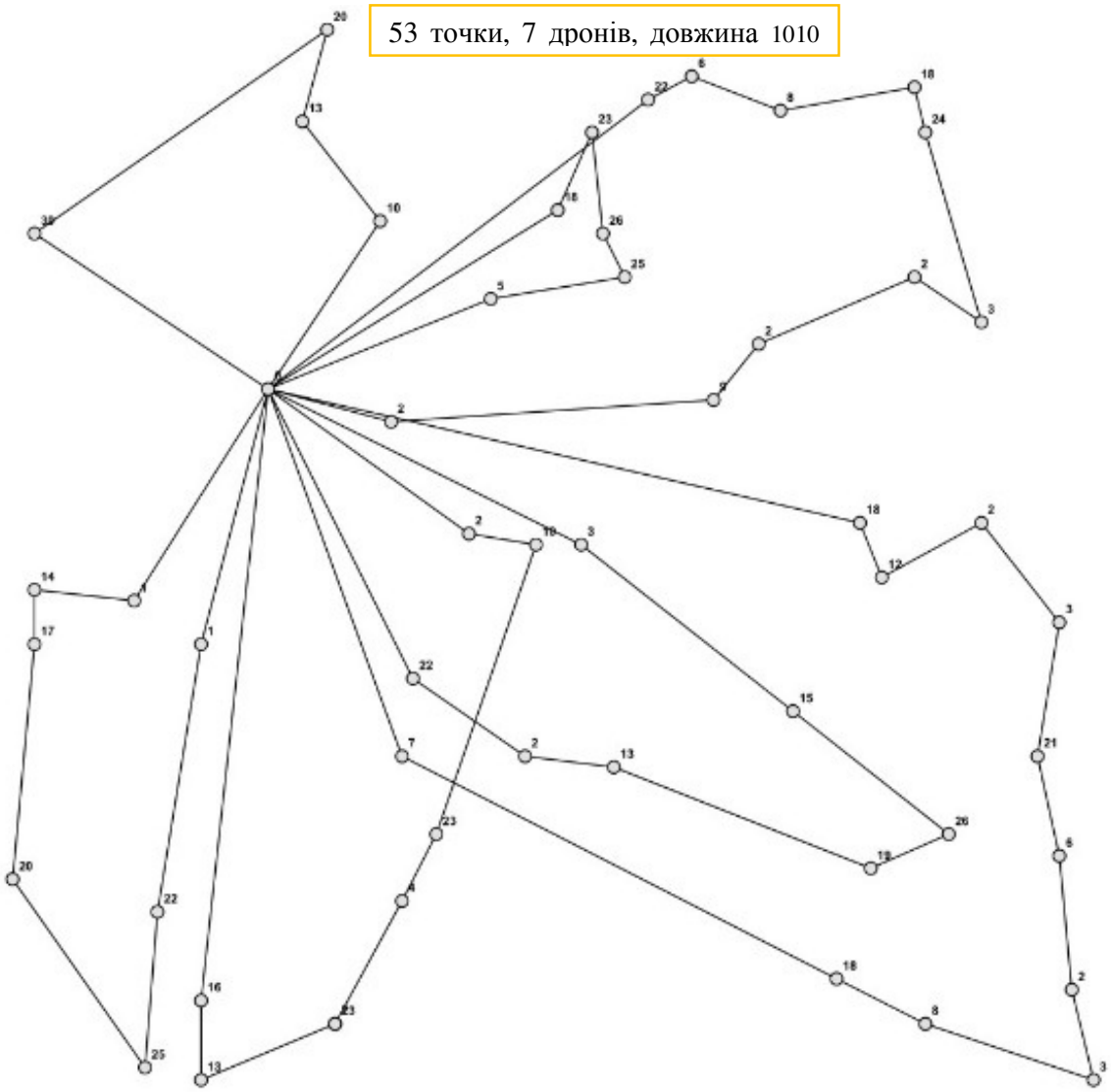


Рис. 3.3 Найкраще оптимальне рішення для 52 користувачів.

Виходячи з отриманого рішення бачимо, що розв'язок нейронної мережі є доволі близьким до оптимального, при тому, що навчання проходило лише для 19 епох та на одній відеокарті. Враховуючи це, можна проводити подальші обчислення з огляду на клас точності.

ВИСНОВОК ДО РОЗДІЛУ 3

В даному розділі були описані програмні реалізації алгоритму і частинні випадки для побудови та навчання головної моделі. Описано процес тренування мережі, роботу з даними. Детально описано налаштування гіперпараметрів і їх вплив на кінцевий результат.

Розглянуто приклад для 53 міст при 19 епохах (ітераціях) навчання. Результатом стало те, що мережа видала маршрут довжиною 1057 км. В порівнянні з найвигіднішим відомим розрахунком в 1010 км, це є доволі гарний показник по відношенню до кількості епох тренування, часу тренування та наявного обладнання.

Нейронна мережа тренувалась протягом двух годин на відеокарті Nvidia GTX 1050 Ti. Отримані результати вийшли досить оптимальними для подальшого моделювання їх на симуляторі Mission Planner з попередньо заготованими координатами реальної карти місцевості.

Тобто, запропоновані алгоритми виявилися роботоспроможними і можна розпочати серію їх більш складних випробувань.

РОЗДІЛ 4. ВІЗУАЛІЗАЦІЯ РЕЗУЛЬТАТІВ

За допомогою візуалізації підзадач можна краще зрозуміти процеси, що відбуваються “за лаштунками” програм. Головним чином це може допомогти зрозуміти можливості та, з іншого боку, недоліки алгоритму з огляду на фінальний результат. Під час написання програм та задля візуалізації підзадач було використано бібліотеку plotly, яку я попередньо описав в 1 та 2 розділі. Сукупну задачу буде змодельовано в середовищі Mission Planner з використанням його режиму планування маршрутів. Попередньо заготовані данні нейронної мережі будуть подаватись на вхід середовища у вигляді файлів місій .waypoints файлу. Сам процес буде змодельовано з використанням мапи Києва (як практичний приклад).

На прикладі такого моделювання ми максимально наближуємо ситуацію до реальної, які може дійсно виникнути на практиці ,оскільки це середовище використовується для повноцінного керування, як окремими дронами, так і цілими системами.

4.1 Побудова підзадач

До сценаріїв, що будуть описані нижче, я відношу окремі випадки, так звані, реальні ситуації, що можуть поставати перед підприємством.

Як результат, ці частинні випадки будуть уже враховані при реалізації сукупної, більш-менш повної задачі. Деякі можливі варіанти впровадження обмежень, та створення більш складних постановок будуть мною описані, але їх розгляд не є метою цієї роботи, а лише може розглядатись в контексті послідуючих робіт.

4.1.1. Задача 1. Розрахунок оптимального маршруту окремого дрону

Перш ніж досліджувати складні моделі та розв'язки задачі, потрібно спочатку розглянути простіші її постановки. В основі будь-якої проблеми маршрутизації лежить відома задача торговця, що подорожує.

В своїй попередній роботі [1-4], розділ 1.1 і 4.1, я детально описав найбільш прямий

метод її вирішення, безпосередньо перебираючи всі можливі варіанти. Оскільки вирішення такого роду задачі не має прямого практичного спрямування для системи дронів, то я відразу буду моделювати задачу близьку до реальної.

Прикладом реальної постановки може слугувати ситуація при доставці, скажімо піци, або розвезення покупок людей з онлайн-магазинів доставки (наприклад Aliexpress, Розетка тощо).

Припустимо у нас є 4 замовника і один дрон. У такому випадку оптимальний шлях знайдено:

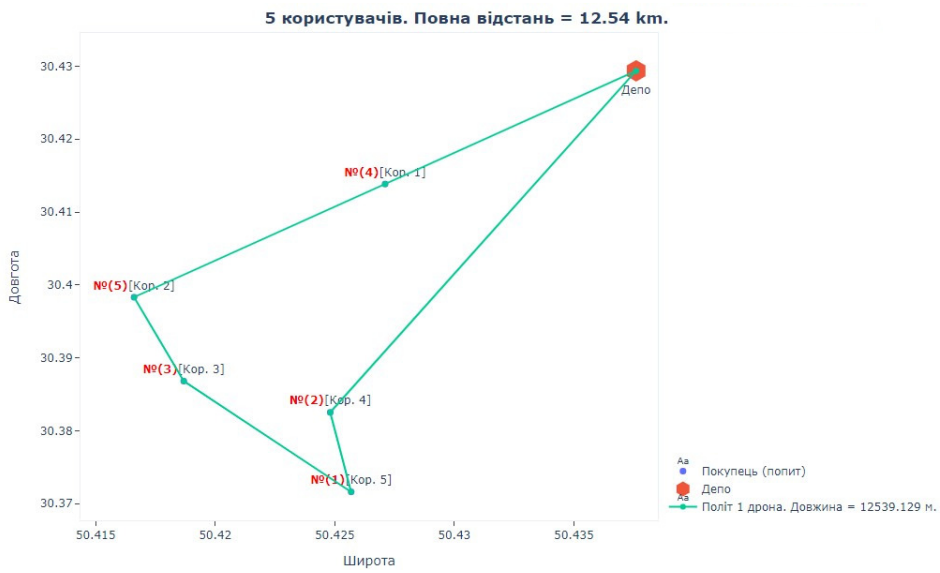


Рис 4.1 Приклад оптимальної роботи мережі для 1 дрона та 4 замовників

4.1.2. Задача 2. Мінімізація маршруту системи дронів

Не у всіх випадках можливо однаково добре знайти оптимальний, з точки зору часу, відстаней та вантажних можливостей маршрут. За таких

ситуацій потрібно збільшувати кількість дронів. Але збільшення кількості не обов'язково буде вести до більш оптимальних рішень з точки зору бізнесу. Для того, щоб мережа видавала рішення з огляду на кількість дронів, був використаний метод так званого "відбору" або (sampling) його суть полягає в тому щоб створити певну кількість розв’язків моделі і серед цієї кількості при тих самих вірогіднісних параметрах обрахувати краще рішення.

Суть методу полягає в тому, що ми серед певного розміру групи рішень вибираємо найбільш вигідне вже жадібним методом (TopKSampler), з огляду на вірогідність поліпшення baseline рішення.

На рисунку 4.2 нижче можна побачити приклад розв’язку такого роду проблеми для 3 дронів та 19 користувачів.

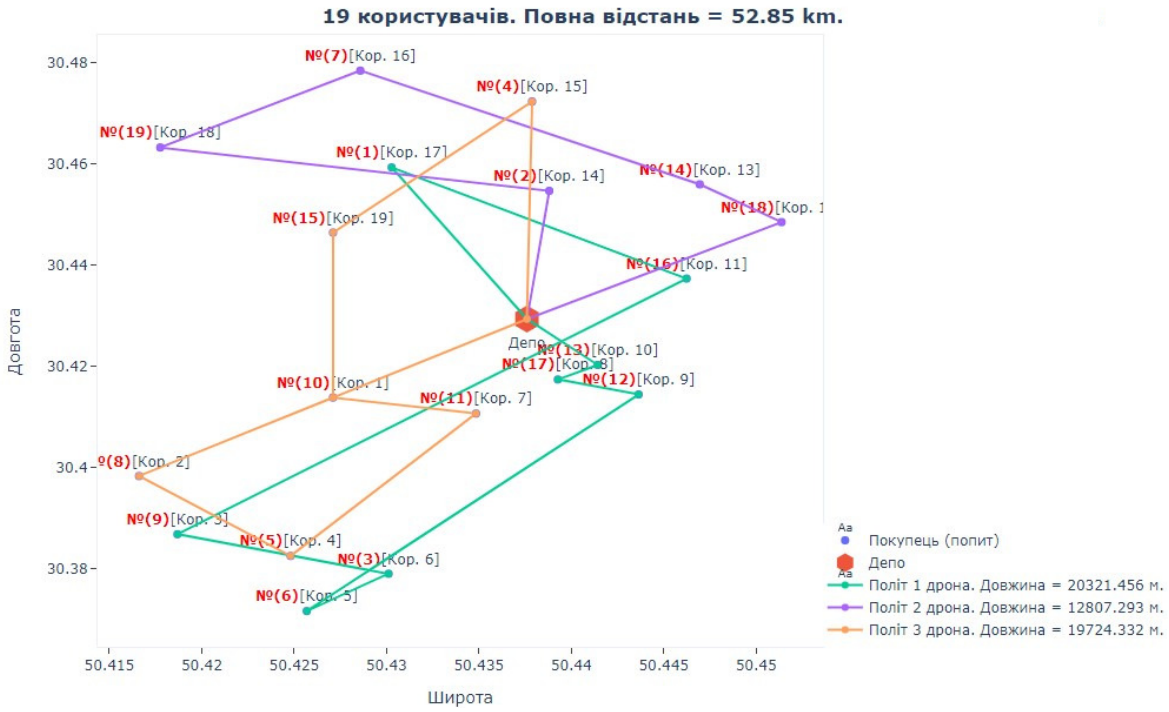


Рис. 4.2. Приклад оптимізації маршруту з урахуванням системи дронів

4.1.3. Задача 3. Мінімізація кількості БПЛА

В роботі було також враховано мінімізацію кількості БПЛА. Програмно це було реалізовано за допомогою зменшення окремих графів-підзадач, котрі під кінець кожної епохи (стадії навчання) були додані до baseline-у.

В реаліях бізнесу неправильне використання наявного обладнання завжди веде до збільшення витрат, тому так важливо оптимізувати цей параметр. Щоб наочно показати необхідність мінімізації цього критерію на рисунку нижче продемонстровано неоптимальне вирішення задачі оптимізації кількості дронів з багатьма одночасно задіяними БПЛА. Це доволі складно зрозуміти з вирішення задачі для 19 точок, але на прикладі з 50 точок для 16-ти дронів це вже має доволі значний вплив.

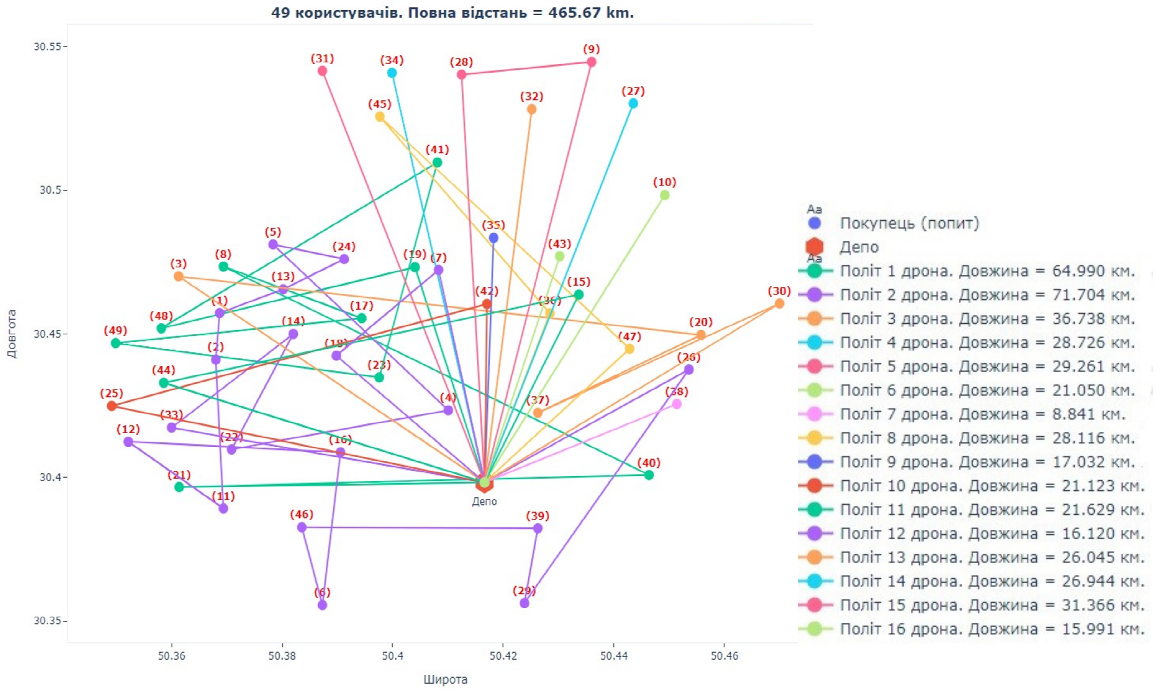


Рис. 4.3. Візуальне представлення рішення для 50 точок, з використанням 16 дронів

Як бачимо з рисунку, такого роду використання ресурсів не є оптимальним, оскільки можна бачити багато маршрутів що ведуть тільки до однієї точки на карті, і його можна покращити. Прикладом правильної оптимізації може слугувати ілюстрація нижче:

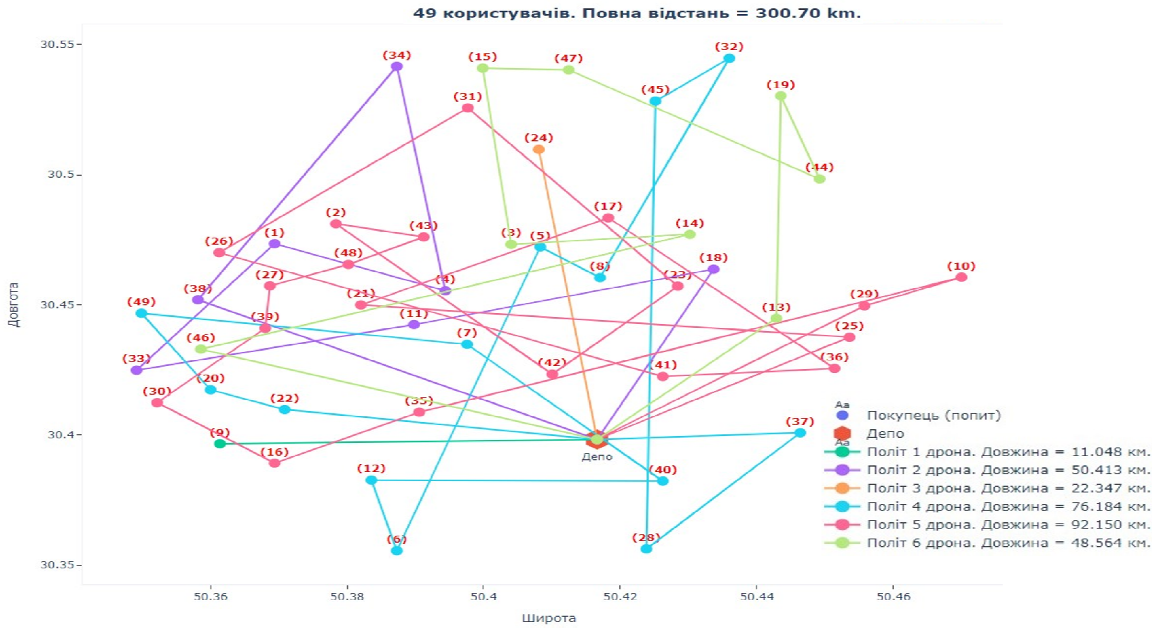


Рис. 4.4. Оптимальний розв’язок для 50 точок, з використанням 6 дронів

4.1.4. Задача 4. Автоматичний вибір висотних коридорів

Під час побудови я вирішив, що також доцільно буде враховувати висотні коридори в тих випадках, коли дрони можуть знаходитись в одному і тому ж часовому вікні. Сама по собі проблема CVRPTWs (Capacitated Vehicle Routing Problem With Time Windows) є доволі складною з огляду її математичного формулювання та наступної реалізації в нейронній мережі. Такого роду задачі виходять за рамки цієї роботи і можуть бути розглянуті в подальших роботах.

В даній роботі буде лише враховано тон момент, що дрони між собою не будуть зіштовхуватись в однакові моменти часу. В послідуючому ці данні у вигляді кортежу типу (50.3838838, 30.494949, 15) будуть зберігатись за допомогою вбудованої бібліотеки серіалізації pickle і можуть передаватись на наземну станцію керування в керованому (Guided) режимі, тим самим передаючи вказівки дрону стосовно напряму руху.

Прикладом використання висотних коридорів може слугувати розрахункове зображення 4.5:

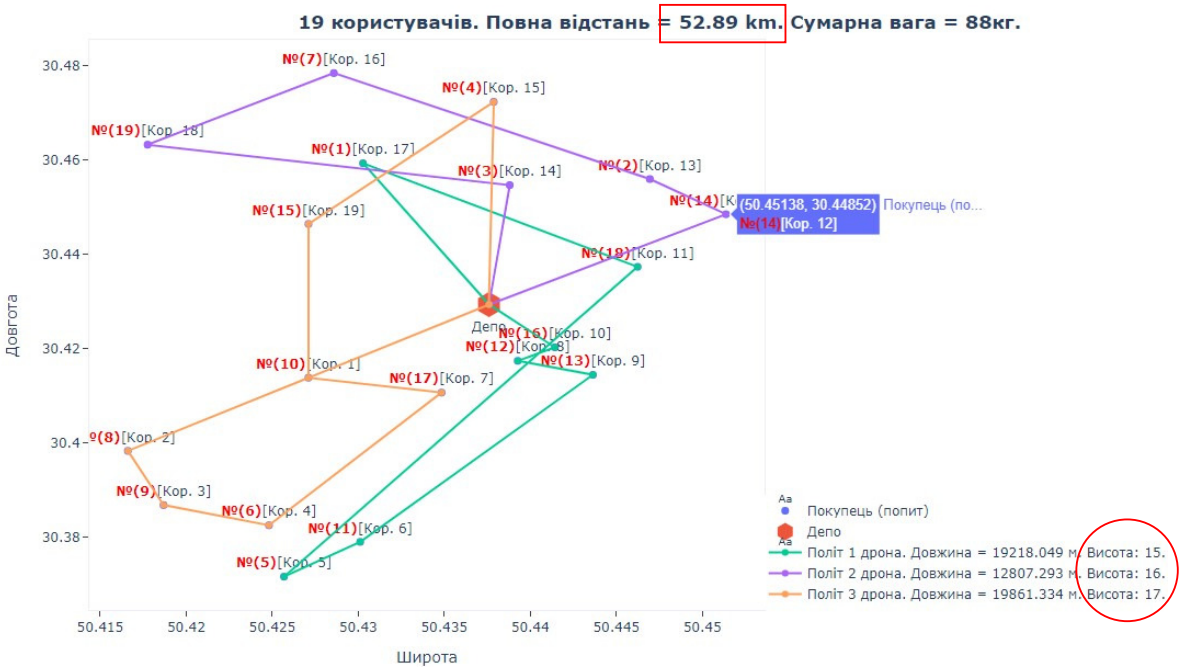


Рис. 4.5. Візуалізація впливу висоти на загальну довжину маршруту

Як бачимо з малюнку, хоча ми і вибираємо різні коридори для БПЛА, але довжина маршруту значно збільшився, оскільки ми враховуємо, що для доставки вантажу треба буде спускатись і підійматись.

4.1.5. Задача 5: Врахування ваги вантажу та часу доставки

До цього моменту було отримано доволі гарні результати з огляду на довжину маршруту та на його доцільність з точки зору кількості дронів, необхідних для доставки вантажу, але його можна покращити, змінивши ще декілька параметрів навчання, що я і знайшов експериментальним чином. Одним із варіантів покращення існуючого рішення, з огляду на мою головну задачу, є оптимізація кількості вантажу з урахуванням найближчих сусідів,

Використовуючи для кожної з точок додатковий параметр – вагу вантажу, що вказана в таблиці нижче, можна отримати розв’язки мережі близькі до реальних.

Таблиця 4.1

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
вага	0	2	5	5	2	6	7	2	4	2	4	3	4	5	6	3	5	5	6	2

Результатом роботи мережі є наступний маршрут:

path: [0, 2, 7, 6, 5, 3, 9, 17, 19, 0, 8, 10, 11, 12, 13, 18, 14, 0, 4, 1, 16, 15, 0]

На рисунку 4.6 траекторії виглядають наступним чином:

Як бачимо з рисунку, мережа підібрала оптимальну кількість дронів, з огляду на вантажопідйомність в 30кг. а також загальний шлях збільшився на декілька кілометрів.

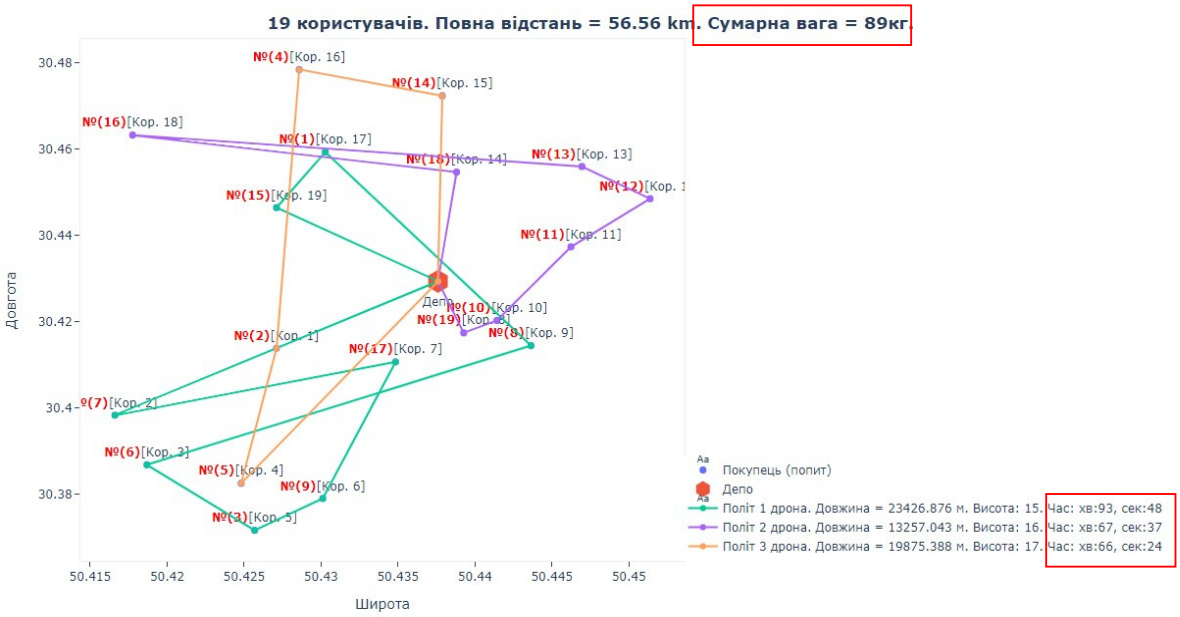


Рис.4.6 Приклад роботи мережі з урахуванням фактору ваги та часу

4.1.6. Можливі проблеми при використанні нейронної мережі

Під час візуалізації попереднього маршруту, я використав мережу, натреновану вирішувати задачі маршрутизації для 20 замовників одночасно. Якщо для тієї самої кількості міст використати іншу мережу, скажімо для 50 точок, вона буде вироджуватись і давати неоптимальний результат. Приклад такого використання буде нижче (Рис. 4.7).

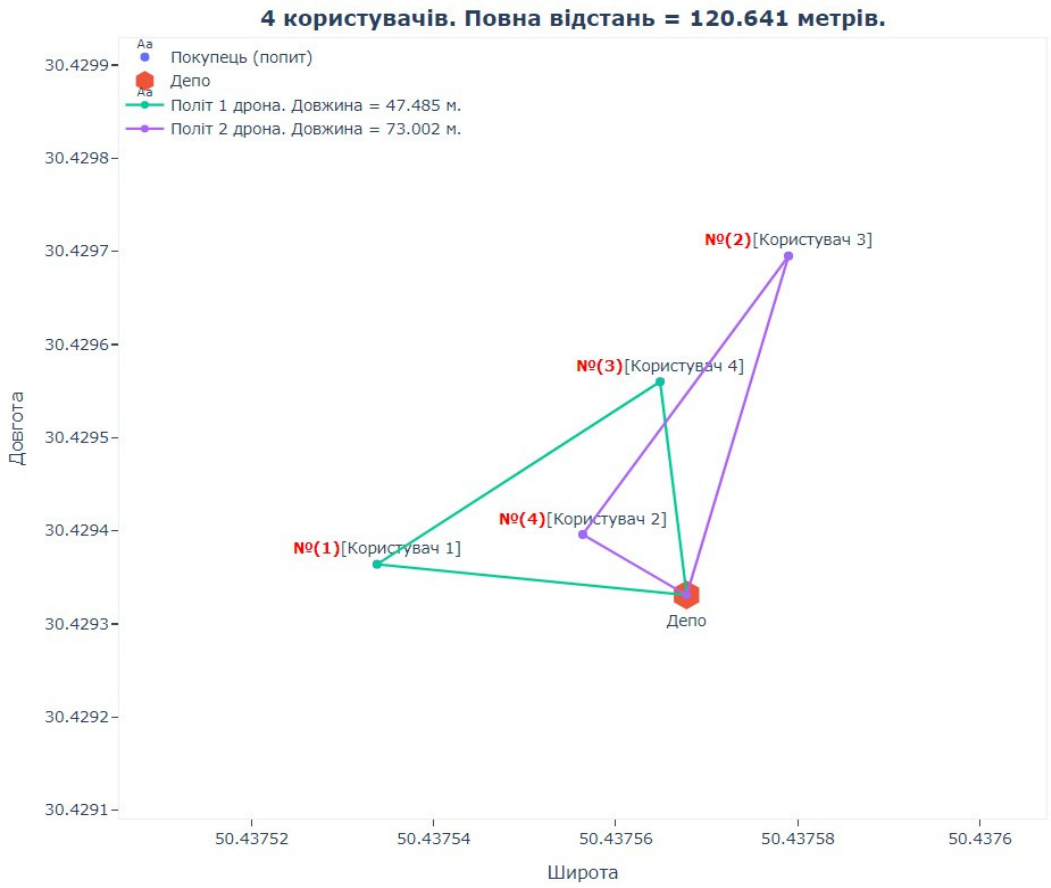


Рис. 4.7. Приклад використання мережі, тренованої на 50 точок для 5

Як можна спостерігати з рисунку, тут вже мережа використовує два дрони і сукупний маршрут виходить майже вдвічі довшим. Інтуїтивно зрозуміло, що використати 1 дрон було б оптимальніше. Тому бажано, заради отримання оптимальних результатів, використовувати натреновані мережі з різницею в ± 10 точок. Ще одним недоліком використання такого роду навчених мереж, скажімо на замовників з вагою, що значно перевищує допустимі межі перевезення дроном, є те, що система буде вироджуватись і

будувати рішення там, де його немає зовсім. Прикладом такого роду проблеми є рисунок нижче (4.8). На ньому можна побачити, що мережа побудувала шлях через замовника з вагою 40, при допустимій 30 для одного дрона.

Щоб цього уникнути можна просто програмно заборонити замовляти

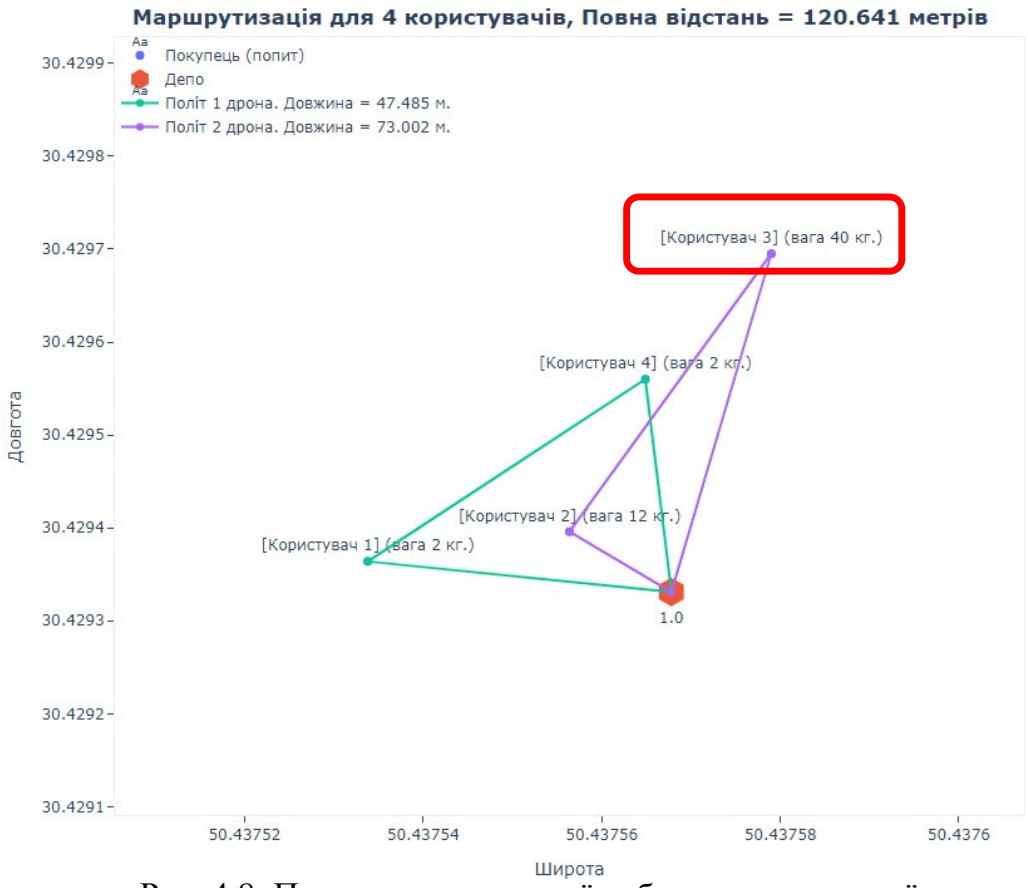


Рис. 4.8. Приклад некоректної роботи для завеликої ваги

вантажі більші за ті, що може донести дрон, або в такому випадку пропонувати замовити автомобіль, або інший вид доставки. Для більш чіткої картини в подальшому я буду уникати цих моментів, щоб розкрити результат роботи мережі “у всій красі”. Проблему з використанням різних мереж для різних задач можна вирішити у вигляді бази даних з мережами і допустимих відхиленнях підрахунків для маршрутів, таким чином можна за постійний час видавати доволі оптимальні рішення для користувачів логістичного сервісу доставки.

Ще однією з можливих проблем при розрахунках, з якою мені довелось зіштовхнутися, було те, що на цілочисловому масштабі нейронна мережа працювала доволі добре – це можна спостерігати з огляду на валідаційну вибірку (тестові дані з розв'язками інших алгоритмів), де рішення мережі було дуже близьким до оптимального. Але коли довелося обраховувати реальні маршрути, то фактор використання дробових чисел також впливає на підрахунки довжини як окремих маршрутів, так і загального в рамках системи.

В кінцевому випадку ми можемо спостерігати, що не всі маршрути є доволі точними, навіть з огляду на те, що є перехрещення точок в центрі маршруту. Ці перехрещення не завжди означають поганий маршрут, інколи це виправдано тим, що до користувача з меншою вагою буде доцільніше прилетіти в першу чергу, а потім вже доставити більш важкий вантаж. Але, як показує практика, прибравши такого роду перехрещення маршрут зазвичай стає коротшим. Ще одним із можливих причин такого роду поведінки було те, що я використовував лише наявне в мене обладнання та обраховував вагові коефіцієнти мережі протягом лише 4 годин, що не є доволі репрезентативним в рамках використання мережі. На сьогодні є безліч досліджень [37-39], що показують необхідність обрахунків відразу на декількох відеокартах та протягом декількох днів, що в моєму випадку не було самоціллю роботи.

Саме тому, для спрощення розрахунків і лише в цілях демонстрації працездатності моєї мережі, я свідомо пішов на такого роду спрощення.

4.2. Реалізація сукупної задачі

Для початку я попередньо спланував маршрут на карті в режимі створення місії з використанням 20 точок, враховуючи депо (початок):

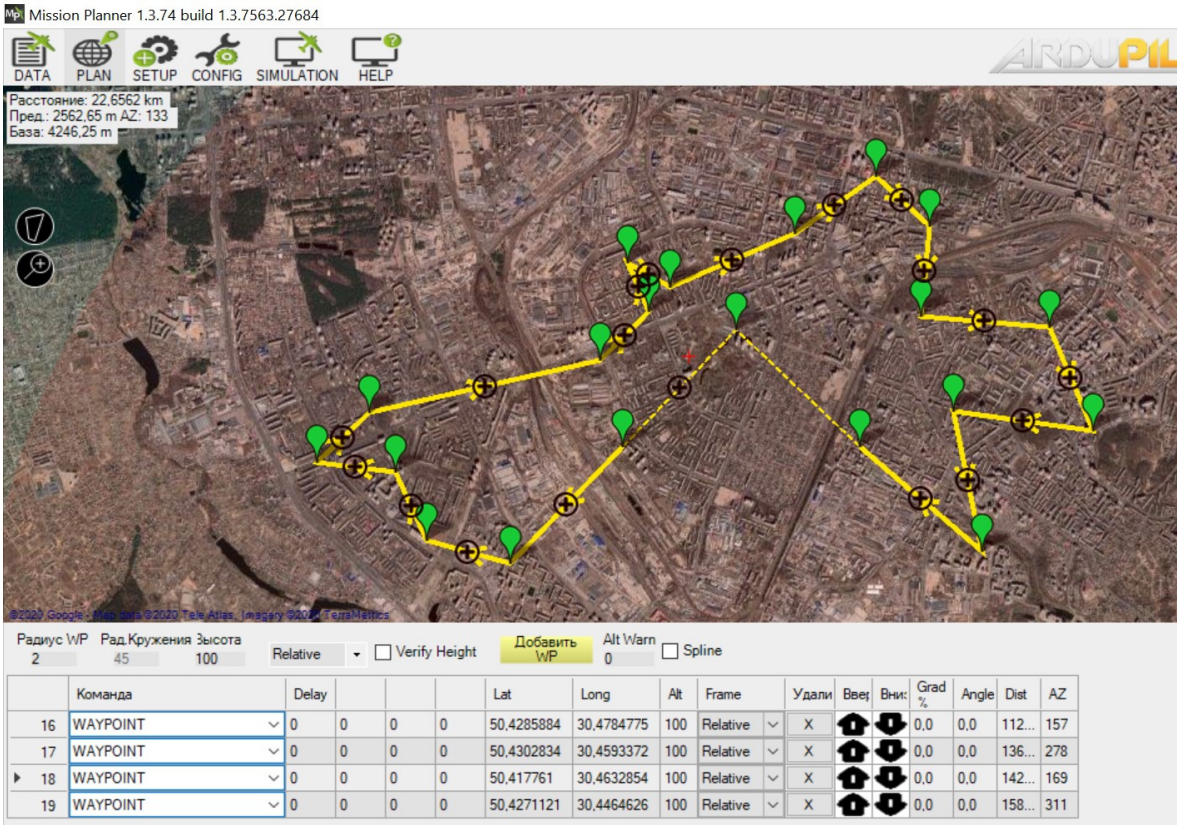


Рис. 4.9. Формування вибірки замовників служби доставки в режимі Plan (візуалізовано за допомогою Mission Planner)

Наступним етапом є використання даних про вагу вантажу, що в моєму випадку буде просто вибрано випадковим чином. Кожному окремому замовнику ваги вантажу, що необхідно доставити БПЛА. Вагова таблиця виглядає наступним чином: Таблиця 4.2

№	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
вага	0	2	5	5	2	6	7	2	9	7	4	3	4	5	6	3	5	5	6	2

Враховуючи той факт, що під час побудови рішення було використано модель для БПЛА, що може нести до 30 кг. ваги, очікувано буде побачити оптимальний політ 3 дронів (рис 4.10 - 4.13)

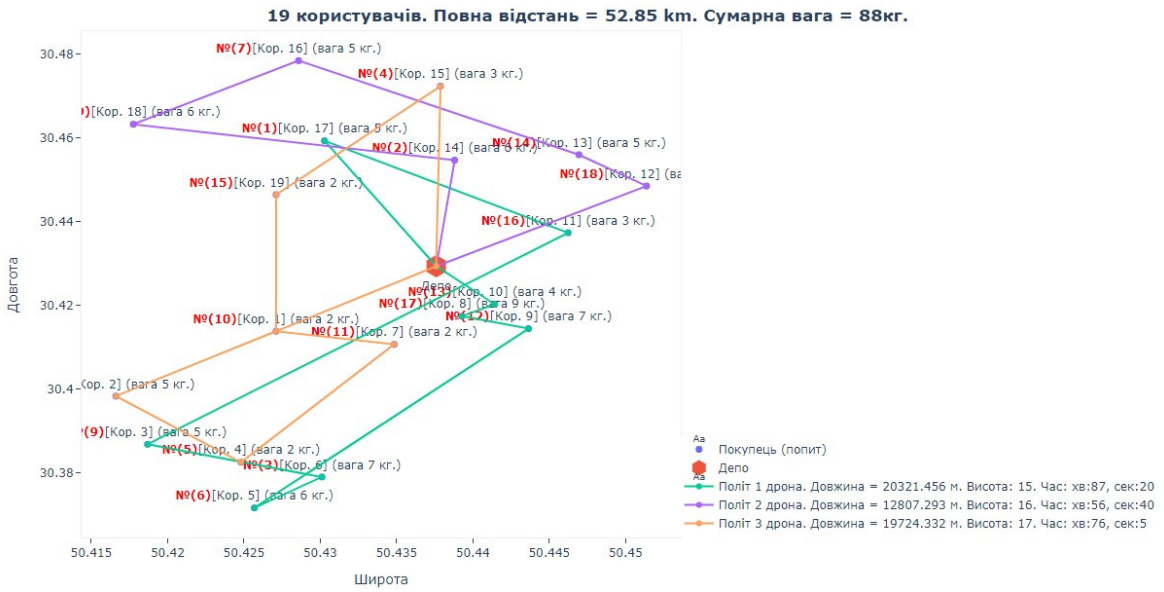
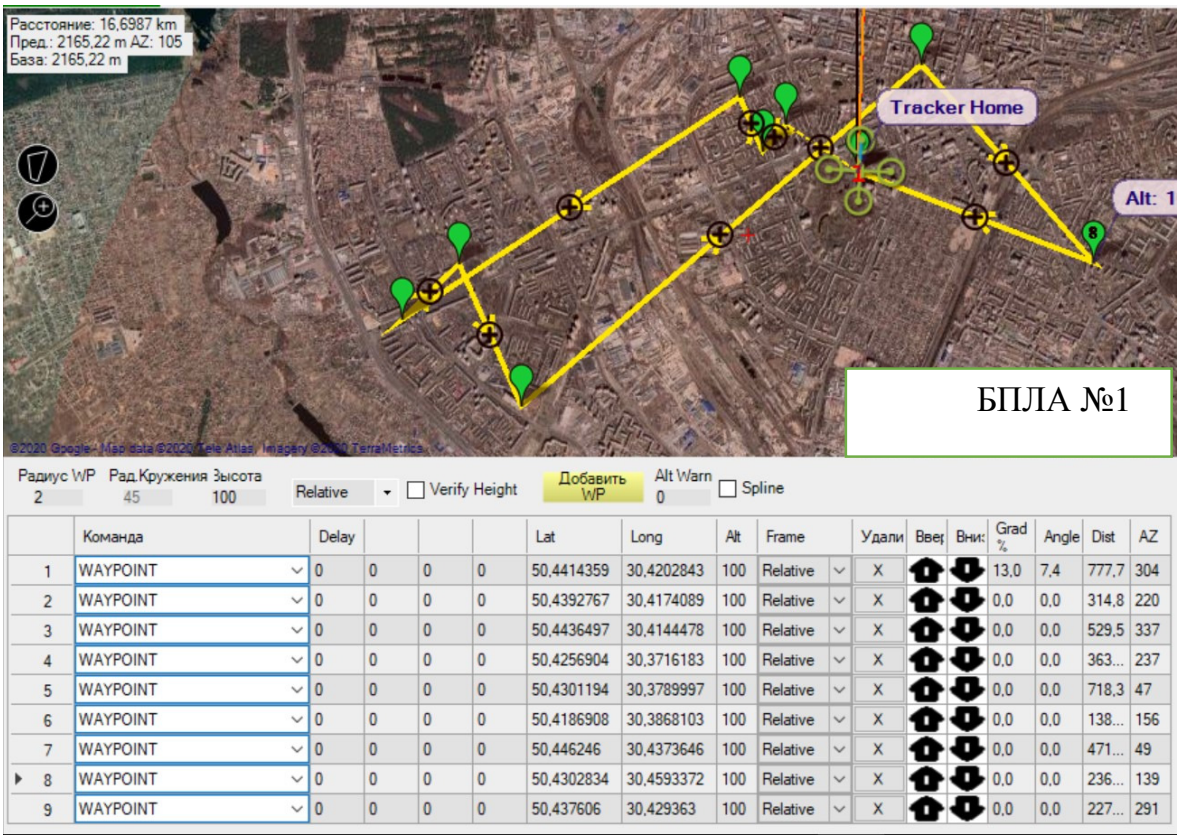


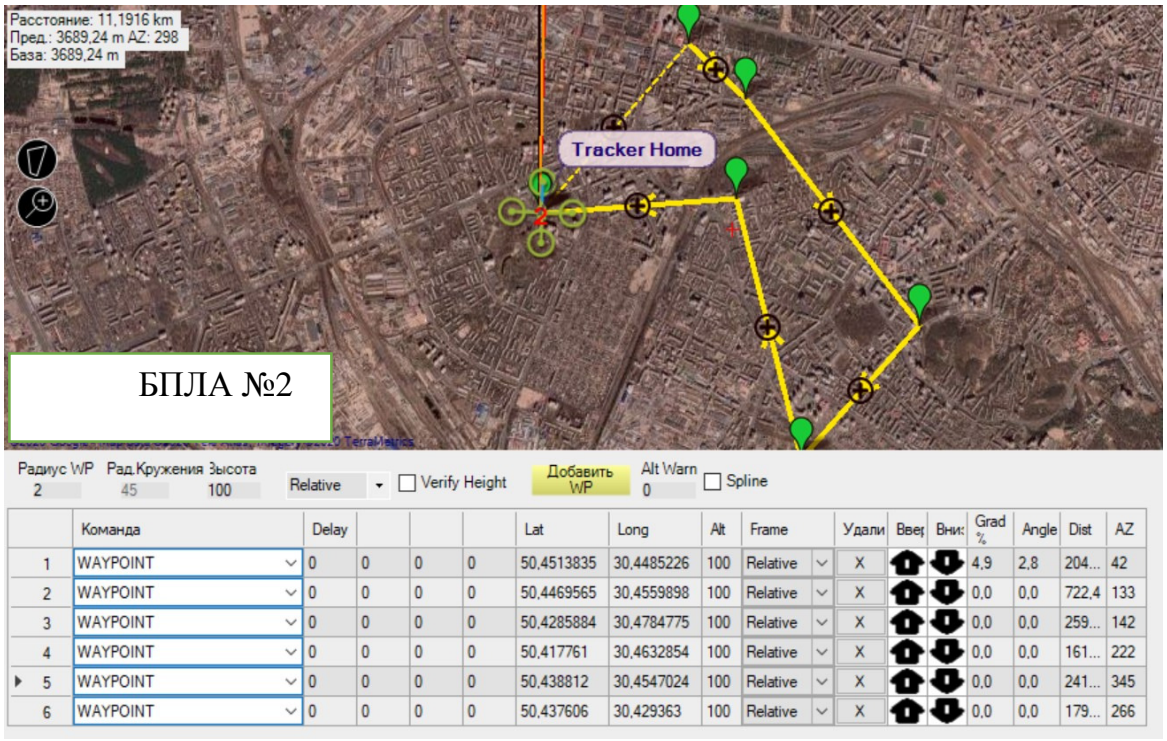
Рис. 4.10. Схематичне зображення польоту 3 БПЛА



	Команда	Delay				Lat	Long	Alt	Frame	Удали	Ввер	Вни:	Grad %	Angle	Dist	AZ
1	WAYPOINT	0	0	0	0	50,4414359	30,4202843	100	Relative	X			13,0	7,4	777,7	304
2	WAYPOINT	0	0	0	0	50,4392767	30,4174089	100	Relative	X			0,0	0,0	314,8	220
3	WAYPOINT	0	0	0	0	50,4436497	30,4144478	100	Relative	X			0,0	0,0	529,5	337
4	WAYPOINT	0	0	0	0	50,4256904	30,3716183	100	Relative	X			0,0	0,0	363...	237
5	WAYPOINT	0	0	0	0	50,4301194	30,3789997	100	Relative	X			0,0	0,0	718,3	47
6	WAYPOINT	0	0	0	0	50,4186908	30,3868103	100	Relative	X			0,0	0,0	138...	156
7	WAYPOINT	0	0	0	0	50,446246	30,4373646	100	Relative	X			0,0	0,0	471...	49
8	WAYPOINT	0	0	0	0	50,4302834	30,4593372	100	Relative	X			0,0	0,0	236...	139
9	WAYPOINT	0	0	0	0	50,437606	30,429363	100	Relative	X			0,0	0,0	227...	291

Рис. 4.11. Завантаження маршруту для першого БПЛА

Нижче на рисунках буде приведено побудову маршрутів для окремих БПЛА за рахунок завантаження розрахованих відстаней через .waypoint файл.



	Команда	Delay				Lat	Long	Alt	Frame	Удали	Ввер	Вни:	Grad %	Angle	Dist	AZ
1	WAYPOINT	0	0	0	0	50,4513835	30,4485226	100	Relative	X			4,9	2,8	204...	42
2	WAYPOINT	0	0	0	0	50,4469565	30,4559898	100	Relative	X			0,0	0,0	722,4	133
3	WAYPOINT	0	0	0	0	50,4285884	30,4784775	100	Relative	X			0,0	0,0	259...	142
4	WAYPOINT	0	0	0	0	50,417761	30,4632854	100	Relative	X			0,0	0,0	161...	222
5	WAYPOINT	0	0	0	0	50,438812	30,4547024	100	Relative	X			0,0	0,0	241...	345
6	WAYPOINT	0	0	0	0	50,437606	30,429363	100	Relative	X			0,0	0,0	179...	266

Рис. 4.12. Завантаження маршруту для другого БПЛА

	Команда	Delay				Lat	Long	Alt	Frame	Удали	Ввер	Вни:	Grad %	Angle	Dist	AZ
1	WAYPOINT	0	0	0	0	50,4248155	30,3825188	100	Relative	X			6,9	4,0	144...	309
2	WAYPOINT	0	0	0	0	50,4348486	30,4106712	100	Relative	X			0,0	0,0	228...	61
3	WAYPOINT	0	0	0	0	50,4271092	30,4138373	100	Relative	X			0,0	0,0	889,3	165
4	WAYPOINT	0	0	0	0	50,4271121	30,4464626	100	Relative	X			0,0	0,0	231...	90
5	WAYPOINT	0	0	0	0	50,4378827	30,4723835	100	Relative	X			0,0	0,0	219...	57
6	WAYPOINT	0	0	0	0	50,437606	30,429363	100	Relative	X			0,0	0,0	304...	269

Рис. 4.13. Завантаження маршруту третього БПЛА

Завантаживши необхідні данні та присвоївши їх кожному з дронів, наступним етапом є запуск та налаштування польоту БПЛА. В процес налаштування головним чином входить постановка БПЛА в режим

готовності, підняття на певну висоту польоту, що може бути вказана вручну, або програмно за допомогою Python-скрипта.

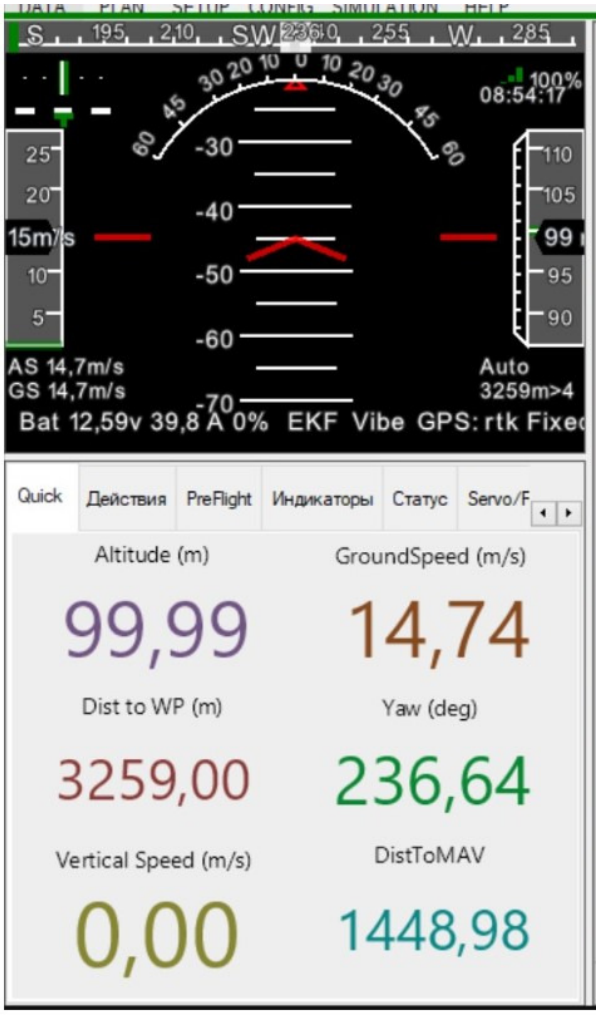


Рис. 4.14. Інформаційна панель наземної керуючої станції в Mission Planner. Приклад для польоту БПЛА на висоті 100м та за швидкості 15м/с

Останнім етапом є запуск місії для кожного окремого дрону. Всі дрони будуть летіти в автоматичному режимі та після виконання місії повернуться додому. Для простоти візуалізації я не враховував фактор використання акумуляторної батареї, що також є важливим чинником впливу на ефективність системи в цілому.

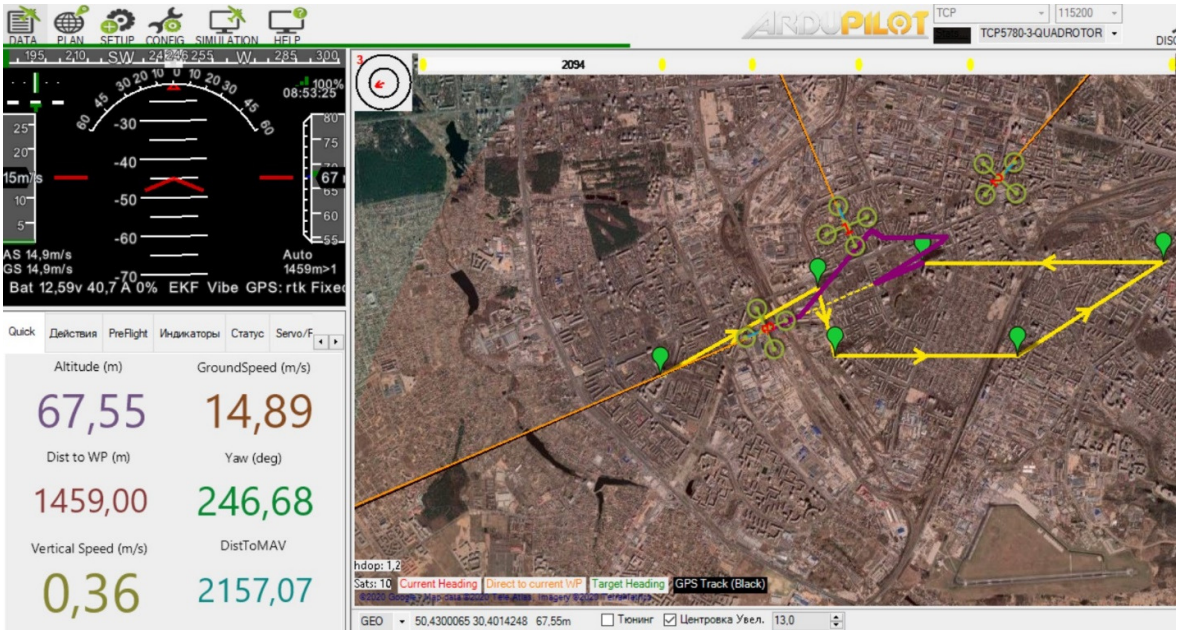


Рис. 4.15 Симуляція в Mission Planner польоту системи дронів за маршрутами нейронної мережі

ВИСНОВОК ДО РОЗДІЛУ 4

Даний розділ завершує дослідження автора. В ньому алгоритм, описаний у Розділі 3, випробувано на кількох задачах:

1. Маршрут окремого дрона
2. Механізм мінімізації маршруту для системи дронів
3. Врахування фактору ваги вантажу
4. Вибір висотних коридорів
5. Мінімізація кількості БПЛА в складі системи доставки

На основі даних підзадач була складена та проілюстрована сукупна задача на прикладі, близькому до реального, з використанням 3 дронів. Результати ілюстровані на 9 рисунках. Можна зробити висновок, що програма і алгоритми дають доволі правдоподібні результати.

Безумовно, на практиці можуть зустрітися і більш складні задачі. Так, у нас не враховувався фактор акумуляторної батареї при побудовах маршрутів. Описано припущення, на які свідомо доводилось йти в процесі написання роботи. Розглянуто можливі проблеми при використанні нейронних мереж та потенційні проблеми при візуалізації такого роду розв’язків.

РОЗДІЛ 5. ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

В наші дні питання використання безпілотних літальних апаратів, в більшості своїй дронів, все частіше постають на порядку денному. Дрон - безпілотний літальний апарат (БПЛА) військового чи цивільного призначення, різновид військового робота; в ширшому сенсі — мобільний, автономний апарат, запрограмований на виконання якихось завдань (наприклад, автономні системи, створені для польоту, розроблені для виконання місій, потенційно небезпечних для людини).

Існують десятки різних типів безпілотних літальних апаратів; вони в основному діляться на дві категорії: ті, які використовуються для розвідки й спостереження, та ті, що мають на озброєнні ракети й бомби. Використання дронів швидко зростає в останні роки, тому що, на відміну від пілотованих літаків, вони можуть перебувати в повітрі протягом багатьох годин.

Попит на безпілотну техніку зумовлений її практичністю не тільки у військовій сфері, а й корисністю в інших сферах людського існування. Зокрема, у країнах з великими агрохолиднгами (Україна, Польща, Китай тощо) активно запроваджується використання дронів для дослідження та аналізу стану землі, оцінки врожаїв та збитків, і оптимізації роботи сільськогосподарської техніки.

У сільських районах безпілотні літальні апарати призначені для заміни дорожніх поставок з метою подолання інфраструктурних проблем; оскільки безпілотники споживають значно менше і, отже, менше впливають на навколишнє середовище, їх оцінка повного життєвого циклу все ж повинна бути піддана комплексному аналізу, щоб зрозуміти їх вплив на навколишнє середовище.

Специфікація безпілотника та його елементів

Таблиця 5.1

Діагональна колісна база	1133 мм
Розміри	1 668 мм x 1518 мм x 727 мм (з гвинтами, каркасними важелями і розгорнуте кріплення GPS (Включаючи шасі)) 437 мм x 402 мм x 553 мм з гвинтами, каркасними важелями і складаний GPS-приймач (За винятком шасі)
Вага (з шістьма TB47S акумуляторами)	9,5 кг
Вага (з шістьма TB48S акумуляторами)	10 кг
Максимальна злітна вага	15,5 кг (дрон разом з товарами)
Точність парирування (P-GPS)	Вертикальна: ± 0,5 м, Горизонтальна: ± 1,5 м
Максимальний опір вітрі	8 м $с^{-1}$
Максимальна швидкість підйому	5 м $с^{-1}$
Максимальна швидкість спуску	3 м $с^{-1}$
Максимальна службова висота	гвинти 2170R: 2500 м

над рівнем моря	2195 пропелери 4500 м
Максимальна швидкість	65 км / г $^{-1}$ (без вітру)
Час паузи (З шістьма батареями TB47S)	Ніякого корисного навантаження: 32 хв, 6 кг Вантажопідйомність: 16 хв
Час паузи (З шістьма батареями TB48S)	Ніякого корисного навантаження: 38 хв, 5,5 кг Вантажопідйомність: 18 хвилин
Система управління польотом	A3 Pro
Рухова система Модель двигуна:	DJI 6010
Модель пропелера:	DJI 2170R
Висувне шасі	Стандартне
Робоча температура	від 10 С до 40 С
Використання енергії з розрахунку на кілометр	0.093 MJ

Джерела впливу системи на довкілля

Я не можу обійти питання екологічної безпеки, в разі використання в своїй системі одного з можливих варіантів безпілотного літального апарату. Для більш комплексної оцінки в роботі було враховано відразу декілька факторів ризику при виробництві БПЛА. При цьому було використано так званий метод оцінки впливу циклу або МОВЦ.

Метод оцінки впливу циклу (МОВЦ) використовується для перетворення даних життєвого циклу в дані про вплив на навколишнє середовище. Спостережувані результати в попередніх дослідженнях, показують, що система БПЛА з використанням безпілотної доставки є одним з найбільш екологічно чистих варіантів транспортування по широкому спектру напрямів.

Виробництво деталей сприяє значному впливу на навколишнє середовище, в той час як сама експлуатація безпілотних літальних апаратів показує найменший ефект.

В своїй роботі я досконало розглянув всі можливі чинники, що можуть впливати на екосистему та на людину, як її частину. Було розглянуто основні фактори, що сприяють глобальному потеплінню, абіотичні виснаження (абіотичні елементи і копалини), підкислення повітря, категорії впливу на евтрофікацію, виснаження озонового шару і фотохімічні утворення були спричинені в основному видобутоком вугілля і виробленням електроенергії.

Однак вуглецеве волокно і акамуляторна батарея є основними джерелами інших категорій впливу, до яких відносяться токсичністі: водну екотоксичність, морську екотоксичність і екотоксичність земних екосистем.

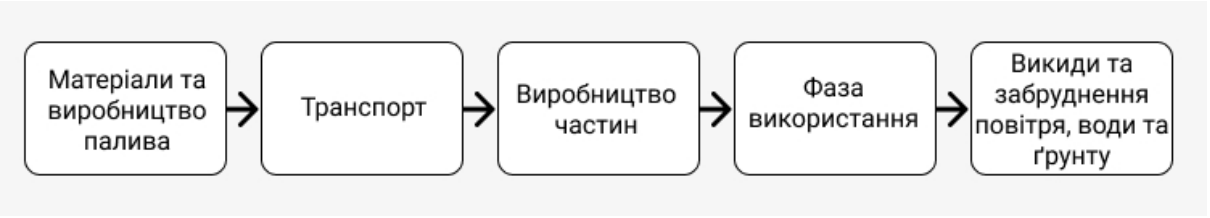


Рис. 5.1 Процес формування викидів під час виробництва БПЛА

Джерела впливу на людину і на екосистему

Оцінка впливу на довкілля (ОВД), або оцінка впливу на навколишнє середовище (ОВНС) призначена для виявлення характеру, інтенсивності і ступеня небезпеки впливу будь-якого виду планованої господарської діяльності на стан довкілля і здоров'я населення.

Вплив на довкілля — будь-які наслідки планованої діяльності для довкілля, в тому числі наслідки для безпечності життєдіяльності людей та їхнього здоров'я, флори, фауни, біорізноманіття, ґрунту, повітря, води, клімату, ландшафту, природних територій та об'єктів, історичних пам'яток та інших матеріальних об'єктів чи для сукупності цих факторів, а також наслідки для об'єктів культурної спадщини чи соціально-економічних умов, які є результатом зміни цих факторів;

Проведення ОВД майбутньої господарської, і іншої діяльності на довкілля сприяє ухваленню екологічно грамотного управлінського рішення про реалізацію наміченої господарської і іншої діяльності за допомогою визначення можливих несприятливих дій оцінки екологічних наслідків, обліку громадської думки, розробки заходів зі зменшення і запобігання дій.

Методологія ОВД дістала своє визнання майже в усіх розвинених країнах. У червні в 1988 р. була введена в дію Директива ЄС № 337/85 «Оцінка впливу деяких державних і приватних проектів господарської діяльності на навколишнє середовище». Відповідно до неї, для країн — членів ЄС обов'язковим є проведення ОВД до видачі дозволу на здійснення всіх великих проектів, що можуть спричинити негативний вплив на навколишнє середовище.

Раніше в Україні таку роль відігравала «Оцінка впливу на навколишнє середовище» (ОВНС), відповідно до законів «Про охорону навколишнього природного середовища», «Про екологічну експертизу» та Державних будівельних норм України ДБН А.2.2-1-95 «Склад і зміст матеріалів оцінки впливу на навколишнє середовище (ОВНС) при проектуванні і будівництві підприємств, будівель і споруд».

З 18 грудня 2017 року набув чинності Закон України «Про оцінку впливу на довкілля», який фактично скасовує дію закону України «Про екологічну експертизу» та вводить новий, більш сучасний та європейський порядок проведення оцінки впливу на довкілля. Без наявності висновку про

оцінку впливу на довкілля суб'єкт господарювання не має права здійснювати заплановану діяльність. Тож беручи до уваги всі вищезгадані моменти я проаналізував можливі впливи на екологію моєї системи доставки за допомою дронів.

Дане дослідження спрямоване на визначення "гарячих точок" по всій території повного життєвого циклу системи безпілотників.

Основна увага в цьому дослідженні приділяється всім 11 категоріям впливу на навколишнє середовище, які включають потенціал глобального потепління, абіотичне виснаження, абіотичне виснаження копалин, потенціал підкислення, потенціал евтрофікації, потенціал прісноводної екотоксичності, потенціал токсичності для людини, морський потенціал водної екотоксичності, потенціал виснаження озонового шару, потенціал створення фотохімічного озону, і потенціал екотоксичності земних екосистем.

Потенціал глобального потепління

Вплив на навколишнє середовище з точки зору глобального потепління Потенціал становить 0,079 кг CO^2 в системі доставки безпілотників.
Вплив було викликано в першу чергу виробництвом деталей, які підзвітні 99,2% впливу. CO_2, CH_4 та N_2O – це основні речовини, які сприяють глобальному потеплінню, яке склали 80,88%, 2,75% і 0,89%, відповідно.

Абіотичне виснаження (елементи безпілотника)

Загальний вплив абіотичного виснаження становить 1,67 x 10^{-8} кг SO_2 зокрема, для системи доставки безпілотних літальних апаратів. Виробництво комплектуючих і деталей склали наступні транспортні витрати: 93,50% і 6,44%, відповідно. Основні речовини, які вносять вклад в забруднення повітря, що не викликає ракові захворювання, включає мідь (Cu), срібло (Ag), літій (Li), а також свинець (Pb), на який доводилося 20,47%, 17,67%, 15,72% і 15,37%, відповідно, від впливу в цій категорії.

Абіотичне виснаження (корисні копалини для елементів безпілотника)

Загальна кількість в цій категорії впливу на навколишнє середовище складає 5.16 x 10^{-5} МДж.

Вплив на абіотичне виснаження, складає близько 98,12% від загального обсягу впливу, віднесених до тієї чи іншої категорії. Від експлуатації запчастин і їх транспортування - 1,83%. Основні речовини, які сприяють абіотичному виснаженню є: лігніт, кам'яне вугілля і природний газ, які складають 43,62%, 33,39% і 18,40%, відповідно.

З метою зменшення викидів можуть бути використанні більш екологічні електростанції та поновлювані джерела енергії задля зниження цієї категорії впливу.

Потенціал підкислення повітря

Екологічні наслідки підкислення повітря для такого роду систем складає 5.16 x 10^{-5} кг SO_2. У цій категорії чинників 98,20% загального внеску в цей показник було отримано від виробництва комплектуючих і 1,78% склало транспортування деталей та 0,02% від роботи безпілотника.

Домінуючими факторами підкислення повітря є SO_2, NO_x і H_2S, на які припадає 51,32%, 34,91% і 7,86% внеску в цю категорію впливу, відповідно.

Потенціал евтрофікації

Евтрофікація включає в себе вплив на різні середовища (грунт, повітря або вода) в якості однієї з категорій поживних речовин. Загальний результат цього впливу 8.65 x 10^{-6} кг Фосфатів. Це вказує на те, що потенціал для евтрофікації був отриманий в основному за рахунок виробництва і транспортування, на частку яких припало 93,20% і 6,97% ефекту, відповідно. Основні вкладники в категорію евтрофікації та вплив на забруднення повітря, розраховане на 1-км є NO_x, NO_3, азотний органічний зв'язок, і N_2O, на яку припадає 57,18%, 8,99%, 6,40% і 4,82% внеску в цій категорії впливу, відповідно.

Потенціал прісноводної екотоксичності

Загальний вплив в прісноводну екотоксичність становить 1287,28 кг дихлорбензолу ($C_6H_4Cl_2$). Основні процеси, які сприяють впливу екотоксичності водних ресурсів було виробництво комплектуючих, на яке припадає майже 100%. Є багато речовин, які викликають водну

екотоксичність; деякі приклади включають хімічну природу Cu, Co, Ni і Cd. На їх частку припадає 43,15%, 23,3%, 19,6% і 8,01%, відповідно.

Потенціал токсичності для людини

Система доставки безпілотних літальних апаратів також впливає на людину. Потенціал токсичності становить 5,28 х 10^{-5} кг дихлорбензолу ($C_6H_4Cl_2$). Виробництво деталей вносить майже 100% вкладу в категорію токсичності для людини. NO_x, NO, і NO_2 є основою внеску в категорію евтрофікації, які відповідно дорівнюють 48,62%, 42,65% і 2,22% від суми внеску в цьому розмірі на всю категорію впливу відповідно. Виробництво деталей БПЛА є джерелом викидів важких металів. Викиди Cd, Ni і Cu є в основному викидами з вуглецевих волокон і літій-іонних виробництв.

Потенціал морської водної екотоксичності

У категорії потенційного впливу морської водної екотоксичності, загальний вплив становить 6,9 х 10^6 кг дихлорбензолу ($C_6H_4Cl_2$). Виробництво комплектуючих вносять свій вклад, що складає майже 100%. Важкі метали, які включають в себе Cu, Co, Ni - основні речовини, які сприяють морській водній екотоксичності, та складають 32,43%, 31,09% і 25,94% відсотків відповідно.

Потенціал виснаження озонового шару

Загальний потенціал виснаження озонового шару в безпілотних системах доставки 2.14 х 10^{-12} кг R11-eq – це число було отримане в результаті виробництва і транспортування деталей, які склали 98,17% і 1,83% відповідно основні речовини, які викликали виснаження озонового шару, включаючи діхлортетрафторетан і хлордифторметан, які склали 97,03% і 2,97% відповідно, впливу в цій категорії.

Потенціал виникнення фотохімічного озону

Система показує наступні результати стосовно цієї категорії забруднення 3,82 х 10^{-6} кг Етилену. На виробництво і транспортування на поставку безпілотника доводиться 98,22% і 1,78%. відповідно фотохімічному потенціалу виникнення озону в цій категорії впливу. Враховуються NMVOC,

SO^2 та NO_x. 29,78%, 27,61% і 26,32% внеску, відповідно в цій категорії впливу.

Потенціал екотоксичності земних екосистем

Загальна кількість в категорії впливу на навколишнє середовище екотоксичності земних екосистем складає 3,85 х 10^{-3} кг дихлорбензолу ($C_6H_4Cl_2$) в системі безпілотної доставки. Цей вплив було зпричинено, головним чином, викидами отриманими в результаті операцій над деталями, на які припадало майже 100%. Домінуючий внесок в екотоксичність земних екосистем становлять As, Cr і Pb, які складають відповідно 52,8%, 40,8% і 2,1% від суми вкладу в цю категорію впливу.

ВИСНОВОК ДО РОЗДІЛУ «ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА»

Результати використання системи доставки з безпілотників показують, що основний внесок у забруднення навколишнього середовища вносить така категорія впливу як виробництво деталей.

Експлуатацію деталей можна поділити на три основні категорії, до яких належать видобуток вугілля, експлуатація електростанції та виробництво деталей. Видобуток вугілля та робота електростанції є головними факторами, що впливають на глобальне потепління, абіотичне виснаження (елементи абіотичного виснаження та викопні речовини), підкислення повітря, евтрофікацію грунтів, виснаження озонового шару та потенціал виникнення фотохімічного озону.

Виробництво деталей, особливо виробництво вуглецевої продукції, яка є сировиною для вантажного ящика та виробництво літій-іонів, яке є основою для виготовлення акумулятора, є основними чинниками, що сприяють токсичності для людини, прісноводній екотоксичності, морській екотоксичності та екотоксичності земних екосистем.

Експлуатація системи безпілотників показує, що найменший вплив для всіх категорій впливу за словами Park et. al. (2018) [56], вплив на глобальне потепління за 1 км доставки безпілотником становить 0,004 кг CO^2. Однак потенціал глобального потепління з цього дослідження було встановлено як 0,079 кг CO^2, що показує вищі показники впливу, ніж вплив [56]. Крім того, потенціал глобального потепління за 1 км, пройдений на мотоциклі та електродвигуні становить 0,028 і 0,018 кг CO^2. Це показує, що доставка системою безпілотників є екологічно чистою в порівнянні з іншими системи доставки, особливо коли вони працюють [50].

Безпілотники підходять для коротких польотів з легкими предметами, тоді як наземні транспортні засоби підходять для перевезення важчих товарів на великі відстані.

Однак робота не може цілком включити в собі всі чинники, які не було враховано в цьому дослідженні та які можуть потенційно призвести до різних наслідків. У майбутньому це дослідження може бути розширено і включати в собі іншу сировинну базу для виробництва. Крім того, в наступних роботах планується огляд і порівняння системи БПЛА з, наприклад, автомобілями. Слід оцінити та порівняти їх екологічні показники з результатами цього дослідження. Такого роду перевірка покаже яка система є більш екологічною та доцільною з точки зору бізнесу.

РОЗДІЛ 6. ОХОРОНА ПРАЦІ

6.1. Оцінка надійності системи матеріально-технічного постачання і виробничих зв'язків

Сутність та завдання матеріально-технічного забезпечення. Матеріально-технічне забезпечення – це форма товарного обігу у сфері матеріального виробництва, процес забезпечення підприємств сировиною, матеріалами, комплектуючими, напівфабрикатами, готовими виробами тощо, необхідними для виробничого і невиробничого споживання. Правильно налагоджена система матеріально-технічного забезпечення є запорукою безперебійного забезпечення підприємства всіма видами матеріальних ресурсів, що є важливим елементом наукової організації виробництва.

6.2. Планування матеріально-технічного забезпечення.

Планування матеріально-технічного забезпечення має своєю головною метою визначення оптимальної потреби підприємства у матеріальних ресурсах для забезпечення виробничо-господарської та комерційної діяльності, створення оптимальних запасів товарно-матеріальних цінностей. В основі планування матеріально-технічного забезпечення підприємства лежить план матеріально-технічного постачання, який формується у чотири етапи.

Управління матеріально-технічним забезпеченням – важлива складова частина керівництва виробничо-господарською діяльністю підприємства. На більшості промислових підприємств використовують централізовану структуру управління матеріально-технічним забезпеченням.

Нормування витрат сировини і матеріалів та показники використання матеріальних ресурсів. Нормування витрат сировини і матеріалів є одним з

найбільш головних завдань матеріально-технічного забезпечення підприємства, завдяки якому підприємство має можливість заощадливо використовувати наявні ресурси і не допускати необґрунтованого підвищення собівартості продукції.

Норма витрат – це планова величина максимально допустимих витрат матеріальних ресурсів на виробництво одиниці продукції або роботи, встановлена для певних виробничо-технічних умов.

Запаси матеріально-технічних ресурсів підприємств. Сутність управління запасами матеріально-технічних ресурсів промислового підприємства полягає у здійсненні постійного контролю за станом запасів і прийнятті рішень, спрямованих на економію часу і засобів за рахунок мінімізації витрат на утримання запасів, необхідних для своєчасного виконання виробничої програми.

Планування економії матеріально-технічних ресурсів. Одне з головних завдань служби матеріально-технічного забезпечення полягає у плануванні економії матеріально-технічних ресурсів. Для ефективного здійснення цього завдання необхідно дотримуватись низки умов, що забезпечують економію сировини та матеріалів.

6.3. Безпека життєдіяльності

Відповідно до статей 33, 34 Закону України «Про захист населення і територій від надзвичайних ситуацій техногенного та природного характеру».

1. Загальні положення.
2. Підготовка і перепідготовка керівного складу підприємств, установ і організацій.

Підготовка і перепідготовка керівного складу підприємств, установ і організацій у сфері цивільного захисту населення і територій від надзвичайних ситуацій, запобігання та оперативного реагування на них (далі – сфера цивільного захисту) проводиться в

Інституті державного управління у сфері цивільного захисту (далі – Інститут), на курсах цивільної оборони, а також під час проведення навчально-методичних зборів та періодичних навчань, тренувань за планами реагування на надзвичайні ситуації та планами локалізації і ліквідації аварій (катастроф).

3. Підготовка населення, яке зайняте у сферах виробництва та обслуговування. Вивчення працівниками основних способів захисту і дій у надзвичайних ситуаціях техногенного і природного характеру здійснюється на підприємствах, в установах і організаціях за спеціальними програмами підготовки населення, які затверджуються Міністерством України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи, а також при прийнятті на роботу і при подальшій роботі у формі інструктажів з питань охорони праці згідно з Типовим положенням про навчання з питань охорони праці, затвердженим наказом Комітету по нагляду за охороною праці України та зареєстрованим у Міністерстві юстиції України.

4. Підготовка студентів, курсантів, учнів та вихованців закладів освіти. При розробці державних стандартів освіти з кожного освітнього
та освітньо-кваліфікаційного рівня обов'язково передбачається мінімум змісту питань з підготовки населення до дій при виникненні надзвичайних ситуацій.

5. Практична підготовка та відпрацювання дій за планами реагування на надзвичайні ситуації, планами локалізації і ліквідації аварій (катастроф) під час підготовки та проведення спеціальних комплексних об'єктових навчань, тренувань.

6. Просвітницько-інформаційна робота та пропаганда знань серед населення з питань захисту та дій у надзвичайних ситуаціях. Просвітницько-інформаційна робота з населенням щодо питань

захисту і дій у надзвичайних ситуаціях здійснюється за місцем проживання у мережі навчально-консультаційних пунктів місцевих органів самоврядування, а також шляхом самостійного вивчення посібників, пам'яток, іншого друкованого навчально-інформаційного матеріалу, перегляду та прослуховування спеціального циклу теле і радіо передач.

7. Міністерство України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи, міністерства та інші центральні органи виконавчої влади, Рада міністрів Автономної Республіки Крим, місцеві державні адміністрації та органи місцевого самоврядування відповідно до своїх повноважень здійснюють підготовку населення до дій у надзвичайних ситуаціях.

8. Забезпечення заходів щодо підготовки населення до дій у надзвичайних ситуаціях.

6.4. Розрахунки можливих ризиків при роботі

Під час виконання дипломної роботи магістра на тему “Дослідження і розробка методів надійної навігації безпілотних літальних апаратів” уся розробка повинна виконуватись із врахуванням вимог техніки безпеки на робочому місці, пожежної безпеки, відповідно з діючими нормативно-правовими актами та встановленими нормами.

Для забезпечення безпечної роботи з ПК та скорочення негативного впливу на здоров’я користувача, потрібно дотримуватись норм, визначених у наступних документах:

— Закон України “Про охорону праці”;

— НПАОП 0.00-7.15-18 “Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями”;

— ДСанПІН 3.3.2.007-98 “Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних

машин”;

— НПАОП 40.1-1.21-98 “Правила безпечної експлуатації електроустановок споживачів”;

— НАПБ А.01.001-2004 “Правила пожежної безпеки в Україні”.

Перед початком розробки системи навігації було проведено інструктажі з охорони праці, техніки безпеки та протипожежної безпеки.

У приміщені де проводиться розробка, площа на одне робоче місце повинна становити не менше ніж 6,0 $м^2$, а об'єм не менше ніж 20,0 $м^3$. Дотримано вимог стосовно освітлення, оптимальних умов мікроклімату, ергономічних характеристик основних елементів робочого місця, рівнів шуму, електромагнітного, ультрафіолетового та інфрачервоного випромінювання та електростатичного поля викладених у ДСанПіН 3.3.2-007-98.

Приміщення, у якому передбачається експлуатація ПК, не повинно межувати з будівлями, у яких рівні шуму і вібрації перевищує допустимі значення за нормативними документами ДСН 3.3.6.037-99, ДСН 3.3.6. 039-99. Крім цього, необхідно передбачити звукоізоляцію огороджувальних конструкцій приміщень з ПК від шуму, що задовольняє вимогам ДСТУ 2325-93, ДСТУ 3130-95.

Оскільки, основне навантаження під час розробки системи навігації БПЛА припадало на зорову систему, тому штучне освітлення в приміщеннях з робочими місцями, обладнаними ПК, здійснювалось системою загального рівномірного освітлення, а значення освітленості на поверхні робочого столу в зоні розміщення документів становила 300-500 лк. За умов коли ці значення освітленості неможливо було забезпечити системою загального освітлення, використовувалось місцеве освітлення. Також, потрібно робити короткі перерви кожні 20 хв., що дозволить зменшити напруженість зорового нерву і відповідно знизити імовірність його травмування. Рекомендується, час від часу, робити спеціальний комплекс вправ для зниження втомлюваності очей та зменшення нервового напруження.

Для профілактики загальної втоми і особливо зорового аналізатора важливе значення має організація режиму праці та відпочинку. Загальна тривалість робочого дня не повинна перевищувати 8 годин. Частота і тривалість перерв залежать від типу та інтенсивності виконуваних робіт. Під час робіт, які виконуються з великим навантаженням, рекомендуються перерви на 10-15 хвилин через кожну годину, а при неінтенсивній і монотонній роботі - на 10-15 хвилин через кожні дві години. Кількість мікропауз (тривалістю до хвилини) потрібно регулювати індивідуально. Зміст регламентованих перерв може бути різний: виробнича гімнастика (вправи для очей, гімнастика, спрямована на корекцію вимушеної робочої пози, поліпшення венозного кровообігу, часткову дисфункцію рухової активності), альтернативна допоміжна робота, приймання їжі тощо.

Так, як розробка системи навігації БПЛА потребує високої концентрації уваги важливим то важливим аспектом є забезпечення шумоізоляції, для цього обладнання, яке становило джерело шуму було розміщено в окремих приміщеннях.

Згідно з правилами протипожежної безпеки приміщення в яких відбувалося дослідження було обладнано системою автоматичної протипожежної сигналізації та необхідною кількістю вогнегасників.
Для захисту користувачів ПК та дотримання регламентованих норм світлового випромінення використовувалися екранні світлофільтри та локальні світлофільтри. При перевірці та оцінці технологічної оснащеності робочого місця було з’ясовано, що ПЕОМ, типу ПК, згідно нормативно-технічної документації, відповідає вимогам НПАОП 40.1-1.21-98.

Відповідно до НПАОП 40.1-1.21-98 це приміщення належить до класу приміщень без підвищеної небезпеки, так як в приміщенні відсутні сильна вологість (відносна вологість не перевищує 75 %), струмопровідний пил, можливість одночасного дотику людини до корпусів ПЕМВ, типу ПК, і до заземлених металевих конструкцій будівлі.

Живлення робочого місця здійснюється від трифазної чотирьох провідної мережі змінного струму з глухозаземленою нейтраллю, частота змінного струму 50 Гц, напруга 220 В, система заземлення TN-C-S типу – система в якій нульовий захисний і нульовий робочий провідники поєднані в одному провіднику на всьому її протязі.

З метою уникнення ризику ураження людини електричним струмом передбачається використання таких технічних засобів захисту: необхідно проводити контроль ізоляції відповідно до вимог ПУЕ-2011. Контроль проводити між нульовим і фазним провідниками і між фазами. Опір ізоляції не менше 500 кОм на фазу. Контроль проводити не рідше 1 разу на рік при відключеному електроживленні.

Роботи в приміщенні, згідно з ДСН 3.3.6.042-99, відносяться до категорії робіт по енерговитратам організму «легка 1а» (роботи, вироблені сидячи, які не потребують систематичного фізичного напруження і переміщення важких предметів з енерговитратами організму до 120 ккал/год).

Виходячи з того, що передбачено ДСН 3.3.6-042-99 «Санітарні норми мікроклімату виробничих приміщень» встановлені оптимальні норми температури, вологості і швидкості руху повітря для даної категорії і показані в таблиці.

Таблиця 6.1

Період року	Температура, С		Відносна вологість, %		Швидкості переміщення повітря, м/с	
	Опт.	Доп.	Опт.	Доп.	Опт.	Доп.
Холодний	22-24	19-25	40-60	≤75	22-24	19-25
Теплий	23-25	22-25	40-60	≤55	23-25	22-28

Таблиця – Оптимальні і допустимі норми температури, вологості і швидкості руху повітря

Для забезпечення встановлених норм мікрокліматичних параметрів і чистоти повітря застосовується опалення в холодний період року і кондиціонування повітря в теплий період.

На робочому місці, де відбувається розробка системи навігації, присутні як природне освітлення, так і штучне. Розрахуємо рівень освітленості приміщення при штучному освітленні.

При проектуванні штучного освітлення в приміщеннях необхідно керуватися вимогами ДБН В.2.5-28-2006 «Природне і штучне освітлення» [58]. нормативне значення штучного освітлення Е = 200-500 лк, природного - КПО = 1,2 %.

Розрахунок штучного освітлення проводиться методом коефіцієнта використання світлового потоку. Мета перевірочного розрахунку - визначення фактичної освітленості в приміщенні.

Основна розрахункова формула методу коефіцієнта використання світлового потоку:

$$\Phi_{св} = \frac{E_ф \cdot k_з \cdot S \cdot z}{n \cdot N \cdot \eta \cdot \gamma},$$

де $E_ф$ – фактична освітленість, лк;

S – площа освітлюваного приміщення, S = 24,96 м^2;

z – коефіцієнт нерівномірності освітленості, z = 1,1;

$k_з$ – коефіцієнт запасу, що враховує запилення світильників і знос джерел запасу світла в процесі експлуатації. Для приміщення дисплейного залу, освітлюється люмінесцентними лампами і за умови чищення світильників не рідше двох разів на рік, $k_з$ = 1,4;

N – число світильників в ряду, N = 5;

η – коефіцієнт використання світлового потоку ламп, η = 0,52;

γ – коефіцієнт затінення, γ = 0,8;

n – число рядів світильників, n = 2.

Світловий потік ламп розраховується за формулою:

$$F_{св} = n_л \cdot F_л,$$

де $F_{св}$ – світловий потік ламп;

$n_л$ – кількість ламп в світильнику, $n_л = 2$;

$F_л$ – світловий потік лампи, $F_л = 2000$

$$F_{св} = 2 \cdot 2000 = 4000.$$

Фактична освітленість розраховується за формулою:

$$E_ф = \frac{F_{св} \cdot n \cdot \eta}{S \cdot k \cdot z},$$

$$E_ф = \frac{4000 \cdot 2 \cdot 0.52}{24{,}96 \cdot 1{,}4 \cdot 1.1} = 108{,}3.$$

Висновок до розділу «Охорона праці та безпека життєдіяльності»

В даному розділі були розглянуто два підрозділи – «Охорона праці» та «Безпека життєдіяльності» після чого було проведено розрахунок освітленості приміщення. В підрозділі «Безпека життєдіяльності» було розглянуто такі питання:

1. Оцінка надійності системи матеріально-технічного постачання і виробничих зв’язків.
2. Планування матеріально-технічного забезпечення
3. Забезпечення безпеки життєдіяльності робітників і службовців об'єкта та населення в умовах надзвичайних ситуацій техногенного походження.

Також були проведені розрахунки можливих ризиків при роботі:

В першому питанні було надано оцінку надійності системи виробничих зв'язків та оцінку надійності системи з використанням БПЛА.

В другому питанні було вироблено план матерільно-технічного забезпечення системи з використанням БПЛА.

В третьому питанні підрозділу було проаналізовано безпеку життєдіяльності робітників і об'єкта в умовах техногенної катастрофи, а також забезпечення безпеки населення в аналогічних умовах.

Як складова частина розрахункової роботи було проведено розрахунки освітленості в приміщенні, де проводились всі етапи планування та моделювання системи.

Як видно з наведених розрахунків фактична освітленість не відповідає вимозі ДБН В.25-28-2006 [71] «Природне і штучне освітлення». Необхідно обов'язкове збільшення освітленості на робочому місці за рахунок збільшення потужності світильника або використання місцевого освітлення для роботи з документами.

ЗАГАЛЬНИЙ ВИСНОВОК

Метою даної дипломної роботи було подальший розвиток мого дослідження [1–4] т.з. задачі комівояжера у застосуванні до сучасних практичних проблем, що виникли у зв'язку з появою безпілотних літальних апаратів (БПЛА). Як і в попередній роботі, я використав не метод прямого перебору усіх варіантів, а продовжив випробування методів штучного інтелекту. За основу було взято з англомовних джерел і подано розробці модифікації нейронної мережі 'зі вчителем' на основі механізму "привертання уваги" для вирішення проблем оптимізації систем БПЛА з певними ускладненнями щодо доставки вантажів. На вхід мережі подавались дані у вигляді координат в реальному мастабі, попередньо нормалізовані на проміжку [0, 1), і також ваги вантажів для доставки низці замовників. Результатом роботи нейронної мережі за 19 стадій навчання (епох) було сформовано модель для знаходження оптимальних варіантів шляху і покращення існуючих варіантів оптимізації пошуку маршрутів як окремих БПЛА, так і їх систем.

В роботі запропонована модифікація багатошарової нейронної мережі 'зі вчителем' на основі 'привертання уваги' з використанням 3 кодерів. За 19 стадій навчання результати такої моделі вже були дуже близькими до існуючих, тестових, варіантів оптимізації і відповідали класу точності такого роду алгоритмів. На основі проведених експериментів для 5, 20, 50, 53, 100 точок можна зробити висновки, що модель є гарною альтернативою сучасним методам вирішення CVRP (Capacitated Vehicle Routing Problem). В роботі було також враховано мінімізацію кількості БПЛА, необхідних для максимально ефективного виконання завдання з доставки. За рахунок цього можна значно зменшити потенційні витрати бізнесу використовуючи наявні ресурси найбільш оптимальним чином.

Під час побудови візуалізації також було враховано фактори часу (перельоту та процесу доставки) та висоти (висотних коридорів).

Для розробки програмної моделі мережі було використано мову програмування Python, бо в ній вже імплементовано безліч фреймворків для роботи з машинним навчанням та великими об’ємами даних. Розроблена програмна модель дозволяє:

1. З легкістю налаштувати процес навчання, просто запустивши програму-скрипт.
2. Всі етапи навчання, включаючи втрати кожної з епох та загальні втрати в порівнянні з середнім значенням, друкуються в консоль, щоб можна було робити висновки та налаштовувати параметри навчання емпіричним шляхом.
3. Зберегти результати роботи мережі для подальшого навчання або використання в роботі для системи доставки.
4. Зробити візуалізації розв'язків мережі як на реальному прикладі, так і на випадковій валідаційній вибірці.
5. Імплементовано можливість збереження і форматування розв'язків мережі відразу в форматі .waypoints, що дозволяє використовувати їх на реальному прикладі під час використання наземної станції керування.

За допомогою розробленої програми був проведений аналіз параметрів алгоритму, визначені їх оптимальні значення для кожного тестового маршруту БПЛА. Було знайдено емпіричним шляхом оптимальні гіперпараметри та техніки оптимізації для отримання найкращих результатів.

Робота нейронної мережі була порівняна з наявними валідаційними вибірками інших евристичних алгоритмів, реалізованими для задачі оптимізації. Отримані дані показують, що розроблений алгоритм практично для усіх проведених тестувань показує кращі результати, ніж інші з поправкою на те, що навчання відбувалось лише 19 стадій та з використанням 1 відеокарти протягом 3 годин. Це дозволяє стверджувати, що даний алгоритм є доволі ефективним для рішення задачі оптимізації маршрутів БПЛА і має великий потенціал його використання для вирішення проблем бізнесу.

Розроблений алгоритм також може бути використаний для розв’язання інших прикладних задач оптимізації, таких як задача маршрутизації наземного транспорту, у різного роду службах онлайн-доставки вантажів, в пошуково-рятувальних операціях з використанням системи дронів, операціях різного роду досліджень ландшафту та екологічного нагляду за певною місцевістю, зрошування місцевості інсектицидами, задача маршрутизації складних логістичних систем та телекомунікаційних мереж, а також різних задач з області оптимізації виробничих процесів на підприємствах.

Результати розрахунків використання системи доставки з безпілотників показують, що основний внесок у забруднення навколишнього середовища вносить така категорія впливу як виробництво деталей. В порівнянні з іншими способами доставки він є найекологічнішим, особливо при роботі.

Виходячи з розрахунків в розділі з охорони праці та безпеки життєдіяльності, необхідне обов'язкове збільшення освітленості на робочому місці за рахунок збільшення потужності світильника, або використання місцевого освітлення для роботи з документами.

На сьогоднішній день використання БПЛА для онлайн-магазинів доставки є перспективним напрямком, оскільки навіть в випадку пандемії, та як наслідку карантину, ми можемо цілком виключити людський фактор з процесу доставки. Перевагами впровадження дронів в сферу доставки вантажів можна віднести: високу оперативність, що є дуже важливим фактором в надзвичайних ситуаціях; економічну ефективність, за рахунок доволі низької вартості дрона; в порівнянні з вартістю людської праці на довгостроковій перспективі, надійність, оскільки відсутній людський фактор; відсутнє або суттєве зменшення загрози для життя та здоров’я персоналу.

Дана робота, таким чином, розширює деякі наші практичні можливості.

ПОСИЛАННЯ

1. Kalmykov V.V., Gayev.Ye.A. The travelling salesman problem in the engineering education programming curriculum. Proceedings of the National Aviation University. 2017. N3(72). -- 90–98 pp.
2. Kalmykov V.V., **“**Minimization of aviation routes by artificial intelligence methods” Bachelor’s thesis 2019
3. K.Khavray, A.Skoroded, O.Linnick, O.Boiko, D.Malinina, A.Baboriga, M.Borosentsev, P.Karengin, V.Kalmykov, A.Rychik, Yu.Simakin, T.Pruss Digital Laboratory of Information Processes Theory: an innovative educational approach. -- Матеріали XIII Міжнародної науково-технічної конференції “Авіа-2017”. 19-21 квітня, Київ: НАУ, 2017. Секц. 9.42, p. 638–641. http://avia.nau.edu.ua/doc/avia-2017/AVIA_2017.pdf
4. Калмиков В.В., Гаєв Є.О. MATLAB-досвід у задачі комівояжера. Тези Загальноукраїнської конференції "MATLAB та комп'ютерні обчислення в освіті, науці та інженерії", Київ: НАУ, травень 2019, с. 26.
5. https://siteofthedrones.com/types-of-drones/ - сайт Tommy Cuentos присвячений історії і сучасному стану дронів.
6. https://en.wikipedia.org/wiki/Travelling_salesman_problem
7. https://en.wikipedia.org/wiki/Glossary_of_graph_theory_terms – словник термінів з теорії графів
8. https://uk.wikipedia.org/wiki/Навігація – детальна інформація стосовно навігації
9. https://uk.wikipedia.org/wiki/Задача_комівояжера – на сайті розкрито поняття задачі подорожуючого торговця з точки зору побудови алгоритмів
10. https://en.wikipedia.org/wiki/Knapsack_problem – на сайті описано проблему “пакування рюкзаку” і можливі варіанти її розв'язку
11. https://tomekent.com/post/multiagent-travellingsalesmanproblem – на сайті описано підхід до проблеми мультиагентної(для системи рухомих об’єктів) задачі подорожучого торговця

12. www-or.amp.i.kyotou.ac.jp/members/umetani/ppt/iwh20020726/Slide02.html – презентація проблеми оптимального розрізання листа сталі
13. https://en.wikipedia.org/wiki/Synergetics_(Fuller) – на сайті розкрито поняття синергетики в математиці та програмуванні
14. https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms – на сайті детально описано алгоритм оптимізації мурашиної колонії
15. https://en.wikipedia.org/wiki/Simulated_annealing – на сайті описано з математичної та алгоритмічної точки зору сутність алгоритму імітації відпалу
16. https://en.wikipedia.org/wiki/2-opt – на сайті описано 2-opt алгоритм, як одна з альтернатив методів штучного інтелекту для проблеми подорожуючого торговця
17. https://en.wikipedia.org/wiki/Concorde_TSP_Solver – на сайті описано найвдаліший алгоритм оптимізації пройденого шляху, що вирішив проблему для всіх міст на планеті і на даний момент є еталонним
18. https://developers.google.com/optimization – веб сторінка компанії Google на якій описані різні алгоритми оптимізації і приклади до них
19. https://docs.qgroundcontrol.com/master/en/PlanView/PlanView.html – офіційний веб сайт з документацією компанії Qgroundcontrol, що займається виробництвом наземних станцій керування БПЛА
20. https://ardupilot.org/planner/docs/mission-planner-overview.html – офіційний сайт з документацією компанії, що виробляє Mission Planner
21. https://dronekit-python.readthedocs.io/en/latest/
22. https://ardupilot.org/ – офіційний сайт Mission Planner та іншого програмного забезпечення в сфері наземних станцій управління польотом
23. https://mavlink.io/en/ – сайт на якому описано роботу протоколу зв’язку БПЛА
24. https://ardupilot.org/copter/docs/rtl-mode.html – детальний опис режиму польоту “повернення додому”
25. https://ardupilot.org/copter/docs/auto-mode.html - опис сценацію автоматичного польоту дрона в середовищі Mission Planner

26. https://ardupilot.org/copter/docs/ac2_guidedmode.html – огляд з прикладами коду сценарію польоту за допомогою Python скрипта.
27. https://www.figma.com/ – веб додаток для створення професійного дизайну
28. https://www.sciencedirect.com/science/article/abs/pii/S0142061515005840 – на сайті описаний гібридний алгоритм мурашиної колонії
29. https://www.researchgate.net/publication/298209081_List-Based_Simulated_Annealing_Algorithm_for_Traveling_Salesman_Problem – на сайті описано роботу алгоритму імітації відпалу
30. https://en.wikipedia.org/wiki/Vehicle_routing_problem – на сайті описано загальне формулювання проблеми оптимізації маршрутів для системи дронів
31. https://tomekent.com/post/multiagent-travellingsalesmanproblem/ – на сайті описано підхід до побудови нейронної мережі для вирішення проблеми оптимальної маршрутизації системи дронів еволюційним алгоритмом
32. https://codeguida.com/post/1418 – веб сторінка де описані загальні принципи нейронних мереж і є простенький приклад побудови такої на мові JavaScript
33. https://papers.nips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf – в статті описано базові принципи побудови нейронних мереж на основі “привертання уваги”
34. https://keras.io/ – сайт бібліотеки для роботи з нейронними мережами на високому рівні, використовуючи лише API та мову програмування Python
35. https://www.tensorflow.org/ - сайт фреймворку для роботи з різними нейронними мережами від компанії Google
36. https://github.com/thushv89/attention_keras – сайт з прикладом побудови моделі на основі привертання уваги користувача Thushan Ganegedara
37. https://arxiv.org/abs/1710.10903 – стаття про Graph Attention Networks та машинне навчання в цілому
38. https://arxiv.org/pdf/1803.08475.pdf - математичне підґрунтя для моделей нейронних мереж на основі привертання уваги для рішення проблем маршрутизації системи дронів

39. https://arxiv.org/abs/2002.03282 – в статті описано алгоритм глибокого навчання з підкріпленням та використанням моделі динамічної уваги для проблем маршрутизації транспортних засобів
40. https://pytorch.org/ – сайт бібліотеки для роботи з нейронними мережами, використовується як більш продуктивна та швидка заміна TensorFlow
41. https://developers.google.com/optimization/routing/vrp – вирішення проблеми маршрутизації від Google
42. https://dash-gallery.plotly.host/Portal/
43. https://dronekit-python.readthedocs.io/en/latest/
44. https://github.com/dronekit/dronekit-python
45. https://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances/ – база даних з валідаційними вибірками проблем подорожуючого торговця

46. https://uk.wikipedia.org/wiki/Дрон
47. https://uk.wikipedia.org/wiki/Безпілотний_літальний_апарат
48. https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle
49. https://ua.euronews.com/2013/10/15/drone-assisted-archeology
50. Stolaroff, J., 2014. The Need for a Life Cycle Assessment of Drone-based Commercial Package Delivery. Lawrence Livermore National Laboratory, LLNL-tr-652316.
51. Wang, D., 2016. The Economics of Drone Delivery. https://spectrum.ieee.org/automaton/robotics/drones/the-economics-of-drone-delivery
52. Davidson,H.,2013.DroneBookDeliveryServiceAimsforTake-offinNovember. https://www.pinterest.com/pin/488992472014637771/.
53. EEA, 2009. EMEP/EEA Emission Inventory Guidebook 2009. 2.C.5.a Copper Production.
54. Koiwanit, J., 2015. Evaluation of Environmental Performance of Hypothetical Canadian Oxy-fuel Combustion Carbon Capture with Risk and Cost Analyses. Doctoral Thesis, University of Regina, Saskatchewan, CA.
55. Mangmeechai, A., 2016. An economic input-output life cycle assessment of

food transportation in Thailand. Int. J. Environ. Stud. 73 (5), 778e790.

56. Park, J., Kim, S., Suh, K., 2018. A comparative analysis of the environmental benefits of drone-based delivery services in urban and rural areas. Sustainability 10 (3), 888.

57. CBC News, 2013. Amazon Unveils Futuristic Plan: Delivery by Drone. https://www.cbsnews.com/news/amazon-unveils-futuristic-plan-delivery-by-drone/.

58. www.deltaquad.com/vtol-drones/view/

59. Є.П. Желібо, Н.М. Заверуха, В.В. Зацарний. Безпека життєдіяльності. 2003.

60. Сакун М.М. Конспект лекцій з дисципліни «Безпека життєдіяльності».-Одеса:ОДАУ, 2010

61. Наказ Державного комітету України з промислової безпеки, охорони праці та гірничого нагляду №3328 від 12.2007 р. про затвердження «Правил вибору та застосування засобів індивідуального захисту органів дихання»

62. Основи охорон праці. Навчально-методичний посібник. /М. М. Сакун, І. В.Москалюк; ОДАУ/. Кафедра безпеки життєдіяльності. – Одеса:«Видавництво ВМВ», 2010 – 160 с.

63. https://te.dsp.gov.ua/ohorona-pratsi-na-pidpryyemstvi-shho-potribno-znaty/

64. https://uk.wikipedia.org/wiki/Охорона_праці

65. https://zakon.rada.gov.ua/laws/show/2694-12#Text

66. http://studies.in.ua/bjd_seminar/1044-ohorona-prac.html

67. https://esop.mcfr.ua/

68. https://medoc.ua/blog/ohorona-praci-na-pidprimstvi-shho-ma-znati-robotodavec

69. https://zakon.rada.gov.ua/laws/show/322-08#n175

70. Удосконалена формула розрахунку коефіцієнтів Пд регуляторів для синтезу складних систем [Электронный ресурс] – Режим доступу: URL: https://iconfs.net/s.infocom2018/udoskonalena-formula-rozrakhunku–koefitsi%D1%94 ntiv-

pd-regulyatoriv-dlya-syntezu–skladnykh-system/ –24.11.2019 р – Заголовок з екрану.

71. Дзюндзюк, Б.В. Охорона праці. Збірник задач. Навч. посібник. [Текст] / Б.В. Дзюндзюк, В.Г. Іванов. – Харків: ХНУРЕ, 2006. – 236 с.

ДОДАТКИ

ДОДАТОК А
Код програмного файлу train.py

```
from time import time
import tensorflow as tf
from baseline import RolloutBaseline
from config import load_pkl
from data import generate_data
from model import AttentionModel

def train(cfg, log_path=None):
    def rein_loss(model, inputs, bs, num_of_batch):
        cost, logarithmic_likelihood = model(inputs, decode_type='greedy', training=True)
        baseline_value = bs[num_of_batch] if bs is not None else baseline.eval(inputs, cost)
        baseline_value = tf.stop_gradient(baseline_value)
        return tf.reduce_mean((cost - baseline_value) * logarithmic_likelihood), tf.reduce_mean(cost)

    def grad_func(model, inputs, base_line, batch):
        with tf.GradientTape() as tape:
            loss, mean_loss = rein_loss(model, inputs, base_line, batch)
        return loss, mean_loss, tape.gradient(loss, model.trainable_variables)

    model = AttentionModel(cfg.embed_dim, cfg.n_encode_layers, cfg.n_heads, cfg.tanh_clipping)
    baseline = RolloutBaseline(model, cfg.task, cfg.weight_dir, cfg.n_rollout_samples,
                               cfg.embed_dim, cfg.n_customer, cfg.warmup_beta, cfg.wp_epochs)
    optimizer = tf.keras.optimizers.Adam(learning_rate=cfg.lr)
    ave_epoch_loss = tf.keras.metrics.Mean()
    ave_cost_loss = tf.keras.metrics.Mean()
    t1 = time()
    for epoch in range(cfg.epochs):
        dataset = generate_data(cfg.n_samples, cfg.n_customer)
        base_line = baseline.eval_all(dataset)
        base_line = tf.reshape(base_line, (
            -1, cfg.batch)) if base_line is not None else None

        for batch, inputs in enumerate(dataset.batch(cfg.batch)):
            loss, mean_loss, grads = grad_func(model, inputs, base_line, batch)
            grads, _ = tf.clip_by_global_norm(grads, 1.0)
            optimizer.apply_gradients(zip(grads, model.trainable_variables))  # optimizer.step
            ave_epoch_loss.update_state(loss)
            ave_cost_loss.update_state(mean_loss)

            if batch % (cfg.batch_verbose) == 0:
                t2 = time()
                print('Epoch %d (batch = %d): Loss: %1.3f cost: %1.3f, %dmin%dsec' % (
                    epoch, batch, ave_epoch_loss.result().numpy(), ave_cost_loss.result().numpy(), (t2 - t1) // 60,
                    (t2 - t1) % 60))
                if cfg.islogger:
                    if log_path is None:
                        log_path = '%s%s_%s.csv' % (cfg.log_dir, cfg.task, cfg.dump_date)  # cfg.log_dir = ./Csv/
                        with open(log_path, 'w') as f:
                            f.write('time,epoch,batch,loss,cost\n')
                    with open(log_path, 'a') as f:
                        f.write('%dmin%dsec,%d,%d,%1.3f,%1.3f\n' % (
                            (t2 - t1) // 60, (t2 - t1) % 60, epoch, batch, ave_epoch_loss.result().numpy(),
                            ave_cost_loss.result().numpy()))
                t1 = time()
        baseline.epoch_callback(model, epoch)
        model.save_weights('{}{}_epoch{}.h5'.format(cfg.weight_dir, cfg.task, epoch), save_format='h5')
```

```
        ave_epoch_loss.reset_states()
        ave_cost_loss.reset_states()

if __name__ == '__main__':
    cfg = load_pkl("./Parameters/VRP20_test.pkl")
    train(cfg)
```

ДОДАТОК Б
Код програмного файлу model.py

```
import random
import tensorflow as tf
from data import generate_data
from decoder import DecoderCell
from encoder import GraphAttentionEncoder

class AttentionModel(tf.keras.models.Model):
    def __init__(self, embed_dim=128, n_encode_layers=3, n_heads=8, tanh_clipping=10., FF_hidden=512):
        super().__init__()

        self.Encoder = GraphAttentionEncoder(embed_dim, n_heads, n_encode_layers, FF_hidden)
        self.Decoder = DecoderCell(embed_dim, n_heads, tanh_clipping)

    def call(self, x, training=True, return_pi=False, decode_type='greedy'):
        encoder_output = self.Encoder(x, training=training)
        decoder_output = self.Decoder(x, encoder_output, return_pi=return_pi, decode_type=decode_type)
        if return_pi:
            cost, ll, pi = decoder_output
            return cost, ll, pi
        cost, ll = decoder_output
        return cost, ll
if __name__ == '__main__':
    model = AttentionModel()
    dataset = generate_data(seed=random.randint(1, 1000))
    return_pi = False
    for i, data in enumerate(dataset.batch(5)):
        output = model(data, decode_type='sampling', return_pi=False)
        if return_pi:
            print(output[0])
            print(output[1])
            print(output[2])
        else:
            print(output[0])
            print(output[1])
        if i == 0:
            break

    print('model.trainable_weights')
    for w in model.trainable_weights:
        print(w.name)
        print(w.numpy())
    model.summary()
```

ДОДАТОК В
Код програмного файлу layers.py

```
import numpy as np
import tensorflow as tf

class DotProductAttention(tf.keras.layers.Layer):
    def __init__(self, clip=None, return_logits=False, head_depth=16, inf=1e+10, **kwargs):
        super().__init__(**kwargs)
        self.clip = clip
        self.return_logits = return_logits
        self.inf = inf
        dk = tf.cast(head_depth, tf.float32)
        self.scale = tf.math.sqrt(dk)

    def call(self, x, mask=None):
        Q, K, V = x
        logits = tf.matmul(Q, K, transpose_b=True) / self.scale

        if self.clip is not None:
            logits = self.clip * tf.math.tanh(logits)
        if self.return_logits:
            if mask is not None:
                logits = tf.where(tf.transpose(mask, perm=(0, 2, 1)), tf.ones_like(logits) * (-np.inf), logits)
            return logits
        if mask is not None:
            logits = tf.where(mask[:, None, None, :, 0], tf.ones_like(logits) * (-np.inf), logits)

        probs = tf.nn.softmax(logits, axis=-1)
        return tf.matmul(probs, V)

class MultiHeadAttention(tf.keras.layers.Layer):
    def __init__(self, n_heads=8,
                 embed_dim=128,
                 clip=None,
                 return_logits=None,
                 need_W=None,
                 **kwargs):
        super().__init__(**kwargs)
        self.n_heads = n_heads
        self.embed_dim = embed_dim
        self.head_depth = self.embed_dim // self.n_heads
        if self.embed_dim % self.n_heads != 0:
            raise ValueError("embed_dim = n_heads * head_depth")

        self.need_W = need_W
        self.attention = DotProductAttention(clip=clip,
                                             return_logits=return_logits,
                                             head_depth=self.head_depth)
        if self.need_W:
            self.Wk = tf.keras.layers.Dense(self.embed_dim, use_bias=False)
            self.Wv = tf.keras.layers.Dense(self.embed_dim, use_bias=False)
            self.Wq = tf.keras.layers.Dense(self.embed_dim, use_bias=False)
            self.Wout = tf.keras.layers.Dense(self.embed_dim, use_bias=False)

    def split_heads(self, T, batch):
        T = tf.reshape(T, (batch, -1, self.n_heads, self.head_depth))
        return tf.transpose(T, perm=(0, 2, 1, 3))

    def combine_heads(self, T, batch):
        T = tf.transpose(T, perm=(0, 2, 1, 3))
        return tf.reshape(T, (batch, -1, self.embed_dim))
```

```
    def call(self, x, mask=None):
        Q, K, V = x
        if self.need_W:
            Q, K, V = self.Wq(Q), self.Wk(K), self.Wv(V)

        batch = K.shape[0]
        output = self.attention([self.split_heads(T, batch) for T in [Q, K, V]], mask=mask)
        output = self.combine_heads(output, batch)
        if self.need_W:
            return self.Wout(output)
        return output

if __name__ == '__main__':
    mha = MultiHeadAttention(n_heads=8, embed_dim=128, need_W=True)
    batch, n_nodes, embed_dim = 200, 21, 128
    x = tf.random.uniform((batch, n_nodes, embed_dim), dtype=tf.float32)
    output = mha([x, x, x])
    print(output.shape)
```

ДОДАТОК В
Код програмного файлу encoder.py

```
import tensorflow as tf
import tensorflow.keras.backend as K
from data import generate_data
from layers import MultiHeadAttention

class ResidualBlock_BN(tf.keras.layers.Layer):
    def __init__(self, MultHeadAttention, BatchNode, **kwargs):
        super().__init__(**kwargs)
        self.MultHeadAttention = MultHeadAttention
        self.BatchNode = BatchNode

    def call(self, x, mask=None, training=True):
        if mask is None:
            return self.BatchNode(x + self.MultHeadAttention(x), training=training)
        return self.BatchNode(x + self.MultHeadAttention(x, mask), training=training)

class SelfAttention(tf.keras.layers.Layer):
    def __init__(self, MultHeadAttention, **kwargs):
        super().__init__(**kwargs)
        self.MultHeadAttention = MultHeadAttention

    def call(self, x, mask=None):
        return self.MultHeadAttention([x, x, x], mask=mask)

class EncoderLayer(tf.keras.layers.Layer):
    def __init__(self, n_heads=8, FF_hidden=512, activation='relu', **kwargs):
        super().__init__(**kwargs)
        self.n_heads = n_heads
        self.FF_hidden = FF_hidden
        self.activation = activation
        self.BN1 = tf.keras.layers.BatchNormalization(trainable=True)
        self.BN2 = tf.keras.layers.BatchNormalization(trainable=True)

    def build(self, input_shape):
        self.MHA_sublayer = ResidualBlock_BN(
            SelfAttention(
                MultiHeadAttention(n_heads=self.n_heads, embed_dim=input_shape[2], need_W=True)
            ),self.BN1 )
        self.FF_sublayer = ResidualBlock_BN(
            tf.keras.models.Sequential([
                tf.keras.layers.Dense(self.FF_hidden, use_bias=True, activation=self.activation),
                tf.keras.layers.Dense(input_shape[2], use_bias=True) ]),self.BN2 )
        super().build(input_shape)

    def call(self, x, mask=None, training=True):
        return self.FF_sublayer(self.MHA_sublayer(x, mask=mask, training=training), training=training)


class GraphAttentionEncoder(tf.keras.models.Model):
    def __init__(self, embed_dim=128, n_heads=8, n_layers=3, FF_hidden=512):
        super().__init__()
        self.init_W_depot = tf.keras.layers.Dense(embed_dim, use_bias=True)
        self.init_W = tf.keras.layers.Dense(embed_dim, use_bias=True)
        self.encoder_layers = [EncoderLayer(n_heads, FF_hidden) for _ in range(n_layers)]

    @tf.function
    def call(self, x, mask=None, training=True):
        x = tf.concat([self.init_W_depot(x[0])[:, None, :],
                       self.init_W(tf.concat([x[1], x[2][:, :, None]], axis=-1))], axis=1)
```

```
        for layer in self.encoder_layers:
            x = layer(x, mask, training)
        return (x, tf.reduce_mean(x, axis=1))

if __name__ == '__main__':
    training = False
    K.set_learning_phase(training)
    batch = 5
    n_nodes = 21
    encoder = GraphAttentionEncoder()
    dataset = generate_data()
    mask = tf.zeros((batch, n_nodes, 1), dtype=tf.bool)
    for i, data in enumerate(dataset.batch(batch)):
        output = encoder(data, training=training, mask=mask)
        print('output[0].shape:', output[0].shape)
        print('output[1].shape', output[1].shape)
        if i == 0:
            break
    encoder.summary()
    for w in encoder.trainable_weights:
        print(w.name)
```

ДОДАТОК Г
Код програмного файлу decoder.py

```
import tensorflow as tf
from data import generate_data
from decoder_utils import TopKSampler, CategoricalSampler, Env
from layers import MultiHeadAttention, DotProductAttention

class DecoderCell(tf.keras.models.Model):
    def __init__(self, embed_dim=128, n_heads=8, clip=10., **kwargs):
        super().__init__(**kwargs)

        self.Wk1 = tf.keras.layers.Dense(embed_dim, use_bias=False)
        self.Wv = tf.keras.layers.Dense(embed_dim, use_bias=False)
        self.Wk2 = tf.keras.layers.Dense(embed_dim, use_bias=False)
        self.Wq_fixed = tf.keras.layers.Dense(embed_dim, use_bias=False)
        self.Wout = tf.keras.layers.Dense(embed_dim, use_bias=False)
        self.Wq_step = tf.keras.layers.Dense(embed_dim, use_bias=False)

        self.MHA = MultiHeadAttention(n_heads=n_heads, embed_dim=embed_dim, need_W=False)
        self.SHA = DotProductAttention(clip=clip, return_logits=True, head_depth=embed_dim)
        self.env = Env

    def compute_static(self, node_embeddings, graph_embedding):
        Q_fixed = self.Wq_fixed(graph_embedding[:, None, :])
        K1 = self.Wk1(node_embeddings)
        V = self.Wv(node_embeddings)
        K2 = self.Wk2(node_embeddings)
        return Q_fixed, K1, V, K2

    @tf.function
    def _compute_mha(self, Q_fixed, step_context, K1, V, K2, mask):
        Q_step = self.Wq_step(step_context)
        Q1 = Q_fixed + Q_step
        Q2 = self.MHA([Q1, K1, V], mask=mask)
        Q2 = self.Wout(Q2)
        logits = self.SHA([Q2, K2, None], mask=mask)
        return tf.squeeze(logits, axis=1)

    def call(self, x, encoder_output, return_pi=False, decode_type='sampling'):
        node_embeddings, graph_embedding = encoder_output
        Q_fixed, K1, V, K2 = self.compute_static(node_embeddings, graph_embedding)
        env = Env(x, node_embeddings)
        mask, step_context, D = env._create_t1()
        selecter = {'greedy': TopKSampler(), 'sampling': CategoricalSampler()}.get(decode_type, None)
        log_ps = tf.TensorArray(dtype=tf.float32, size=0, dynamic_size=True, element_shape=(env.batch,
env.n_nodes))
        tours = tf.TensorArray(dtype=tf.int32, size=0, dynamic_size=True, element_shape=(env.batch,))

        for i in tf.range(env.n_nodes * 2):
            logits = self._compute_mha(Q_fixed, step_context, K1, V, K2, mask)
            log_p = tf.nn.log_softmax(logits, axis=-1)
            next_node = selecter(log_p)
            mask, step_context, D = env._get_step(next_node, D)
            tours = tours.write(i, tf.squeeze(next_node, axis=1))
            log_ps = log_ps.write(i, log_p)

        pi = tf.transpose(tours.stack(), perm=(1, 0))
        ll = env.get_log_likelihood(tf.transpose(log_ps.stack(), perm=(1, 0, 2)), pi)
        cost = env.get_costs(pi)

        if return_pi:
```

```
            return cost, ll, pi
        return cost, ll

if __name__ == '__main__':
    batch, n_nodes, embed_dim = 10, 21, 128
    dataset = generate_data()
    decoder = DecoderCell(embed_dim, n_heads=8, clip=10.)
    for i, data in enumerate(dataset.batch(batch)):
        node_embeddings = tf.ones((batch, n_nodes, embed_dim), dtype=tf.float32)
        graph_embedding = tf.ones((batch, embed_dim), dtype=tf.float32)
        encoder_output = (node_embeddings, graph_embedding)
        cost, ll, pi = decoder(data, encoder_output, return_pi=True, decode_type='sampling')
        print('cost', cost)
        print('ll', ll)
        print('pi', pi)
        if i == 0:
            break
    decoder.summary()
    for w in decoder.trainable_weights:
        print(w.name, w.shape)
```

ДОДАТОК Д
Код програмного файлу config.py

```
import argparse
import os
import pickle
from datetime import datetime
from random import randint

def arg_parser():
    parser = argparse.ArgumentParser()
    parser.add_argument('-m', '--mode', metavar='M', type=str, default='train', choices=['train', 'test'],
                        help='train or test')
    parser.add_argument('--seed', metavar='SE', type=int, default=123,
                        help='random seed number for inference, reproducibility')
    parser.add_argument('-n', '--n_customer', metavar='N', type=int, default=20,
                        help='number of customer nodes, time sequence')
    parser.add_argument('-b', '--batch', metavar='B', type=int, default=128, help='batch size')
    parser.add_argument('-bs', '--batch_steps', metavar='BS', type=int, default=100,
                        help='number of samples = batch * batch_steps')
    parser.add_argument('-bv', '--batch_verbose', metavar='BV', type=int, default=10,
                        help='print and logging during training process')
    parser.add_argument('-nr', '--n_rollout_samples', metavar='R', type=int, default=100,
                        help='baseline rollout number of samples')
    parser.add_argument('-e', '--epochs', metavar='E', type=int, default=20,
                        help='total number of samples = epochs * number of samples')
    parser.add_argument('-em', '--embed_dim', metavar='EM', type=int, default=128, help='embedding size')
    parser.add_argument('-nh', '--n_heads', metavar='NH', type=int, default=8, help='number of heads in
MHA')
    parser.add_argument('-c', '--tanh_clipping', metavar='C', type=float, default=10.,
                        help='improve exploration; clipping logits')
    parser.add_argument('-ne', '--n_encode_layers', metavar='NE', type=int, default=3,
                        help='number of MHA encoder layers')
    parser.add_argument('--lr', metavar='LR', type=float, default=1e-4, help='initial learning rate')
    parser.add_argument('-wb', '--warmup_beta', metavar='WB', type=float, default=0.8,
                        help='exponential moving average, warmup')
    parser.add_argument('-we', '--wp_epochs', metavar='WE', type=int, default=1, help='warmup epochs')
    parser.add_argument('--islogger', action='store_false', help='flag csv logger default true')
    parser.add_argument('-ld', '--log_dir', metavar='LD', type=str, default='./Logs/', help='csv logger dir')
    parser.add_argument('-wd', '--weight_dir', metavar='MD', type=str, default='./Weights/',
                        help='model weight save dir')
    parser.add_argument('-pd', '--pkl_dir', metavar='PD', type=str, default='./Parameters/', help='pkl save dir')
    parser.add_argument('-cd', '--cuda_dv', metavar='CD', type=str, default='0', help='os
CUDA_VISIBLE_DEVICE')
    args = parser.parse_args()
    return args

class Config:
    def __init__(self, **kwargs):
        mode = "test"
        for k, v in kwargs.items():
            self.__dict__[k] = v
        self.task = 'VRP%d_%s' % (self.n_customer, mode)
        self.dump_date = datetime.now().strftime('%m%d_%H_%M')
        for x in [self.log_dir, self.weight_dir, self.pkl_dir]:
            os.makedirs(x, exist_ok=True)
        self.pkl_path = self.pkl_dir + self.task + '.pkl'
        self.n_samples = self.batch * self.batch_steps

def dump_pkl(args, verbose=True, param_log=True):
    cfg = Config(**vars(args))
```

```
    with open(cfg.pkl_path, 'wb') as f:
        pickle.dump(cfg, f)
        print('--- save pickle file in %s ---\n' % cfg.pkl_path)
        if verbose:
            print(''.join('%s: %s\n' % item for item in vars(cfg).items()))
        if param_log:
            path = '%sparam_%s_%s.csv' % (cfg.log_dir, cfg.task, cfg.dump_date)
            with open(path, 'w') as f:
                f.write(''.join('%s,%s\n' % item for item in vars(cfg).items()))

def load_pkl(pkl_path, verbose=True):
    print("Path?  = " + str(os.path.isfile(pkl_path)))
    if not os.path.isfile(pkl_path):
        raise FileNotFoundError('pkl_path')
    print(pkl_path)
    with open(pkl_path, 'rb') as f:
        cfg = pickle.load(f)
        if verbose:
            print(''.join('%s: %s\n' % item for item in vars(cfg).items()))
        os.environ['CUDA_VISIBLE_DEVICE'] = cfg.cuda_dv
    return cfg

def file_parser():
    parser = argparse.ArgumentParser()
    parser.add_argument('-p', '--path', metavar='P', type=str,
                        default='Parameters/VRP20_train.pkl', help='file path, pkl or h5 only')
    args = parser.parse_args()
    return args

def test_parser():
    parser = argparse.ArgumentParser()
    parser.add_argument('-p', '--path', metavar='P', type=str, required=False,
                        help='Weights/VRP***_train_epoch***.h5, h5 file required')
    # parser.add_argument('-b', '--batch', metavar = 'B', type = int, default = 2, help = 'batch size')
    parser.add_argument('-b', '--batch', metavar='B', type=int, default=100, help='batch size')
    parser.add_argument('-n', '--n_customer', metavar='N', type=int, default=20,
                        help='number of customer nodes, time sequence')
    parser.add_argument('-s', '--seed', metavar='S', type=int, default=randint(1, 1000),
                        help='random seed number for inference, reproducibility')
    parser.add_argument('-t', '--txt', metavar='T', type=str,
                        help='if you wanna test out on text file, example: ../OpenData/A-n53-k7.txt')
    parser.add_argument('-d', '--decode_type', metavar='D', type=str, required=False, choices=['greedy',
'sampling'],
                        help='greedy or sampling required')
    args = parser.parse_args()
    return args

if __name__ == '__main__':
    args = arg_parser()
    dump_pkl(args)
```

ДОДАТОК Е
Код програмного файлу data.py

```
import os
import random
import numpy as np
import tensorflow as tf

def generate_data(n_samples=1000, num_of_customers=20, seed=random.randint(1, 1000)):
    generated_data = tf.random.experimental.Generator.from_non_deterministic_state()
    if seed is not None:
        generated_data = tf.random.experimental.Generator.from_seed(seed)
    CAPACITIES = {10: 20., 20: 30., 50: 40., 100: 50.}

    @tf.function
    def tf_rand():
        return [generated_data.uniform(shape=[n_samples, 2], minval=0, maxval=1),
                generated_data.uniform(shape=[n_samples, num_of_customers, 2], minval=0, maxval=1),
                tf.cast(generated_data.uniform(shape=[n_samples, num_of_customers], minval=1, maxval=10,
                                               dtype=tf.int32), tf.float32) / tf.cast(CAPACITIES[num_of_customers],
                                                                                      tf.float32)]
    return tf.data.Dataset.from_tensor_slices(tuple(tf_rand()))

def data_from_txt(path):
    if not os.path.isfile(path):
        raise FileNotFoundError

    with open(path, 'r') as f:
        lines = list(map(lambda s: s.strip(), f.readlines()))
        customer_xy, demand = [], []
        ZERO, DEPOT, CUSTO, DEMAND = [False for _ in range(4)]
        ZERO = True
        for line in lines:
            if (ZERO):
                if (line == 'NODE_COORD_SECTION'):
                    ZERO = False
                    DEPOT = True

            elif (DEPOT):
                depot_xy = list(map(lambda k: float(k) / 100., line.split()))[
                           1:]
                DEPOT = False
                CUSTO = True

            elif (CUSTO):
                if (line == 'DEMAND_SECTION'):
                    DEMAND = True
                    CUSTO = False
                    continue
                customer_xy.append(list(map(lambda k: float(k) / 100., line.split()))[1:])
            elif (DEMAND):
                if (line == '1 0'):
                    continue
                elif (line == 'DEPOT_SECTION'):
                    break
                else:
                    demand.append(list(map(lambda k: float(k) / 100., line.split()))[1])
    tf_rand = [np.expand_dims(np.array(depot_xy), axis=0),
               np.expand_dims(np.array(customer_xy), axis=0),
               np.expand_dims(np.array(demand), axis=0)]
    return tf.data.Dataset.from_tensor_slices(tuple(tf_rand))
```

```
if __name__ == '__main__':
    dataset = generate_data(n_samples=1280, num_of_customers=20, seed=random.randint(1, 1000))

    for i, data in enumerate(dataset.batch(1)):
        print(data[0])
        print(data[1])
        print(data[2])
        if i == 0:
            break
```

ДОДАТОК Ж
Код програмного файлу baseline.py

```
import numpy as np
import tensorflow as tf
from scipy.stats import ttest_rel
from tqdm import tqdm
from data import generate_data
from model import AttentionModel

def copy_model(model, embedded_dimension=128, num_of_customers=20):
    dataset = generate_data(n_samples=1000, num_of_customers=num_of_customers)
    new_model = AttentionModel(embedded_dimension)
    for data in (dataset.batch(5)):
        _, _ = model(data, decode_type = 'sampling')
    for a, b in zip(new_model.variables, model.variables):
        a.assign(b)
    return new_model

def load_model(path, embedded_dimension=128, num_of_customers=20, n_encode_layers=3):
    n_of_dataset = 5
    decode_type = "sampling"
    dataset = generate_data(n_samples=n_of_dataset, num_of_customers=num_of_customers)
    model_loaded = AttentionModel(embedded_dimension, n_encode_layers=n_encode_layers)
    for data in (dataset.batch(n_of_dataset)):
        _, _ = model_loaded(data, decode_type=decode_type)
    model_loaded.load_weights(path)
    return model_loaded

def rollout(model, dataset, batch=100, disable_tqdm=False):
    costs_list = tf.TensorArray(dtype=tf.float32,
                                size=0,
                                dynamic_size=True,
                                element_shape=(batch,))
    for i, inputs in tqdm(enumerate(dataset.batch(batch)),
                          disable=disable_tqdm,
                          desc='Rollout greedy execution'):
        cost, _ = model(inputs, decode_type='greedy')
        costs_list = costs_list.write(i, cost)
    return tf.reshape(costs_list.stack(), (-1,))

def validate(dataset, model, batch=100):
    val_costs = rollout(model, dataset, batch=batch)
    mean_cost = tf.reduce_mean(val_costs)
    print(f"Validation score: {np.round(mean_cost, 4)}")
    return mean_cost

class RolloutBaseline:
    def __init__(self, model, task, weight_dir, n_rollout_samples=100,
                 embedded_dimension=128, num_of_customers=20, warmup_beta=0.8, warmup_epochs=1,
                 from_checkpoint=False, path_to_checkpoint=None, epoch=0,
                 ):
        self.n_rollout_samples = n_rollout_samples
        self.current_epoch = epoch
        self.warmup_epochs = warmup_epochs
        self.beta = warmup_beta
        self.alpha = 0.0
        self.mean_value = None
        self.task = task
        self.from_checkpoint = from_checkpoint
        self.path_to_checkpoint = path_to_checkpoint
        self.embedded_dimension = embedded_dimension
```

```
        self.num_of_customers = num_of_customers
        self.weight_dir = weight_dir
        self._update_baseline(model, epoch)

    def _update_baseline(self, model, epoch):
        if self.from_checkpoint and self.alpha == 0:
            print('Baseline model loaded')
            self.model = load_model(self.path_to_checkpoint, embedded_dimension=self.embedded_dimension,
num_of_customers=self.num_of_customers)
        else:
            print('Baseline model copied')
            self.model = copy_model(model, embedded_dimension=self.embedded_dimension,
num_of_customers=self.num_of_customers)
        self.dataset = generate_data(n_samples=self.n_rollout_samples,
num_of_customers=self.num_of_customers)

        print(f'Evaluating baseline model on baseline dataset (epoch = {epoch})')
        self.bl_vals = rollout(self.model, self.dataset)
        self.mean = tf.reduce_mean(self.bl_vals)
        self.current_epoch = epoch

    def ema_eval(self, cost):
        if self.mean_value is None:  # first iteration
            self.mean_value = tf.reduce_mean(cost)
        else:
            self.mean_value = self.beta * self.mean_value + (1. - self.beta) * tf.reduce_mean(cost)
        return self.mean_value

    def eval(self, batch, cost):
        if self.alpha == 0:
            return self.ema_eval(cost)
        if self.alpha < 1:
            v_ema = self.ema_eval(cost)
        else:
            v_ema = 0.0
        v_b, _ = self.model(batch, decode_type='sampling')
        v_b = tf.stop_gradient(v_b)
        v_ema = tf.stop_gradient(v_ema)
        return self.alpha * v_b + (1 - self.alpha) * v_ema

    def eval_all(self, dataset):
        if self.alpha < 1:
            return None
        val_costs = rollout(self.model, dataset, batch=512)
        return val_costs
    def epoch_callback(self, model, epoch):
        self.current_epoch = epoch
        print(f'Evaluating candidate model on baseline dataset (callback epoch = {self.current_epoch})')
        candidate_vals = rollout(model, self.dataset)
        candidate_mean = tf.reduce_mean(candidate_vals)
        print(f'Epoch {self.current_epoch} candidate mean {candidate_mean}, baseline mean {self.mean}')
        if candidate_mean < self.mean:
            t, p = ttest_rel(candidate_vals, self.bl_vals)
            p_val = p / 2
            print(f'p-value: {p_val}')
            if p_val < 0.05:
                print('Update baseline')
                self._update_baseline(model, self.current_epoch)
        if self.alpha < 1.0:
            self.alpha = (self.current_epoch + 1) / float(self.warmup_epochs)
            print(f'alpha was updated to {self.alpha}')
```

ДОДАТОК З
Код файлу-конфігурації nau.txt

```
NAME : nau
TYPE : CVRP
DIMENSION : 50
EDGE_WEIGHT_TYPE : EUC_2D
CAPACITY : 20
NODE_COORD_SECTION
1 50.416613 30.398312
2 50.38728890 30.35556790
3 50.38356740 30.38269040
4 50.39057180 30.40878300
5 50.37086850 30.40981290
6 50.36939040 30.38921360
7 50.36139680 30.39668080
8 50.36002670 30.41736600
9 50.35219710 30.41247370
10 50.34918490 30.42491910
11 50.35860330 30.43298720
12 50.34989690 30.44680600
13 50.35816620 30.45195580
14 50.36802170 30.44105530
15 50.36870610 30.45727730
16 50.36128730 30.47006610
17 50.36939040 30.47349930
18 50.37836790 30.48113820
19 50.38020150 30.46556000
20 50.38203500 30.44998170
21 50.38980560 30.44242860
22 50.39757610 30.43487550
23 50.39440260 30.45547490
24 50.39122900 30.47607420
25 50.41004910 30.42337420
26 50.40403130 30.47324190
27 50.40829870 30.47229770
28 50.41704310 30.46045300
29 50.44285710 30.44483180
30 50.43020140 30.47710420
31 50.43367320 30.46371460
32 50.42842440 30.45719150
33 50.41825220 30.48345570
34 50.40807990 30.50971980
35 50.39768440 30.52568440
36 50.38728890 30.54164890
37 50.39987240 30.54096230
38 50.41245580 30.54027560
39 50.42514360 30.52825930
40 50.43596930 30.54473880
41 50.44351310 30.53031920
42 50.44919740 30.49839020
43 50.46996120 30.46062470
44 50.45575540 30.44963840
45 50.45356950 30.43762210
46 50.45138350 30.42560580
47 50.44635530 30.40088650
48 50.42623720 30.42251590
49 50.42623720 30.38234710
50 50.42383130 30.35625460
DEMAND_SECTION
1 0
2 5
3 4
4 5
5 8
6 6
7 17
8 9
9 5
10 9
11 9
12 13
13 14
14 5
15 6
16 3
17 12
18 5
19 6
20 8
21 9
22 12
23 9
24 6
25 7
26 8
27 9
28 9
29 7
30 11
31 9
32 7
33 8
34 13
35 6
36 2
37 2
38 9
39 8
40 9
41 8
42 4
43 2
44 2
45 2
46 7
47 2
48 2
49 8
50 3
DEPOT_SECTION
 1
 -1
EOF
```

ДОДАТОК И
Код файлу-програми побудови графіків plot.py

```
import pickle  # For Serialization
from random import randint
from time import time

import numpy as np  # computation library
import plotly.graph_objects as go  # plotting library
import tensorflow as tf  # main library for neural network

from baseline import load_model  # Load baseline part of the model
from data import data_from_txt  # Data formation function


def clarify_path(arr):
    p1, p2 = 0, 1
    output = []
    while p2 < len(arr):
        if arr[p1] != arr[p2]:
            output.append(arr[p1])
            if p2 == len(arr) - 1:
                output.append(arr[p2])
        p1 += 1
        p2 += 1

    if output[0] != 0:
        output.insert(0, 0)
    if output[-1] != 0:
        output.append(0)
    return output


def plot_route(data, pi, costs, title, index_in_batch=0):
    cost = costs[index_in_batch].numpy()
    print("Cost: " + str(cost))
    pi_ = clarify_path(pi[index_in_batch].numpy())
    print("pi[index_in_batch].numpy() = " + str(pi[index_in_batch].numpy()))
    print("pi_ = " + str(pi_))
    depot_xy_pos = data[0][index_in_batch].numpy() * 100
    customer_xy = data[1][index_in_batch].numpy() * 100
    initial_arr = [depot_xy_pos]
    for coord in customer_xy:
        initial_arr.append(coord)
    array_to_serialize = []
    for i in pi_:
        array_to_serialize.append((i, initial_arr[i]))
    with open('data.pkl', 'wb') as f:
        pickle.dump(array_to_serialize, f, pickle.HIGHEST_PROTOCOL)
    print("customer_xy  -> " + str(customer_xy))
    demands = data[2][index_in_batch].numpy()
    print(demands)
    xy = np.concatenate([depot_xy_pos.reshape(1, 2), customer_xy], axis=0)
    list_of_paths, cur_path = [], []
    for idx, node in enumerate(pi_):
        cur_path.append(node)

        if idx != 0 and node == 0:
            if cur_path[0] != 0:
                cur_path.insert(0, 0)
            list_of_paths.append(cur_path)
            cur_path = []
```

```
    path_traces = []
    all_heights = []
    for i, path in enumerate(list_of_paths, 1):
        coords = xy[[int(x) for x in path]]
        lengths = np.sqrt(np.sum(np.diff(coords, axis=0) ** 2, axis=1)) * 100000
        print("lengths" + str(lengths))
        total_length = np.sum(lengths)
        while True:
            current_height = randint(15, 15 + len(all_heights))
            if current_height not in all_heights:
                all_heights.append(current_height)
                break

        def seconds_to_minutes(seconds):
            minutes = 0
            if seconds >= 60:
                minutes = seconds // 60
                return minutes, minutes * 60 - seconds
            return minutes, seconds

        delivery_time = total_length / 8 + 5 * len(lengths) * 60
        mins, secs = seconds_to_minutes(delivery_time)
        path_traces.append(go.Scatter(x=coords[:, 0],
                                      y=coords[:, 1],
                                      mode='markers+lines',
                                      marker=dict(size=12),
                                      name=f'Політ {i} дрона. Довжина = {int(total_length / 100000)} км. Висота:
{current_height}. Час: хв:{int(mins)}, сек:{int(abs(secs))}',
                                      opacity=1.0))
    dict1 = {}
    for i in range(1, len(customer_xy) + 1):
        dict1[i] = (customer_xy[i - 1], demands[i - 1])
    points = enumerate(zip(map(lambda x: round(x, 2), demands), list(filter(lambda x: x > 0, pi_))), 1)
    customer_labels = [f'<b style="color:red;">({str(i[1][1])})</b>' for i in points]

    trace_points = go.Scatter(x=customer_xy[:, 0],
                              y=customer_xy[:, 1],
                              mode='markers+text',
                              name='Покупець (попит)',
                              text=customer_labels,
                              textposition='top center',
                              marker=dict(size=10),
                              opacity=1.0,
                              )

    trace_depo = go.Scatter(x=[depot_xy_pos[0]],
                            y=[depot_xy_pos[1]],
                            mode='markers+text',
                            name='Депо',
                            text=['Депо'],
                            textposition='bottom center',
                            marker=dict(size=26),
                            marker_symbol='hexagon'
                            )

    layout = go.Layout(title=dict(
        text=f'<b>{customer_xy.shape[0]} користувачів. Повна відстань = {int(cost * 1e2)} km. Сумарна вага
= {int(sum(data[2][0].numpy()) * 100)}кг.</b>',
        x=0.5, y=1, yanchor='bottom', yref='paper', pad=dict(b=10)),
        xaxis=dict(title='Широта', showgrid=False, ticks='outside', linewidth=1, mirror=True),
        yaxis=dict(title='Довгота', showgrid=False, ticks='outside', linewidth=1, mirror=True),
        showlegend=True,
```

```
        width=1800,
        height=900,
        autosize=True,
        template="plotly_white",
        legend=dict(x=1.8, xanchor='right', y=0, bordercolor='#444', borderwidth=0)
    )

    data = [trace_points, trace_depo] + path_traces

    print('path: ', pi_)
    fig = go.Figure(data=data, layout=layout)
    fig.show()

if __name__ == '__main__':
    dataset = data_from_txt("OpenData/nau.txt")
    data_path = "Weights/VRP20_train_epoch19.h5"
    num_of_customers = 20
    t1 = time()
    embed_dim = 128
    pretrained_model = load_model(data_path,
                                  embedded_dimension=embed_dim,
                                  num_of_customers=num_of_customers,
                                  n_encode_layers=3)
    print(f'model loading time:{time() - t1}s')
    print(f'data generation time:{time() - t1}s')
    batch_size = 2000
    decode_type = "sampling"
    for i, data in enumerate(dataset.repeat().batch(batch_size)):
        costs, _, pi = pretrained_model(data, return_pi=True, decode_type=decode_type)
        index_in_batch = tf.argmin(costs, axis=0)
        print('costs:', costs)
        print(
            f'decode type:{decode_type}\nminimum cost: {costs[index_in_batch]:.3f} and idx: {index_in_batch}
out of {batch_size} solutions')
        print(f'{pi[index_in_batch]}\ninference time: {time() - t1}s')
        plot_route(data, pi, costs, 'Pretrained', index_in_batch)
        if i == 0:
            break
```