**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**ІМЕНІ ТАРАСА ШЕВЧЕНКА**
Факультет комп'ютерних наук та кібернетики
Кафедра системного аналізу та теорії прийняття рішень

# Кваліфікаційна робота
**на здобуття ступеня магістра**

за спеціальністю 124 Системний аналіз

на тему:

# РОЗВ'ЯЗАННЯ ЛОГІСТИЧНОЇ ЗАДАЧІ КОМІВОЯЖЕРА З УРАХУВАННЯМ СУБ'ЄКТИВНОЇ ОЦІНКИ ЧАСУ ПРОЇЗДУ

Виконав студент 2-го курсу магістратури
Кіптик Кірілл Вікторович

(підпис)

Науковий керівник:
професор, доктор фіз.-мат. наук
Івохін Євген Вікторович

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри системного аналізу та теорії прийняття рішень
«04» травня 2023 р.,
протокол № 11
Завідувач кафедри
О. Г. Наконечний _____
(підпис)

Київ – 2023

# Реферат

Обсяг роботи 64 сторінки, 14 ілюстрацій, 24 джерел посилань.

РОЗВ'ЯЗАННЯ ЛОГІСТИЧНОЇ ЗАДАЧІ КОМІВОЯЖЕРА З УРАХУВАННЯМ СУБ'ЄКТИВНОЇ ОЦІНКИ ЧАСУ ПРОЇЗДУ.

Об’єктом дослідження роботи є процес розв’язування задачі комівояжера використовуючи суб’єктивні оцінки часу отримані з аналізу новинних ресурсів. Предметом дослідження роботи є розробка програмного засобу для аналізу новинних ресурсів використовуючи технологію розпізнавання іменованих сутностей та за допомогою цих оцінок розв’язання логістичної задачі комівояжера.

Метою роботи є дослідження сучасних проблем логістики, аналіз сучасних проблем розподілу даних, аналіз різних підходів до розв’язання задачі комівояжера та на основі даного теоретичного матеріалу скомпонувати оптимальну методику вирішення логістичних проблем за допомогою задачі комівояжера. Навчити лінгвістичну модель аналізувати новинні ресурси та розробити програмну реалізацію розв’язання даної задачі комівояжера.

Методи розробки: теоретичне дослідження, методи розв’язання задачі комівояжера, навчання моделі розпізнавання іменованих сутностей за допомогою методу опорних векторів, визначення суб’єктивних часових оцінок за допомогою нечіткого регулятора. Інструменти розробки: безкоштовний, вільно поширюваний мовний корпус від команди lang-uk; безкоштовна, вільно поширювана бібліотека MITIE; мова програмування Python.

Результати роботи: виконано загальний огляд сучасних проблем логістики та розподілу потоків даних, проаналізовані різні методики розв’язання задачі комівояжера та сконцентровано увагу на методі гілок та меж. Навчено лінгвістичну модель розпізнавання іменованих сутностей (в

нашому випадку локацій), за допомогою даної моделі проведений аналіз новинних ресурсів.

Запропоновано підхід до розв’язання задачі комівояжера методом гілок та меж за умови використання суб’єктивних оцінок на час переміщень. Розроблено програмну реалізацію процесу розв’язування нечіткої задачі комівояжера.

Результати роботи можна використовувати для покращення планування маршрутів з урахуванням нечіткого часу на подолання ділянок шляху в різних навігаційних задачах та ситуаціях прийняття рішень. Запропоноване вдосконалення постановки та способу розв’язання задачі комівояжера дозволяє покращити та вирішити окремі суттєві проблеми у галузі промислової логістики.

# Зміст

# Перелік умовних позначень і скорочень

**OEM –** Original equipment manufacturer (виробник комплектного обладнання)

**ОПР** – особа, що приймає рішення

**NLP** – Natural language processing (обробка природної мови)

**NER** – Named entity recognition (розпізнавання іменованих сутностей)

**SVM** - Support vector machine (метод опорних векторів)

**FLC** – Fuzzy logic control (теорія нечіткого керування)

# Вступ

**Оцінка сучасного стану об'єкта дослідження**. Проблеми логістики можуть сильно відрізнятися залежно від конкретної галузі та регіону, але загалом логістика стикається з проблемами ефективності, рентабельності та стійкості. Деякі з ключових проблем включають збої в ланцюзі поставок, проблеми транспортування, проблеми управління запасами та доставки товару.

Одним із способів вирішення цих проблем є оптимізація, а особливої уваги заслуговує задача комівояжера. Задача комівояжера — це класична задача в комп'ютерній науці, яка прагне знайти найкоротший можливий маршрут для комівояжера, якому потрібно відвідати набір міст і повернутися до початкової точки. Ця проблема має практичне застосування в логістиці, де її можна використовувати для оптимізації маршрутів доставки, оптимізації промислових процесів та зниження транспортних витрат.

Поточний стан задачі комівояжера полягає в тому, що вона все ще є активною областю дослідження, для її вирішення розробляються нові алгоритми та методи. Проблема є NP-складною, тобто невідомо, чи існує поліноміальний алгоритм для її вирішення. Однак існує кілька евристичних і наближених алгоритмів, які забезпечують хороші рішення на практиці.

В останні роки також був досягнутий значний прогрес у вирішенні задачі комівояжера з використанням квантових обчислень. Було показано, що квантова нормалізація забезпечує прискорення вирішення задачі, особливо для великих випадків задачі.

Задачею комівояжера займалися багато вчених, але найбільший вклад в дану сферу внесли: Джордж Данциг, Річард Карп, Річард Беллман, Нікос Крістофідес, Джон Голланд та ін. У працях цих вчених розглядаються питання удосконалення існуючих методів та/чи запровадження нових. В.

даній роботі ми не раз будемо посилатися на роботи цих вчених, як і в теоретичному досліджені, так і в практичній частині. Особливої уваги заслуговує робота Зайченка Ю.П. в області нечіткої логіки, різних моделей даної області та методик, в тому числі й нечіткого регулятора.

**Актуальність роботи та підстави для її виконання**. При вирішені задач сфери логістики не раз виникають випадки коли гарно поставлена задача лінійного програмування може повністю змінити проблемну ситуацію, навіть зробити логістичний процес більш ефективним та оптимізованим. Для головної задачі даної роботи була вибрана більш вузька область використання задачі комівояжера, але не поступається її актуальністю.

Особливо під часи війни, серйозних природних катаклізмів та інших негараздів, одним з найактуальніших джерел новин є новинні веб-сайти, тож в даній роботі ми їх і використовуємо для визначення суб'єктивних часових оцінок кожного шляху, що планує подолати комівояжер. Таким чином, розроблений підхід можна тестувати та використовувати на багатьох транспортних задачах в реальному часі, наприклад для економії палива чи економії часу проїзду.

**Мета й завдання роботи**. Метою кваліфікаційної роботи є розробка сучасних методик розв'язування задач на прикладі вирішення задачі комівояжера, використовуючи суб'єктивні часові оцінки тривалості переміщень. Для досягнення цієї мети поставлено такі завдання:

- Провести аналіз сучасних проблем логістики.
- Дослідити методики ефективного накопичення та розподілу даних.
- Проаналізувати підходи до розв'язання задачі комівояжера.
- Розробити конструктивну методику вирішення логістичних проблем на прикладі нечіткої задачі комівояжера.

- Формалізувати підхід для навчання лінгвістичних моделей на основі новинних ресурсів.
- Розробити програмну реалізацію запропонованого методу розв’язання нечіткої задачі комівояжера з використанням елементів машинного навчання.

**Об'єкт, методи й засоби розроблення**. Об’єктом дослідження роботи є процес розв’язування задачі комівояжера використовуючи суб’єктивні оцінки часу отримані з аналізу новинних ресурсів. Предметом дослідження роботи є розробка програмного засобу для аналізу новинних ресурсів використовуючи технологію розпізнавання іменованих сутностей та за допомогою цих оцінок розв’язання логістичної задачі комівояжера.

Для написання кваліфікаційної роботи були використані наступні методи розроблення: теоретичне дослідження, методи розв’язання задачі комівояжера, навчання моделі розпізнавання іменованих сутностей за допомогою методу опорних векторів, визначення суб’єктивних часових оцінок за допомогою нечіткого регулятора.

**Можливі сфери застосування**. Результати роботи можна використовувати для покращення планування маршрутів з урахуванням нечіткого часу на подолання ділянок шляху в різних навігаційних задачах та ситуаціях прийняття рішень. Запропоноване вдосконалення постановки та способу розв’язання задачі комівояжера дозволяє покращити та вирішити окремі суттєві проблеми у галузі промислової логістики.

# Розділ 1. Моделі та постановки сучасних оптимізаційних задач

## 1.1. Моделі сучасних задач логістики

Більше 90% найбільших світових компаній зіткнулися з перебоями в логістиці, спричиненими пандемією COVID-19. Світовий ринок і досі повністю не відновився після пандемії, а розв'язана росією жорстока війна в Україні ще більше сколихнула світовий ринок. Через неминучі санкції проти країни агресора, та минулі події зв'язані з пандемією, більшість менеджерів логістичних компаній відчули серйозні збої на власному досвіді, оскільки це все виявило слабкі сторони традиційних робочих процесів у існуючій логістиці ланцюгів поставок.

Наприклад, відсутність вертикального бачення усіх процесів, застарілі процеси управління попитом, недостатня стійкість до змін попиту та несподівані збої через залежність від ручних зусиль у логістичних операціях зруйнували ланцюжок поставок під час першої хвилі карантину.

Компанії, що займаються логістикою, були змушені аналізувати свої логістичні процеси. Вони розуміли, що зміни в поведінці та очікуваннях клієнтів навряд чи зможуть усунути ці несподівані проблеми логістики, навіть коли пандемія вщухне. Покупці як B2B, так і B2C – очікували швидшої доставки та зручних можливостей відстеження товарів.

Стало зрозуміло, що компаніям необхідно швидко оптимізувати управління логістикою. Залежно від поставленої задачі, існує багато різних математичних підходів до різних логістичних проблем, такі як лінійне програмування, оптимізація мереж, аналіз рішень, генетичні алгоритми та інше. Але без розуміння головних проблем логістичної галузі та напрямків для формулювання цих проблем, ніяка подальша оптимізація не можлива,

тож зупинимся на цьому більш детально. Існують наступні проблеми даної галузі[1][2][3]:

- **Збільшення транспортних витрат.** У той час як російське повномасштабне вторгнення на Україну спричинило нещодавню нестабільність цін на паливо, транспортні витрати зростали і до цього останні кілька років. Декілька транспортних компаній прогнозують, що ставки річних контрактів цього року зростуть двозначними цифрами через збільшення цін на паливо.
- **Невідповідність трекінгу.** Незважаючи на переваги різного програмного забезпечення, багато брендів продовжують дотримуватися ручних процесів відстеження. Використання електронних таблиць і кількох програм для однієї діяльності знижує продуктивність та ефективність робочої сили.
- **Обмежена видимість відправлень.** Сучасні споживачі очікують від своїх замовлень видимості. Однак недостатня видимість у всьому ланцюжку постачання може спричинити проблеми, які можуть серйозно перешкодити потоку товарів. Без наскрізної прозорості непотрібні затримки стають нормою, як і неефективність роботи складу.
- **Фрагментована комунікація.** Логістичний ланцюжок поставок починається з виробництва і закінчується доставкою кінцевого продукту клієнту. На жаль, OEM-виробники часто страждають від відсутності інтегрованого каналу зв'язку. Цей недолік призводить до фрагментованого зв’язку, що негативно впливає на час доставки та ефективність.
- **Порожні милі.** Порожні милі, означають пробіг, накопичений під час руху порожнього контейнера чи причепа без отримання прибутку. Вони призводять до непотрібного збільшення витрат, негативного впливу на

навколишнє середовище та негативно впливають на ефективність як перевізників, так і відправників.

- **Затримки доставки.** Спіральний ефект пандемії спричинив закриття фабрик і низку проблем, пов’язаних із нестачею робочої сили. Крім того, багато фабрик мають ізольовані та застарілі логістичні процеси, що негативно впливають на час доставки.
- **Форс-мажорні обставини та інше.** Як правило обставини, що не можна передбачити математичними методами, але мають глобальний вплив на логістику, такі як природні катаклізми (землетрус в Туреччині та Сирії, 2023р), або війни (російське воєнне вторгнення в Україну, 2022-до сьогодні).

Кожна з цих проблем має свої труднощі, деякі вирішуються завдяки гарному менеджерському відділу, а деякі заточені під спеціалістів системного аналізу, дата аналізу та інших, саме на таких ми і зупинимся. Даний клас проблем передбачає аналіз та оптимізацію логістичних операцій, включаючи планування, координацію та контроль руху та зберігання товарів, послуг та інформації. Ось деякі математичні підходи до вирішення різних таких задач[2][3]:

- **Лінійне програмування.** Лінійне програмування — це математичний метод, який використовується для оптимізації лінійної цільової функції з урахуванням ряду лінійних обмежень. Його можна використовувати для оптимізації логістичних операцій, таких як транспортування, управління запасами та розташування об’єктів. Саме на цьому ми зупинимося більш детально в наступних розділах даної роботи.
- **Оптимізація мережі.** Оптимізація мережі – це процес оптимізації потоку товарів, послуг або інформації через мережу взаємопов’язаних вузлів. Це можна зробити за допомогою таких методів, як алгоритми

найкоротшого шляху, алгоритми максимального потоку та алгоритми потоку мінімальних витрат.

- **Теорія масового обслуговування.** Теорія масового обслуговування — це математичне дослідження потоків вимог на обслуговування, що надходять у систему і виходять з неї, тривалості очікування і довжини черг. Це можна використовувати для моделювання та оптимізації логістичних операцій, таких як черги вантажівок на завантажувальних платформах або черги клієнтів у сервісному центрі.
- **Імітаційне моделювання.** Імітаційне моделювання — це математичний підхід, який передбачає створення комп'ютерної моделі логістичної системи та її використання для тестування різних сценаріїв та оптимізації продуктивності системи.
- **Евристика та метаевристика.** Евристика – це методи розв'язання проблем, які використовують практичні, емпіричні підходи для пошуку рішень. Метаевристики — це стратегії вищого рівня, які спрямовують пошук рішень. Ці методи можна використовувати для вирішення складних проблем логістики, де інші математичні підходи можуть бути непрактичними, як правило, за рахунок зниження точності результатів.
- **Генетичні алгоритми.** Генетичні алгоритми — це тип метаевристики, який використовує принципи біологічної еволюції для оптимізації рішень. Їх можна використовувати для вирішення проблем логістики, таких як маршрутизація транспортних засобів і планування.
- **Аналіз рішень.** Аналіз рішень — це математичний підхід, який передбачає аналіз можливих результатів різних рішень і вибір того, що максимізує очікувану корисність. Його можна використовувати для оптимізації логістичних рішень, таких як управління запасами та вибір постачальника.

Це лише деякі з математичних підходів, які можна використовувати для вирішення логістичних задач. Конкретний підхід залежатиме від

характеру проблеми та наявних даних. Для задачі поставленою даною роботою використовуватимемо саме підхід лінійного програмування, а саме буде розглянута задача комівояжера.

Оскільки завдання даної роботи заключається у використанні, обробці, форматуванні та навчанні лінгвістичних моделей доволі великих потоків даних, має сенс розглянути сучасні проблеми розподілу потоків даних.

## 1.2. Сучасні проблеми розподілу потоків даних

Розподіл потоків даних є критично важливим аспектом сьогодення, з проблемами якого, можна з легкістю зіткнутися розв’язуючи задачі логістики, оптимізації сховищ даних, бізнес та дата аналізу, та інше. У цих сферах використовують дані, що зберігаються та обробляються на кількох вузлах або серверах, які можуть бути географічно розподілені по різних кутках світу. Проблеми з розподілом потоку даних виникають, саме коли дані потрібно ефективно та результативно розподіляти між цими вузлами, забезпечуючи послідовність та доступність даних.

У цілому можна виділити наступні проблеми розподілу потоків даних:

1. **Нерівномірний розподіл даних.** Нерівномірний розподіл даних може відбуватися через низку причин, наприклад алгоритми розподілу даних, які не є оптимальними для розподілу даних або топології мережі, дані з нерівномірними шаблонами доступу, або збій вузлів у мережі. Ці проблеми можуть призвести до сповільнення часу обробки та зниження пропускної здатності на деяких вузлах, тоді як інші залишаються бездіяльними. Щоб вирішити цю проблему, можна використовувати методи балансування навантаження для рівномірного розподілу даних між вузлами, щоб кожен вузол обробляв однакову

кількість даних. Балансування навантаження може здійснюватися статично або динамічно, коли розподіл даних між вузлами здійснюється під час налаштування системи або під час виконання на основі поточного робочого навантаження.[4][5]

2. **Асиметрія даних.** Асиметрія даних може призвести до нерівномірного розподілу даних, оскільки деякі вузли можуть отримувати набагато більше даних, ніж інші через нерівномірний розподіл певних значень або атрибутів. Це може призвести до проблем, подібних до нерівномірного розподілу даних, наприклад повільної обробки та зниження пропускної здатності. Для вирішення цієї проблеми можна використовувати такі методи, як розподіл асиметричних даних або балансування навантаження з урахуванням викривлення. Наприклад, дані можна розділити на основі спотворених атрибутів, щоб кожен вузол отримував збалансований набір даних.[4]
3. **Розміщення даних.** Розміщення даних відноситься до проблеми визначення місця зберігання даних у розподіленій системі. Це може бути проблемно, оскільки різні варіанти зберігання можуть мати різні характеристики продуктивності залежно від типу даних, що зберігаються. На вибір розміщення даних також можуть впливати такі фактори, як моделі доступу до даних, розташування даних і розмір даних. Щоб вирішити цю проблему, дані можна зберігати різними способами, наприклад сегментацією (поділ даних на менші частини та їх зберігання на різних вузлах), реплікацією (зберігання кількох копій тих самих даних на різних вузлах) або гібридними підходами, які використовують поєднання цих технік.
4. **Реплікація даних.** Реплікація даних на кількох вузлах може допомогти підвищити продуктивність і надійність, але також може створити проблеми з узгодженістю даних, якщо оновлення не синхронізуються належним чином. Для забезпечення узгодженості даних між вузлами

можна використовувати такі методи реплікації даних, як «головний-підлеглий», «декілька головних» і реплікація на основі кворуму. У реплікації головний-підлеглий один вузол діє як основний/головний вузол, і всі оновлення виконуються на цьому вузлі, який потім реплікує зміни на інші вузли. У реплікації з декількома майстрами кілька вузлів можуть робити оновлення одночасно, а зміни поширюються на інші вузли. Для реплікації на основі кворуму потрібен кворум (більшість вузлів), щоб узгодити оновлення, що гарантує, що всі вузли мають узгоджені дані. Послідовне хешування також можна використовувати для визначення того, які вузли повинні зберігати копію даних, що може допомогти рівномірно розподілити дані в мережі.[4][5]

Оскільки обсяг даних продовжує зростати, а потреба в розподілених обчислювальних системах зростає, проблеми розподілу потоку даних залишатимуться критичною областю досліджень і розробок. В практичній частині даної роботи не раз стикалися з даними проблемами, особливо при навчанні лінгвістичних моделей, де залучалося велика кількість анотованих даних, що без належного розуміння проблем розподілу потоків даних, в край би ускладнило цей процес зменшивши в рази ефективність та швидкість.

Рухаючись далі, приступаємо до теорії пов’язаної напряму з задачею поставленою даною роботою. Працюючи з класифікацією лінгвістичних змінних або у випадках, коли оцінити деякий параметр тяжко за допомогою звичайною чіткою логікою, має сенс використати апарат нечіткої логіки, про це і йдеться в наступному розділі.

## 1.3. Нечіткі задачі оптимізації

Більшість наших традиційних інструментів для формального моделювання, міркувань і обчислень є чіткими, детермінованими та

точними. Під чіткістю ми маємо на увазі дихотомічний, тобто тип «так чи ні», а не тип «більш-менш». У звичайній подвійній логіці, наприклад, твердження може бути істинним або хибним і не мати нічого між ними. У теорії множин елемент може як належати до множини, так і ні; а в оптимізації рішення або можливе, або ні[6].

Але далеко не всі моделі можна описати дихотомічним шляхом, нечіткість можна знайти в багатьох сферах повсякденного життя, таких як техніка, медицина, метеорологія, виробництво тощо. Однак це особливо часто зустрічається в сферах, де важливі людські судження, оцінка та рішення. Це сфери прийняття рішень, міркування, навчання, тощо. Деякі причини цієї нечіткості вже згадувалися. Інші полягають у тому, що більшість нашого повсякденного спілкування проходить з використанням «природної мови», і значна частина нашого мислення також здійснюється таким шляхом. В природній мові значення слів дуже часто розпливчасте. Значення слова може бути навіть чітко визначеним, але при використанні слова, як анотації для набору даних, часто тяжко чітким шляхом описати приналежність різних об’єктів до цього слова, тут і з’являється поняття нечіткості та поняття міри приналежності до тої чи іншої нечіткої множини, саме це і вивчає теорія нечітких множин[6].

### 1.3.1. Теорія нечітких множин

Теорія нечітких множин просувалася різними шляхами в багатьох сферах суспільства протягом останніх 60 років. Застосування цієї теорії можна знайти, наприклад, у штучному інтелекті, інформатиці, технологіях управління, теорії прийняття рішень, експертних системах, логіці, дослідженні операцій, розпізнаванні образів і робототехніці.

В цих галузях інформація, що використовується може бути суб'єктивною, а її відображення в мові людей, як правило, містить багато невизначеностей таких, як "багато", "мало", "приблизно", які не мають

аналогів в мові математики. Тому опис цієї інформації засобами традиційної математики сильно ускладнює математичну модель. Таким чином, для подальшого застосування математичних методів аналізу та дослідження все більш складних систем, з'явилася потреба в створенні нового математичного апарату, що дав би можливість формально описувати нечіткі поняття, якими користується людина, описуючи свої бажання, цілі та уявлення.

Таким апаратом стала теорія нечітких множин, створена Л. Заде, перша фундаментальна праця якого була опублікована ще в 1965 р. З тих часів теорія нечітких множин значно розвинулася, з’явилося багато наукових робіт, присвячених теоретичним та практичним аспектам цього напрямку. Тож, що таке нечітка множина говорячи математичною мовою?

Нечіткою множиною А, яка задана на універсальній множині $X$ називається сукупність пар $(x, \mu_A(x))$ де $x \in X$, а $\mu_A$ – функція $x \to [0; 1]$[7], яка має назву функції приналежності множини А (Рис. 1).

Звичайні множини, у свою чергу, утворюють підклас нечітких множин. Як видно на (Рис. 2), функцією приналежності множини $B \subseteq X$ є її характеристична функція $\mu_B(x)$[7]:

$$\mu_B(x) = \begin{cases} 1, \text{ якщо } x \in B \\ 0, \text{ якщо } x \notin B \end{cases}$$

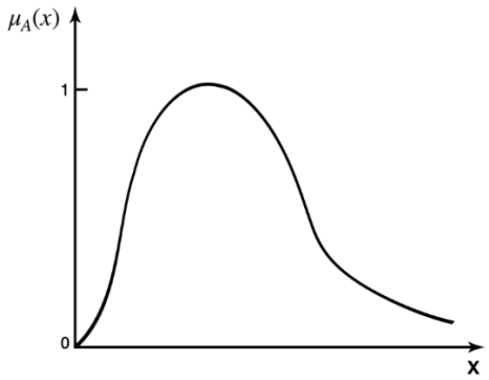


Рис. 1

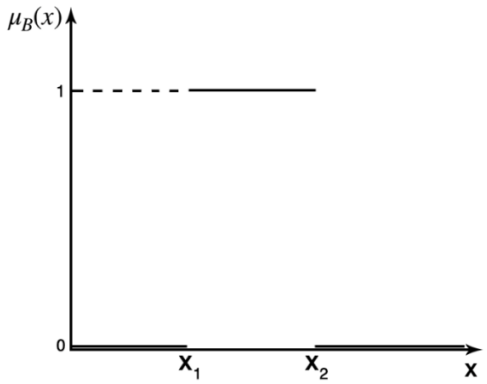


Рис. 2

Тепер розглянемо деякі базові операції над нечіткими множинами, які будуть потрібні в подальшому[7].

Об'єднанням нечітких множин A і B в $X$ називається нечітка множина $C = A \cup B$ з функцією приналежності вигляду:

$$\mu_c(x) = \max\{\mu_A(x); \mu_B(x)\}$$

Перетином нечітких множин A і B в $X$ називається нечітка множина $C = A \cap B$ з функцією приналежності вигляду:

$$\mu_c(x) = \min\{\mu_A(x); \mu_B(x)\}$$

Різницею нечітких множин A і B в $X$ називається нечітка множина $C = A \backslash B$ з функцією приналежності вигляду:

$$\mu_C(x) = \begin{cases} \mu_A(x) - \mu_B(x), \text{ якщо } \mu_A(x) \geq \mu_B(x) \geq 0 \\ 0, \text{ в іншому випадку} \end{cases}$$

Декартовим добутком $A_1 \times A_2 \times \ldots \times A_n$ нечітких множин $A_i$ в $X_i, i = 1, n$ називається нечітка множина в декартовому добутку $X_1 \times X_2 \times \ldots \times X_n$ з функцією приналежності вигляду:

$$\mu_A(x) = \min\{\mu_{A_1}(x_1); \mu_{A_2}(x_2), \ldots, \mu_{A_n}(x_n)\}$$

### 1.3.2. Нечітке лінійне програмування

Тепер, розібравшись з нечіткістю, можна приступити до нечітких задач оптимізації, а саме до класичної задачі даної теми, нечітке лінійне програмування, що якраз є підґрунтям до розв’язання задачі комівояжера. Але перед цим треба розібратися ще з одним поняттям, теорією прийняття рішень в класичній та нечіткій логіці.

У класичній теорії прийняття рішень, рішення можна охарактеризувати набором альтернатив рішень (простір рішень); сукупність станів природи (простір станів); відношення, що призначає кожній парі рішення та стан результату; і, нарешті, функція корисності, яка

впорядковує результати відповідно до їх бажаності. Приймаючи рішення в умовах визначеності, особа, що приймає рішення (ОПР), знає, якого стану очікувати, і вибирає альтернативу рішення з найвищою корисністю. Приймаючи рішення під ризиком, він точно не знає, який стан настане; він знає лише функцію ймовірності станів. В такому виипадку прийняти рішення стає складніше. Ми обмежимо нашу увагу прийняттям рішень в умовах визначеності. У цьому випадку модель прийняття рішень є несиметричною: простір рішень можна описати перерахуванням або низкою обмежень. Функція корисності впорядковує простір рішень через взаємозв'язок результатів з альтернативними рішеннями. Отже, ми можемо мати лише одну функцію корисності, яка забезпечує порядок, але ми можемо мати кілька обмежень, що визначають простір рішень. На рис. 3 зображений приклад моделі прийняття рішень при умові чіткості та визначеності.

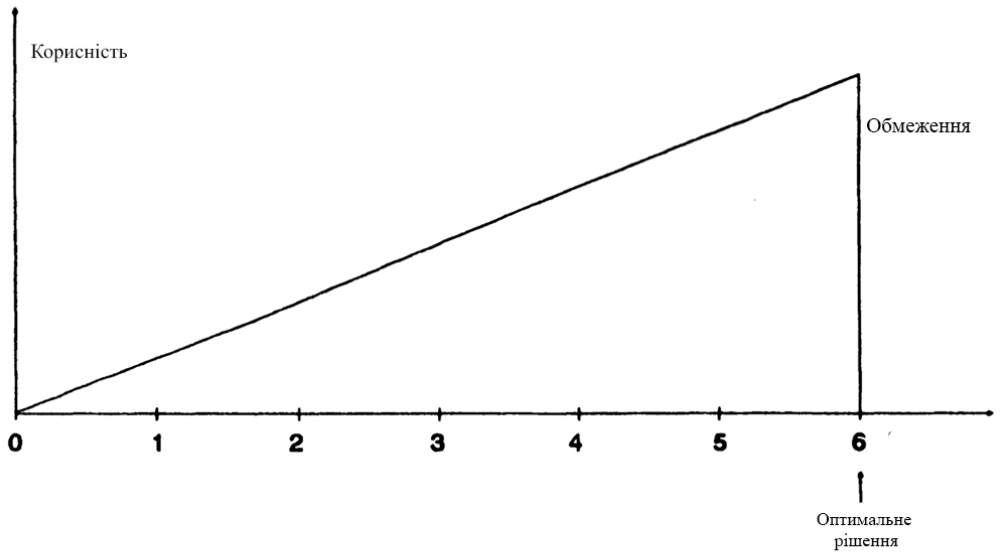


Рис. 3

У 1970 році Беллман і Заде розглянули цю класичну модель рішення і запропонували модель прийняття рішень у нечіткому середовищі, яка послужила відправною точкою для більшості авторів теорії «нечітких» рішень. Вони розглядають ситуацію прийняття рішень в умовах визначеності, в якій цільова функція, а також обмеження є нечіткими, і

стверджують наступне: нечітка цільова функція характеризується функцією приналежності, як і обмеження. Оскільки ми хочемо оптимізувати цільову функцію, а також обмеження, рішення в нечіткому середовищі визначається за аналогією з чітким середовищем як вибір дій, які одночасно задовольняють цільову функцію(-ї) та обмеження. Таким чином, «рішення» в нечіткому середовищі можна розглядати як перетин нечітких обмежень і нечіткої цільової функції. Таким чином, зв'язок між обмеженнями та цільовими функціями в нечіткому середовищі є повністю симетричним, тобто більше немає різниці між першим і другим. На рис. 4 зображений приклад такої моделі прийняття рішень при умові нечіткості.

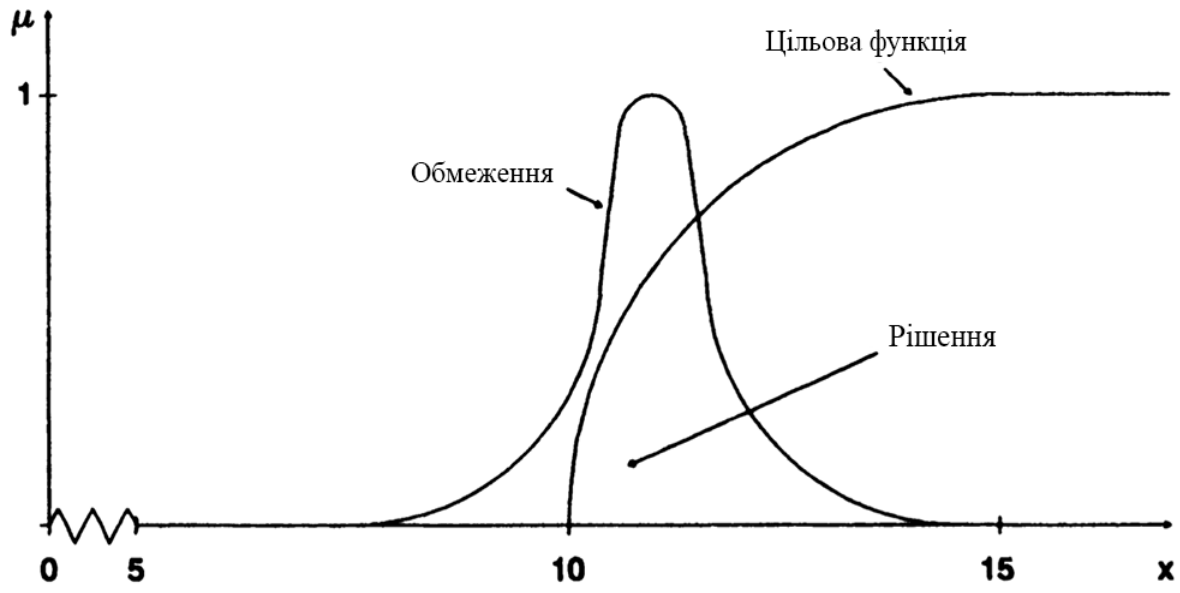


Рис. 4

Моделі лінійного програмування слід розглядати як особливий вид моделі рішень розглянутих вище: простір рішень визначається обмеженнями; функція корисності визначається цільовою функцією; а тип рішення – прийняття рішеннь в умовах визначеності. Класичну модель лінійного програмування можна сформулювати як:

$$\text{максимізувати} \quad f(x) = c^T x$$

$$\text{така, що} \qquad Ax \leq b$$

$$x \geq 0$$

$$\text{при} \quad c, x \in \mathbb{R}^n, b \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n} \qquad (1)$$

Тепер відступимо від класичних припущень, що всі коефіцієнти $A, b$ і $c$ є чіткими числами, що $\leq$ мається на увазі в чіткому сенсі, і що «максимізація» є суворим імперативом.

Якщо ми припустимо, що ЛП-рішення має прийматися в нечітких середовищах, існує досить багато можливих модифікацій моделі (1). Перш за все, особа, що приймає рішення, може насправді не хотіти фактично максимізувати або мінімізувати цільову функцію. Швидше, він або вона може захотіти досягти певних прагнень, які навіть не можна чітко визначити. Таким чином, як приклад, ОПР може захотіти «значно покращити поточну ситуацію з витратами» та інше.

По-друге, обмеження можуть бути нечіткими різними способами: знак $\leq$ може не означати щось в строго математичному сенсі. Це може статися, коли, наприклад, обмеження представляють сенсорні відчуття (смак, колір, запах тощо), які не можуть бути адекватно наближені чітким обмеженням. Звичайно, коефіцієнти векторів $b$ або $c$ або самої матриці $A$ можуть мати нечіткий характер бо вони є нечіткими за своєю природою, або тому, що їх сприйняття є нечітким.

Нарешті, роль обмежень може відрізнятися від ролі обмежень у класичному лінійному програмуванні, де порушення будь-якого окремого обмеження будь-якою величиною робить рішення нездійсненним. ОПР може прийняти невеликі порушення обмежень, але також може надавати різні (чіткі або нечіткі) степені важливості порушенням різних обмежень. Можна визначити такі типи нечітких моделей лінійного програмування[6][8]:

1. Випадок, де нечіткі множини визначаються над цільовою функцією та/або обмеженнями. Коефіцієнти цільової функції та/або права частина обмежень є нечіткими числами.

2. Нечіткі множини визначаються над цільвими змінними. Коефіцієнти цільової функції та/або права частина обмежень є чіткими числами.
3. Задіяні як нечіткі множини, так і нечіткі цільові змінні. Коефіцієнти цільової функції та/або права частина обмежень є чіткими числами.
4. Цільова функція нечітка, а обмеження чіткі. Цільові змінні є чіткими.
5. Цільова функція є чіткою, а обмеження нечіткими. Цільові змінні є чіткими.
6. І цільова функція, і обмеження є нечіткими. Цільові змінні є чіткими.
7. І цільова функція, і обмеження є нечіткими. Цільові змінні також є нечіткими.

Кожен тип моделі нечіткого лінійного програмування вимагає різних методів і технік для розв'язку. Вибір відповідної моделі нечіткого лінійного програмування залежить від характеру проблеми та наявної інформації. У випадку практичної задачі поставленою даною роботою, дефазифікація проходить до використання задачі комівояжера, а для визначення часових оцінок кожного з етапів маршруту використовується теорія нечіткого регулятора, про що буде згадуватися більш детально в наступних розділах.

## 1.4. Моделі та аналіз постановок задачі комівояжера

Задача комівояжера це класична задача оптимізація в комп'ютерній науці, дослідженні операцій та нечіткій оптимізації. Дану задачу легко сформулювати: комівояжер, починаючи з одного міста, хоче відвідати кожне з $n-1$ інших міст і повернутися на початок. У якому порядку він повинен відвідувати міста, щоб мінімізувати загальну відстань, яку він пройде? Під «відстанню» в даній задачі може виступати час, вартість шляху між містами (вузлами графа) або якась інша міра за бажанням. Передбачається, що відстань або витрати між усіма парами міст відомі.

Проблема стала відомою, адже вона поєднує в собі легкість формулювання з труднощами розв'язання. Причому задача являється повністю обчислювальною, адже відповідь очевидно існує. Є (n-1)! можливих шляхів, один або кілька з яких повинні мати мінімальну вартість.

Походження даної задачі невідоме. Німецький довідник комівояжера 1832 року вперше згадує проблему та містить приклади шляхів по Німеччині та Швейцарії, але не охоплює математику проблеми.

Перше математичне формулювання було зроблено в 1800-х роках В. Р. Гамільтоном і Томасом Кіркманом. Ікосіанська гра Гамільтона була розважальною головоломкою, яка базувалася на пошуку гамільтонового циклу, який фактично є рішенням задачі комівояжера на графі.

Один із перших ефективних методів розв'язку задачі комівояжера була методика запропонована Данцигом, Фулкерсоном та Джонсоном, які виразили дану задачу на базі цілочисельного лінійного програмування та розробили метод січної площини для її вирішення. Формулювання було наступним[10]:

Позначимо міста $1, \dots, n$ і дамо визначення:

$$x_{ij} = \begin{cases} 1 & \text{шлях прямує з міста } i \text{ в місто } j \\ 0 & \text{в іншому випадку} \end{cases}$$

Візьмемо $c_{ij} > 0$ як відстань від міста $i$ до міста $j$. Тоді задачу комівояжера можна записати як наступну задачу цілочисельного лінійного програмування:

$$\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij}$$

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1 \quad j = 1, \dots, n;$$

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1 \quad i = 1, \dots, n;$$

$$\sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \leq |Q| - 1 \quad \forall Q \subsetneq \{1, \dots, n\}, |Q| \geq 2.$$

Задач комівояжера – NP-повна. Щодо розв’язання даної задачі, існує велика кількість різних підходів, у цілому вони поділяться на точні методи та евристичні методи. Точні методи — це алгоритми, які гарантують знаходження оптимального розв’язку задачі за певний час або певні ресурсові обмеження. Зазвичай вони передбачають математичне формулювання проблеми та її розв’язання за допомогою методів оптимізації, таких як лінійне програмування, динамічне програмування або метод гілок та меж. Однак, точні методи часто є дорогими з точки зору обчислень і можуть бути застосовані лише до задач невеликого масштабу.

З іншого боку, евристичні методи — це алгоритми, які не гарантують знаходження оптимального розв’язку, а натомість спрямовані на швидкий пошук майже оптимального розв’язку. Тут зазвичай використовують підходи «спроб і помилок», наприклад випадковий пошук або жадібний алгоритм, щоб швидко дослідити простір рішень і знайти перспективне рішення. Евристичні методи є більш гнучкими і можуть бути застосовані до проблем більшого масштабу, але розв’язок, який вони пропонують, може бути не оптимальним. З усієї цієї групи евристичних методів уваги заслуговують також ті, що імітують біологічні процеси: мурашиний алгоритм та генетичний алгоритм.

Отже підсумовуючи сказане вище, існує багато різних методів розв’язання задачі комівояжера, деякі страждають від обчислювальної неефективності, інші не мають гарантій, що розв’язок буде оптимальним, а ще інші вимагають інтуїтивних суджень, які важко було б запрограмувати. Для досягнення мети даної роботи, а особливо програмної реалізації, евристичні методи не підходять якраз через їх не точність в розрахунках, тож було приділено більшу увагу саме точним методам.

Найбільш легким рішенням було б спробувати всі перестановки міст (впорядковані комбінації) і побачити, яка з них найдешевша (за допомогою брутфорс алгоритму). Час роботи для цього підходу лежить у поліноміальному коефіцієнті $\Theta(n!)$, факторіалі кількості міст, тому ця методика стає непрактичною навіть для лише 20 міст.

Одним із найпопулярніших і в той же час одним із найефективніших точних методів для розв'язання задачі комівояжера є метод динамічного програмування. Метод був запропонований в 1962 році незалежно Белманом, та Хелдом і Карпом для вирішення задачі комівояжера та має експоненціальну часову складність $\Theta(2^n n^2)$, що значно краще, ніж суперекспоненціальна складність $\Theta(n!)$ брутфорс алгоритму, однак для зберігання всіх обчислених значень потрібно $\Theta(n2^n)$ простору, у той час як брутфрс алгоритм потребує лише $\Theta(n^2)$ простору для зберігання самого графа. Даний алгоритм починається з визначення двовимірної матриці (таблиці) $A_{i,j}$, де $a_{i,j}$ представляє собою найкоротшу можливу відстань, починаючи з міста $i$, відвідуючи всі міста точно один раз і повертаючись до міста $j$. Таблиця ініціалізується так, що $a_{i,j}$ має значення для всіх i ≠ j, а $a_{i,i}$= 0. Далі алгоритм використовує вкладений цикл для заповнення таблиці, враховуючи всі можливі комбінації проміжних міст. Для кожної комбінації алгоритм обчислює мінімальну довжину шляху, враховуючи всі можливі проміжні міста між $i$ та $j$, оновлюючи значення $A_{i,j}$, якщо знайдено коротший тур. Алгоритм продовжує цей процес, доки не буде заповнена вся таблиця, після чого можна знайти мінімальну довжину шляху, враховуючи всі можливі кінцеві міста[11].

Не зважаючи на всі плюси та легкість розуміння, метод динамічного програмування не так гарно справляється з задачам з багатьма містами, порівнюючи з методом гілок та меж. Цей метод був вперше запропонований Айлзою Ленд і Елісон Дойгом в 1960 році та з тих пір став найбільш часто

використовуваним інструментом для вирішення NP-складних задач оптимізації, а одна із варіацій цього методу до сьогодні тримає рекорд по кількості міст для розрахунку задачі комівояжера. Мінімальний розв’язок задачі комівояжера даним методом можна знайти, якщо взяти набір усіх можливих розв’язків і розбити його на непересічні підмножини. Для кожної підмножини знаходять нижню межу значення (цільової функції) для мінімального розв’язку в цій підмножині. Підмножину з найменшою нижньою межею розбивають далі, і нижні межі знову знаходять для нових підмножин. Процес продовжується таким чином і це зручно візуалізувати як розгалужений процес або дерево. Точки розгалуження (вузли) представляють підмножини, які були розбиті, а кінцеві точки (кінцеві вузли) представляють поточну колекцію підмножин. Процес зупиняється, коли знайдено підмножину, яка містить один розв’язок, значення якого менше або дорівнює нижнім межам для всіх кінцевих вузлів. Оскільки термінальні вузли, зібрані разом, завжди містять усі розв’язки, мінімальний розв’язок задачі комівояжера ідентифіковано.[12]

# Розділ 2. Розробка методів розв'язання нечіткої задачі комівояжера на основі технологій машинного навчання

## 2.1. Технологія використання машинного навчання для задачі розпізнавання іменованих сутностей

Обробка природної мови (NLP) — це підгалузь штучного інтелекту, яка фокусується на взаємодії між комп'ютерами та людьми за допомогою природної мови. NLP набуло величезної популярності в останні роки завдяки своєму великому потенціалу в різних галузях, включаючи медицину, фінанси, логістика та інше. Одним із найважливіших завдань NLP є розпізнавання іменованих сутностей (NER), яке передбачає ідентифікацію та категоризацію іменованих сутностей в неструктурованих текстових даних. Іменовані або анотовані об'єкти – це конкретні слова чи фрази, які стосуються людей, місць, організацій, дат, часу, кількості та інших типів інформації.[13]

Основною метою NER є витяг і класифікація іменованих сутностей із текстових даних у попередньо визначені категорії або типи. Зазвичай це робиться за допомогою моделей машинного навчання, які навчаються на великих наборах даних анотованого тексту, де кожна названа сутність позначена відповідною міткою або категорією.

В практичній частині даної роботи було використано набір інструментів MITIE для навчання моделі розпізнавання класифікованих слів з тестів. Даний інструмент використовує різні алгоритми машинного навчання, зокрема метод опорних векторів, для NER завдань і класифікації тексту. Про цей метод більш детально далі.

### 2.1.1. Метод опорних векторів

Ідея методу опорних векторів (SVM) виникла в 1960-х роках під час роботи Вапника та його колег із теорії статистики. Однак лише в 1990-х роках метод опорних векторів набув популярності як метод машинного навчання для задач класифікації та регресії.

Перший алгоритм методу опорних векторів був запропонований Бозером, Гайоном і Вапником у 1992 році, який використовував лінійне ядро для пошуку оптимальної гіперплощини, яка розділяє дані з максимальним розділенням. Після цього в 1997 році Шелькопф і Смола представили трюк з ядром, який дозволив опорно-векторним машинам обробляти нелінійно розділені дані, перетворюючи їх у простір з більшою вимірністю.

У 1995 році Вапник і Кортес представили алгоритм опорної векторної мережі (SVN), який розширив опорно-векторні машини для обробки завдань класифікації з більш ніж двома класами. Того ж року оригінальний алгоритм метод опорних векторів був опублікований у журналі Machine Learning, що привернуло значну увагу дослідницького співтовариства.[14]

Відтоді метод опорних векторів широко вивчався та застосовувався в різних областях, включаючи класифікацію тексту, класифікацію зображень, біоінформатику та фінанси. Метод опорних векторів також було розширено для виконання інших завдань, таких як регресія та кластеризація, і поєднано з іншими алгоритмами машинного навчання, такими як випадкові ліси та глибинне навчання, для покращення продуктивності.

Основна ідея методу опорних векторів полягає в тому, щоб знайти гіперплощину (пряму в двох вимірах, площину в трьох вимірах і гіперплощину у більших вимірах), яка розділяє класи вхідних даних з максимальною відстанню до найближчих точок тренувальних даних будь-якого з класів. Завдяки максимізації розділення опорно-векторна машина

може створити межу рішення, яка є стійкою до шуму та добре узагальнює нові дані.[14][15]

Метод опорних векторів працює шляхом перетворення вхідних даних у багатовимірний простір ознак за допомогою функції ядра, яка дозволяє лінійно розділяти дані в перетвореному просторі, навіть якщо вони не лінійно розділяються у вихідному просторі. Потім опорно-векторна машина знаходить гіперплощину, яка розділяє класи з максимальною відстанню у перетвореному просторі (рис. 5).

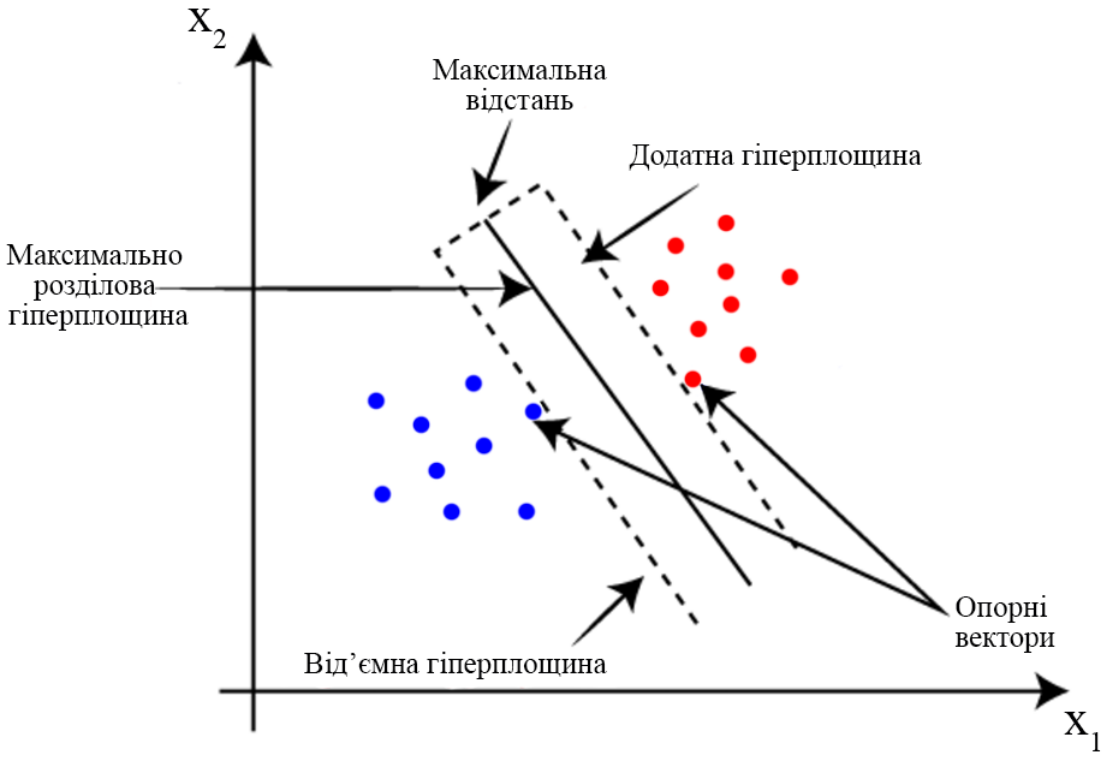


Рис. 5

Однією з переваг методу опорних векторів є те, що він менш схильний до перенавчання, ніж інші алгоритми машинного навчання. Метод опорних векторів також ефективний, коли кількість ознак значно перевищує кількість прикладів, що часто буває в реальних програмах. Однак метод опорних векторів може бути обчислювально дорогим і чутливим до вибору функцій і параметрів ядра.

Тепер розглянемо більш детально інструменти за допомогою яких навчали NER-модель для практичної частини даної роботи.

### 2.1.2. Бібліотека MITIE для розпізнавання анотованих сутностей

MITIE (MIT Information Extraction) — це бібліотека програмного забезпечення обробки природної мови, розроблена лабораторією Лінкольна Массачусетського технологічного інституту. Вона надає набір інструментів і алгоритмів для різних завдань обробки природної мови, таких як розпізнавання іменваних сутностей (NER), тегування частин мови та класифікація тексту.

MITIE використовує розподільні вкладання слів для зменшення розмірності та підвищення продуктивності, викристовує умовні випадкові поля та структуровані опорно-векторні машини для вивчення синтаксичних зв’язків, а також використовує автоматичну оптимізацію гіперпараметрів для полегшення налаштувань користувачем. MITIE побудовано на основі високопродуктивної бібліотеки машинного навчання Dlib (написана на C++), містить інтерфейси для C, C++, Java, R і Python, і її легко навчати на нових наборах даних, наприклад тих, що були використані в практичній частині даної роботи.

Однією з ключових особливостей MITIE є здатність виконувати з’єднане вилучення (joint entity extraction) та класифікацію об’єктів. Це означає, що є можливість не тільки ідентифікувати сутності в тексті, але й класифікувати їх за попередньо визначеними категоріями, такими як особа, організація, локація тощо.

В розділі 3, буде наглядно показано використання моделі навченої за допомогою MITIE бібліотеки, а код знаходиться в додатках до роботи. Тепер розглянемо навчальну вибірку на якій була натренована дана модель.

### 2.1.3. Навчальна вибірка для моделі розпізнавання іменованих сутностей

Майже у всіх моделях обробки природньої мови (NLP), щоб натренувати гарну модель (F1 оцінка 85+) потрібно мати велику кількість анотованих даних, розпізнавання іменованих сутностей не виняток в цьому питанні. Одній людині анотувати сотні тисяч речень неймовірно складно, але гарно, що існує українська команда Lang-uk, які захоплені NLP та інтелектуальною обробкою тексту. Команда Lang-uk базується на єдиних засадах та принципах і орієнтована на розвиток нових проектів збору українських мовних корпусів та інших текстових ресурсів. Один з їх відкритих до публічного доступу проектів є корпус NER-анотацій, що містить 229 текстів з українського браунівського корпусу на 217 381 токенів з 6 751 розмічених іменованих сутностей. З яких майже 1500 проіменованих локацій, саме та класифікація, що нам потрібна для моделі, тож саме цей ресурс і був використаний, як навчальна вибірка для практичної частини даної роботи.

Кожний з оброблених командою lang-uk текстів складається з двох файлів:

- файл із розширенням tok.txt складається з токенізованих текстів
- файл із розширенням tok.ann містить NER анотаціїї для цих текстів у форматі Brat Standoff Format (кожен рядок файлу містить 3 анотації, розділені табуляцією: номер анотації, початковий і кінцевий індекси в токенізованому тексті і сама сутність, приклад зображено на рис. 6).

```
T1      PERS 263 268    Чумак
T2      LOC 395 413     Черкаській області
T3      LOC 446 457     Херсонській
T4      ORG 510 514     УТМР
T5      PERS 657 662    Чумак
T6      LOC 784 791     В'юнище
T7      LOC 801 814     Олександрівки
T8      PERS 968 981    Андріан Чумак
T9      PERS 1349 1354  Чумак
T10     PERS 1657 1662  Чумак
T11     LOC 2058 2065   В'юнище
T12     LOC 2396 2415   Херсонській області
```

Рис. 6

## 2.2. Методика добуття інформації для вирішення задачі

Одною з ключових задач даної роботи було, зібрати якомога більше інформації з новинних ресурсів, для подальшого аналізу та зчитування інформації про ремонти доріг на вибраних шляхах. Для цього була використана технологія вебскрепінгу. Вебскрепінг можна використовувати для різних цілей, наприклад для аналізу даних, дослідження ринку, аналізу конкурентів і порівняння цін. Його також можна використовувати для більш складних програм, таких як машинне навчання та штучний інтелект. Процес вебскрепінгу передбачає доступ до HTML-коду веб-сайту та вилучення з нього потрібної інформації. Витягнуті дані часто зберігаються в структурованому форматі, наприклад CSV або JSON, і можуть бути додатково оброблені або проаналізовані за допомогою інших інструментів або програмного забезпечення.

Існує багато різних інструментів і методів, які використовуються для вебскрепінгу, починаючи від простого ручного пошуку даних за допомогою веб-браузерів і копіювання цих даних до більш прогресивних автоматизованих підходів, таких як використання веб-бібліотек або фреймворків, таких як BeautifulSoup, Scrapy або Selenium.

Для вилучення інформації з новинних ресурсів для практичної задачі даної роботи, була використана бібліотека BeautifulSoup. Бібліотека

полегшує навігацію та пошук у складних HTML і XML документах, а також дозволяє користувачам отримувати дані на основі різних атрибутів, таких як клас, ідентифікатор, тег тощо. BeautifulSoup також надає інструменти для зміни та створення HTML і XML документів. Таким чином і була написана частина головної програми, яка знаходить та зчитує усі релевантні новини за запитом “Ремонт доріг Київ”. Більш детально програмна реалізація описана в розділі 3, а код програми в додатках до роботи.

## 2.3. Метод розв’язання задачі комівояжера з урахуванням нечітких оцінок часу переміщення

В цілому практичну задачу можна описати наступним чином:

1. Навчання моделі розпізнавання іменованих сутностей (описано в розділі 2.1)
2. Зчитування тексту з новинних ресурсів, для подальшого використання на ньому натренованої моделі (описано в розділі 2.2)
3. Ну і вже отримавши інформацію від комівояжера (всі можливі шляхи між місцями призначення), з’являється задача описана в даній роботі, використовуючи інформацію з новинних джерел обрати правильний шлях для комівояжера.

В цьому розділі якраз і буде розказано про третій пункт. Він містить дві головні задачі: з отриманих даних визначити часові оцінки кожного з можливих шляхів; розв’язати задачу комівояжера з цими часовими оцінками, як вхідними даними.

Зі всіх можливих методів, було вирішено використати теорію нечіткого регулятора для визначення можливих часових затримок на тому чи іншому шляху, зважаючи на кількість згадувань різних локацій в текстах та міру впевненості. Про це далі більш детально.

### 2.3.1. Методика формування нечітких часових оцінок

Метою теорії нечіткого керування (FLC) є управління складними процесами за допомогою людського досвіду. Таким чином, системи нечіткого керування та експертні системи мають схоже походження. Однак не варто нехтувати їх важливими відмінностями. У той час як експертні системи намагаються використовувати невизначені знання, отримані від експерта для підтримки користувачів у певній області, системи нечіткого керування розроблені для контролю вже саме технічних процесів. Більше того, методи нечіткого керування перейшли від базового перекладу людського досвіду в правила керування, до підходів більш орієнтованиих на інженерію, де мета полягає в тому, щоб налаштувати контролер, поки поведінка не стане компетентною, незалежно від будь-якої людської поведінки.[16]

Звичайні (нечіткі) системи керування проектуються за допомогою фізичних моделей розглянутого процесу. Розробка відповідних моделей займає багато часу та вимагає серйозної теоретичної підготовки інженера. Оскільки моделювання є процесом абстракції, модель завжди є спрощеною версією процесу. Помилки виправляються за допомогою шумових сигналів, додаткових моделей станів, тощо. Однак багато процесів можуть керуватися людьми без допомоги жодної моделі, також є процеси, якими неможливо керувати за допомогою звичайних систем керування, але є доступними для людського керування – наприклад, більшість людей з водійськими правами можуть керувати автомобілем без будь-якої моделі управління. Формалізація досвіду оператора методами нечіткої логіки була основною ідеєю теорії нечіткого керування.[17]

Основна ідея цього підходу полягає в тому, щоб включити «досвід» людини-оператора процесу в дизайн контролера. З набору лінгвістичних правил, які описують стратегію керування оператором, будується алгоритм керування, де слова визначаються як нечіткі множини. Головними

перевагами цього підходу є можливість застосування «емпіричного правила» (rule of thumb) до досвіду, інтуїції, евристики, а також те, що цей. метод не потребує моделі процесу.

Нечіткі регулятори — це спеціальні системи прямого керування (DDG), які використовують правила для явного моделювання знань про процеси. Замість проектування алгоритмів, які явно визначають керуючу дію як функцію вхідних змінних контролера, розробник нечіткого регулятора пише правила, які пов’язують вхідні змінні з керуючими змінними за допомогою лінгвістичних змінних.

Основні складові нечіткого регулятора: база правил, блок фазифікації, обчислювальний блок, блок дефазифкації. На рисунку 7 зображено загальний приклад так званого нечіткого регулятора «Mamdani».

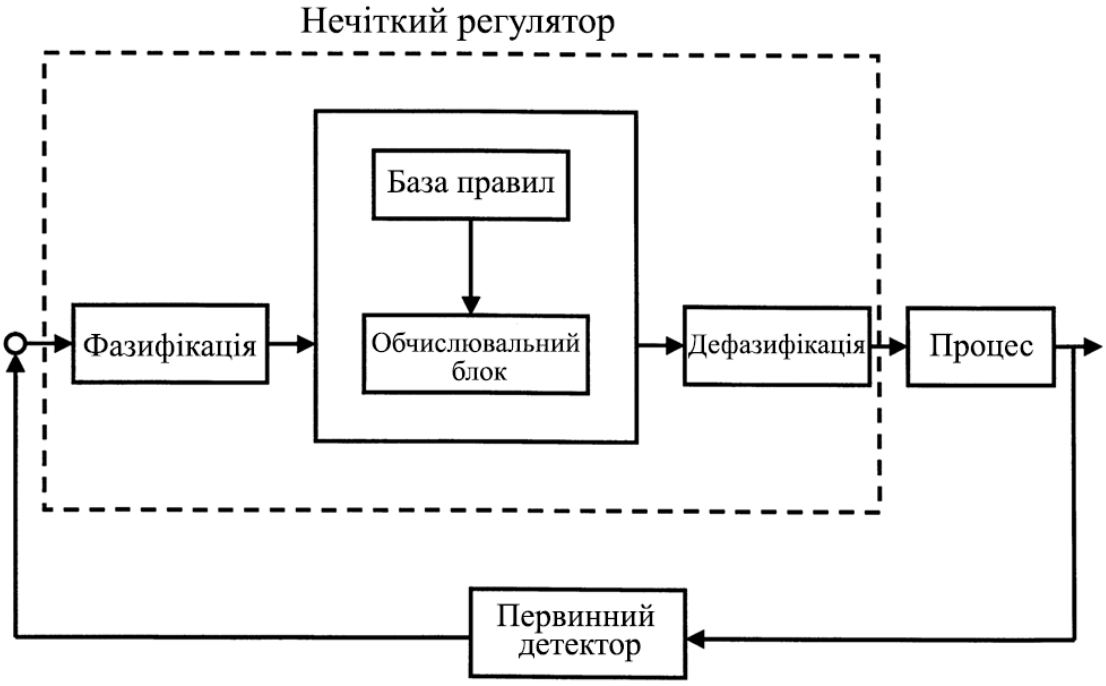


Рис. 7

Розглянемо більш уважно кожний з головних блоків нечіткого регулятора[16][17]:

**Блок фазифікації.** Фазифікація — це процес перетворення чітких вхідних значень у нечіткі значення за допомогою функцій приналежності.

Функції приналежності визначають ступінь належності вхідних даних до певної лінгвістичної змінної, наприклад «низький», «середній» або «високий». Наприклад, якщо вхідними даними є температура, функція приналежності може призначити значення 0,2 нечіткій лінгвістичній змінній «холодно», вказуючи, що температура трохи низька.

**База правил.** Це набір правил «якщо-тоді», які визначають зв’язок між вхідними та вихідними змінними. Кожне правило складається з умови (якщо) і наслідку (тоді). Умовна частина визначає умови, за яких застосовується правило, тоді як наслідкова частина визначає вихідне значення, яке має бути призначене під час запуску правила.

**Обчислювальний блок.** обчислювальний блок оцінює базу правил за допомогою нечіткої логіки для визначення відповідного вихідного значення. Це досягається шляхом агрегування вихідних даних правила за допомогою комбінованого методу, такого як максимінний або середньозважений.

**Дефазифікація.** Це процес перетворення нечітких вихідних значень у чітке вихідне значення. Це робиться шляхом відображення вихідних значень на безперервний діапазон і обчислення центроїда отриманого нечіткого набору.

**Вхідний та вихідний блок.** Вхідні дані для нечіткого регулятора складаються з одного або кількох чітких значень, таких як показання датчиків або дані користувача. Вихідні дані – це єдине чітке значення, яке представляє керуючу дію, яку має виконати система.

В практичній частині було вирішено використати даний апарат для визначення затримки ураховуючи кількість згадувань локацій та міру схожості (впевненості) з відповідними локаціями, які збирається відвідувати комівояжер. Приклад вхідних змінних показано на рис. 8, 9, де було вирішено вибрати трикутні числа з динамічними параметрами (змінюються з урахуванням до вхідних даних з навченої моделі).

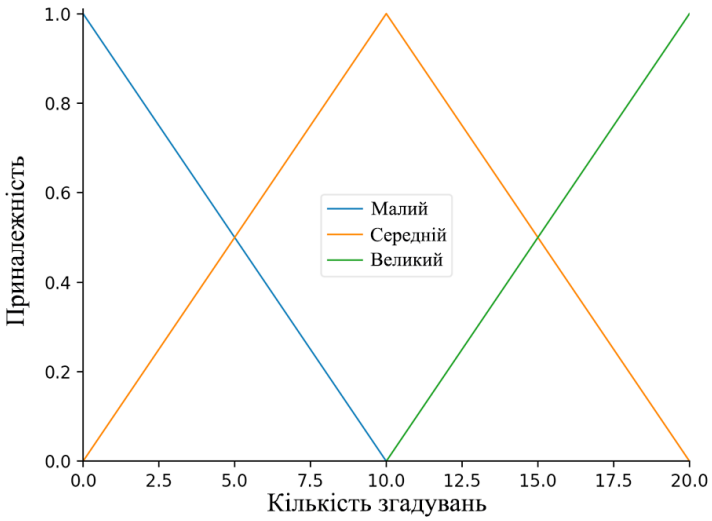


Рис. 8

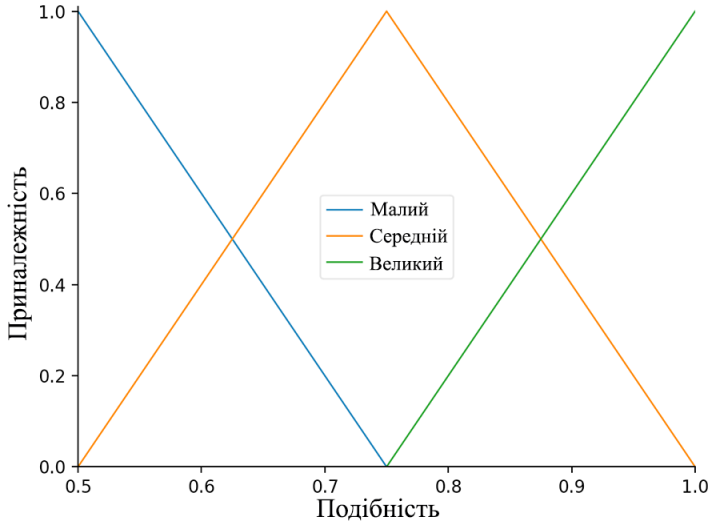


Рис. 9

Та відповідна вихідна змінна показана на Рис. 10, де нечітка змінна визначає кількість хвилин затримки на дорозі треба додати до відповідного шляху.

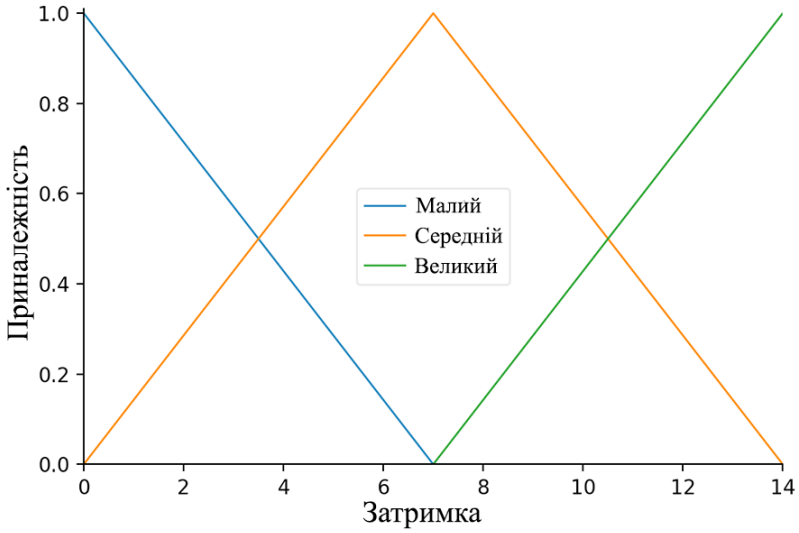


Рис. 10

А база правил складалась з наступних правил:

- Якщо “Кількість згадувань” відповідає оцінці “Малий” та “Подібність” відповідає оцінці “Малий”, тоді “Затримка” відповідає оцінці “Малий”.

- Якщо “Кількість згадувань” відповідає оцінці “Середній” та “Подібність” відповідає оцінці “Малий”, тоді “Затримка” відповідає оцінці “Середній”.
- Якщо “Кількість згадувань” відповідає оцінці “Великий” та “Подібність” відповідає оцінці “Великий”, тоді “Затримка” відповідає оцінці “Великий”.
- Якщо “Кількість згадувань” відповідає оцінці “Малий” та “Подібність” відповідає оцінці “Великий”, тоді “Затримка” відповідає оцінці “Середній”.
- Якщо “Кількість згадувань” відповідає оцінці “Великий” та “Подібність” відповідає оцінці “Середній”, тоді “Затримка” відповідає оцінці “Великий”.
- Якщо “Кількість згадувань” відповідає оцінці “Малий” та “Подібність” відповідає оцінці “Середній”, тоді “Затримка” відповідає оцінці “Середній”.

Цих правил хватило для ефективної роботи нечіткий регулятора. Більш детально можна ознайомитись з цим в практичніій частині данної роботи, ну а код програми знаходиться в додатках.

### 2.3.2. Розв’язання нечіткої задачі комівояжера методом гілок і меж з урахуванням оцінок часу тривалості переміщень

В цілому про задачу комівояжера було розказано в розділі 1.4, але тепер розуміючи краще вхідні дані та умову задачі, можна углибитися в більш математичне пояснення алгоритму розв’язання задачі комівояжера методом, що використовували в практичній задачі.

В цілому алгоритм можна розбити на 6 кроків, а вхідні дані нехай в нас записані у вигляді матриці, як показано на рис. 11. Нехай початкова точка в позиції $(i, j)$, та $c(i, j)$ представляє собою вартість (відстань, час)

проїзду з міста $i$ до міста $j$. Шлях – це такий набір пар міст, наступного вигляду:

$$t = [(1,3)\ (3,2)\ (2,5)\ (5,4)\ (4,1)]$$

який описує подорож комівояжера, де він має потрапити в кожне місто один раз і тільки один раз (в розділі 1.4 описана дана постановка задачі). Нехай z — вартість шляху. З рис. 11 видно, що вищевказаний шлях коштуватиме:

$$z = 52 + 13 + 33 + 10 + 38 = 146$$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | ∞ | 22 | 52 | 38 | 23 |
| 2 | 22 | ∞ | 13 | 36 | 33 |
| 3 | 52 | 13 | ∞ | 24 | 25 |
| 4 | 38 | 36 | 24 | ∞ | 10 |
| 5 | 23 | 33 | 25 | 10 | ∞ |

Рис. 11

Далі відбувається наступний алгоритм[12]:

**Крок 1:** Зменшити рядки та стовпці матриці вартостей на найменше число в рядку чи стовпці. Зберегти суму скорочень, як нижню межу вартості шляху.

**Крок 2:** Нехай дано вузол $X$, від якого слід далі розгалужуватися, і задано матрицю вартостей, пов'язану з $X$, знайти пару міст $(k, l)$, яка максимізує $\theta$, і збільшити дерево від $X$ до вузла $\overline{k, l}$. Додати $\theta(k, l)$ до нижньої межі $X$, щоб встановити нижню межу нового вузла.

**Крок 3:** Якщо кількість пар міст, які входять у шляхи $X$, становить $n - 2$, перейти до **кроку 6**. В іншому випадку продовжувати.

**Крок 4:** Завершити розгалуження на основі $(k, l)$, розширивши дерево від $X$ до вузла $k, l$; $\overline{m, p}$. Де $(m, p)$ пара міст, яка з'єднує кінці найдовшого

шляху, що включає $(k, l)$ у набір прийнятих пар міст нового вузла. Видалити рядок $k$ і стовпець $l$ в матриці вартостей та встановити $c(m, p) = \infty$. Якщо можливо знову скоротити рядки та стовпці, і додати величину скорочення до нижньої межі $X$, щоб встановити нижню межу для нового вузла.

**Крок 5:** Якщо не знайдено жодного вузла, який містить лише один шлях, знайти кінцевий вузол із найменшою нижньою межею та повернутися до **кроку 2** для нового розгалуження. Інакше рухатись далі до **кроку 6**.

**Крок 6:** Якщо вхід до цього кроку здійснюється з **кроку 3**, наступним вузлом є $k, l$; $m, p$, де $(m, p)$ — єдина пара міст, що залишилася після викреслення рядка $k$ і стовпця $l$. Вузол містить один лише шлях. Обчислити його вартість. Тепер, для будь-якого входу до кроку, нехай $t_0$ позначає шлях з найменшою вартістю $z_0$ серед окремих шляхів, знайдених на даний момент. Якщо $z_0$ менше або дорівнює нижній межі кожного кінцевого вузла дерева, $t_0$ є оптимальним. В іншому випадку вибрати кінцевий вузол із кількома шляхами з найменшою нижньою межею та повернутися до **кроку 2** для нового розгалуження.

На цьому закінчується теоретична частина даної роботи, далі в 3 розділі буде вже розглянута сама практична задача у вигляді програмної реалізації.

# Розділ 3. Програмні засоби та технології для вирішення задачі

## 3.1. Архітектура програмної системи

Дана програма є спробою розв’язати популярну задачу комівояжера з урахуванням інформації з новинних джерел, в даному прикладі використовувались дані про Київ, а сам комівояжер вибрав наступні пункти для відвідування: Деміївська площа, Севастопольська площа, Дарницька площа, бульвар Перова та метро Почайна.

Програмна реалізація складається з наступних директорій та файлів:

**"wordrep_for_mitie"** – директорія, що містить дані для mitie функції сховища слів, яка використовується для компіляції "total_word_feature_extractor.dat", що, в свою чергу, використовується для навчання моделі NER.

**"train_data"**— директорія, що містить дані для навчання моделі NER, дані, отримані з веб-сайту ner-uk, складаються з анотованих даних із різних новинних джерел у розмітці ner-uk.

**"workspace"** – директорія, де зберігається навчена модель NER ("mitie_ner_model_ver1.dat")

**"data"** – директорія, де знаходяться всі вхідні дані від комівояжера ("articles.xlxs" і "news_data.txt" будуть скомпільовані основною програмою. Застарілі приклади цих файлів вже є в дерикторії, щоб швидко скомпілювати основну програму, але ви можете видалити їх і створити нові основною програмою "main.py")

**"total_word_feature_extractor.dat"** – файл загального сховища слів для бібліотеки mitie, використовується для компіляції моделі NER.

**"ner_training.py"** – програма для навчання моделі NER. Використовує "ner_utils_func.py" для деяких функцій перетворення анотованих даних з розмітки ner-uk у токенізовану розмітку mitie.

**"ner_utils_func.py"** - допомагає конвертувати дані з різних розміток. (взято з ner-uk github)

**"dev-test-split.txt"** – текстовий файл із іменами файлів розділених на дані для тренування та дані для тестування з директорії "train_data"

**"workplace/mitie/ mitie_ner_model_ver1.dat"** - NER модель.

**"main.py"** - основний програмний код, працюватиме лише за наявності "workspace/mitie/mitie_ner_model_ver1.dat" і "data/routes.txt"

**"data/routes.txt"** - основний вхідний файл від комівояжера, містить маршрути (з назвами локацій), які продавець використовуватиме для відвідування всіх пунктів призначення, також містить початковий час подорожі для кожного маршруту (саме ці значення ми потім оновлюємо, використовуючи дані з джерел новин)

## 3.2. Опис засобів технологічного стеку

Основна програма складається з наступних функцій (main.py):

**extracting_urls() -** вилучення усіх посилань на джерела новин за запитом "Ремонт доріг Київ" (період 1 місяць).

**news_scraper()** - зчитування текстів з новинних джерел за допомогою URL-адрес із extracting_urls()

**ner_locs()** - використовуючи попередньо навчену NER модель, знаходимо всі назви локацій з текстового файлу новин скомпільованого в news_scraper()

**reading_input_data()** - читання вхідного файлу комівояжера, що містить усі доступні маршрути та початковий час подорожі на них.

**road_repair_status_check()** - перевірка того, чи місця з ремонтними роботами, знайдені в новинних джерелах, є на маршрутах між пунктами призначення комівояжера. Також саме тут імплементований нечіткий регулятор за допомогою якого визначається час затримки на тому чи іншому шляху.

**updating_time_matrix()** - створення та оновлення матриці з вагами (час у дорозі) для кожного маршруту (використовується для задачі комівояжера).

**travellingSalesmanProblem()** - реалізація задачі комівояжера наівним шляхом, де просто перебираються всі можливі перестановки. Ця функція в головній програмі не використовується, вона створена тільки для порівняння часової складності з іншими більш ефективними методами, а сама задача комівояжера розв’язується в branch_and_bound_TSP().

**branch_and_bound_TSP()** – функція, що відповідає за розв’язання задачі комівояжера методом гілок та меж.

**visualize_graph()** – візуалізація графа оптимального шляху комівояжера на карті Києва.

**main()** – керуючий код програми.

## 3.3. Приклади практичного використання програми

Для початку розглянемо вхідний файл від комівояжера, що зображений на рис. 12. Він містить шляхи якими комівояжер збирається відвідувати наступні пункти: Деміївська площа, Севастопольська площа, Дарницька площа, бульвар Перова та метро Почайна. Як видно з рис. 12, кожний шлях складається з переліку локацій, де перша локація початок шляху, остання кінець. Вибрані шляхи між кожними двома пунктами призначення за найменшою відстанню між ними. Роздільником локацій є кома та кожний наступний шлях починається з нового рядка текстового документу. Також після переліку всіх локацій в кінці кожного рядку є орієнтовний час за який можна подолати цей шлях без будь яких затримок ураховуючи всі правила дорожнього руху. Саме цей час далі в програмі буде модифіковано за рахунок даних з новинних ресурсів та нечіткого регулятора. Роздільником орієнтовного часу є “,;”(кома разом з крапка з комою).

```
метро Почайна,Оболонський проспект,проспект Степана Бандери,Північний міст,проспект Романа
Шухевича,Керченська площа,бульвар Перова,;9
метро Почайна,Оболонський проспект,проспект Степана Бандери,Північний міст,проспект Романа
Шухевича,Керченська площа,вулииця Бротиславська,Чернігівський міст,метро Чернігівська,проспект Юрія
Гагаріна,Дарницька площа,;16
метро Почайна,Оболонський проспект,проспект Степана Бандери,вулиця Набережно-Рибальська,Гаванський
міст,вулиця Набережно-Хрещатицька,Набережне шосе,бульвар Дружби Народів,Печерський
міст,Голосіївський проспект,Деміївська площа,;16
метро Почайна,Оболонський проспект,роспект Степана Бандери,вулиця Олени Теліги,метро
Дорогожичі,вулиця Олександра Довженка,Шулявський міст,вулиця Вадима Гетьмана,Чоколівський
бульвар,Севастопольська площа,;11
бульвар Перова,проспект Визволителів,вулиця Будівельників,проспект Юрія Гагаріна,Дарницька площа,;7
бульвар Перова,проспект Визволителів,Броварський проспект,Русанівський міст,міст Метро,Набережне
шосе,бульвар Дружби Народів,Печерський міст,Голосіївський проспект,Деміївська площа,;14
бульвар Перова,Керченська площа,проспект Романа Шухевича,Північний міст,проспект Степана
Бандери,вулиця Набережно-Рибальська,Гаванський міст,вулиця Набережно-Хрещатицька,вулиця Нижній
Вал,вулиця Глибочицька,Глибочицький проїзд,вулиця В'ячеслава Чорновола,Повітрофлотський
проспект,Севастопольська площа,;17
Дарницька площа,проспект Соборності,міст Патона,бульвар Дружби Народів,Печерський міст,Голосіївський
проспект,Деміївська площа,;8
Дарницька площа,проспект Соборності,міст Патона,бульвар Дружби Народів,Печерський міст,Деміївська
площа,проспект Валерія Лобановського,Севастопольська площа,;14
Деміївська площа,проспект Валерія Лобановського,Севастопольська площа,;5
```

Рис. 12. Приклад вхідного файлу від комівояжера

На рис. 13 можна побачити приклад виводу в консоль роботи даної програми. Спочатку програма зчитує вхідний файл від комівояжера (routes.txt) рис. 12 та виводяться усі пункти відвідування комівояжера в

консоль. Далі виводиться зчитаний вектор з початкового часу для кожного маршруту та відповідна симетрична матриця для задачі комівояжера.

```
Destinations salesman want to visit:
['метро Почайна', 'бульвар Перова', 'Дарницька площа', 'Деміївська площа', 'Севастопольська площа']

--------------------------------------------------------------------------------


Initial travel time for every route (from input file):
[9, 16, 16, 11, 7, 14, 17, 8, 14, 5]

Initial travel time matrix (from input file):
0       9       16      16      11
9       0       7       14      17
16      7       0       8       14
16      14      8       0       5
11      17      14      5       0

--------------------------------------------------------------------------------
loading NER model...

Locations where repair works are in progress:
['Чернігівський міст', 'метро Чернігівська', 'Голосіївський проспект', 'Броварський проспект', 'Північний міст', 'Дар
ницька площа', 'Русанівський міст', 'Гаванський міст', 'Оболонський проспект', 'Севастопольська площа']

Updated travel time for every route:
[22, 52, 38, 23, 13, 36, 33, 24, 25, 10]

Updated travel time matrix:
0       22      52      38      23
22      0       13      36      33
52      13      0       24      25
38      36      24      0       10
23      33      25      10      0

--------------------------------------------------------------------------------


Path of the most efficient tour:
метро Почайна -> бульвар Перова -> Дарницька площа -> Деміївська площа -> Севастопольська площа -> метро Почайна
Estimated cost of the most efficient tour: 92 min
```

Рис. 13. Приклад виводу в консоль програми

Наступним кроком є підготовка даних зі інтернет ресурсів. Спочатку програма знаходить усі URL-посилання за останній місяць за запитом "Ремонт доріг Київ" та зчитує текст з цих посилань в окремий файл.

Далі завантажується завчасно натренована NER модель та здійснюється пошук локацій зі зчитаного тексту новинних джерел.

Потім програма перевіряє чи існують знайдені локації в маршрутах комівояжера, якщо так, то зберігає їх окремо разом з кількістю згадувань даної локації в текстах та мірою подібності. Далі за допомогою нечіткого регулятора програма визначає кількість хвилин що треба додати до відповідного шляху, звісно при умові якщо там є ремонт доріг. Далі

програма виводить локації з маршруту комівояжера де можливо проходять ремонтні дороги та виводить оновлений вектор часу з оновленою матрицею часу у дорозі.

Далі з урахуванням усіх вхідних даних розв’язується задача комівояжера методом гілок та меж. Виводиться найоптимальніший маршрут та приблизний час потрібний для подолання цього маршруту. Ну і в кінці програма візуалізує оптимальний маршрут на карті Києва, та виводить час потрібний для його подолання (рис. 14).

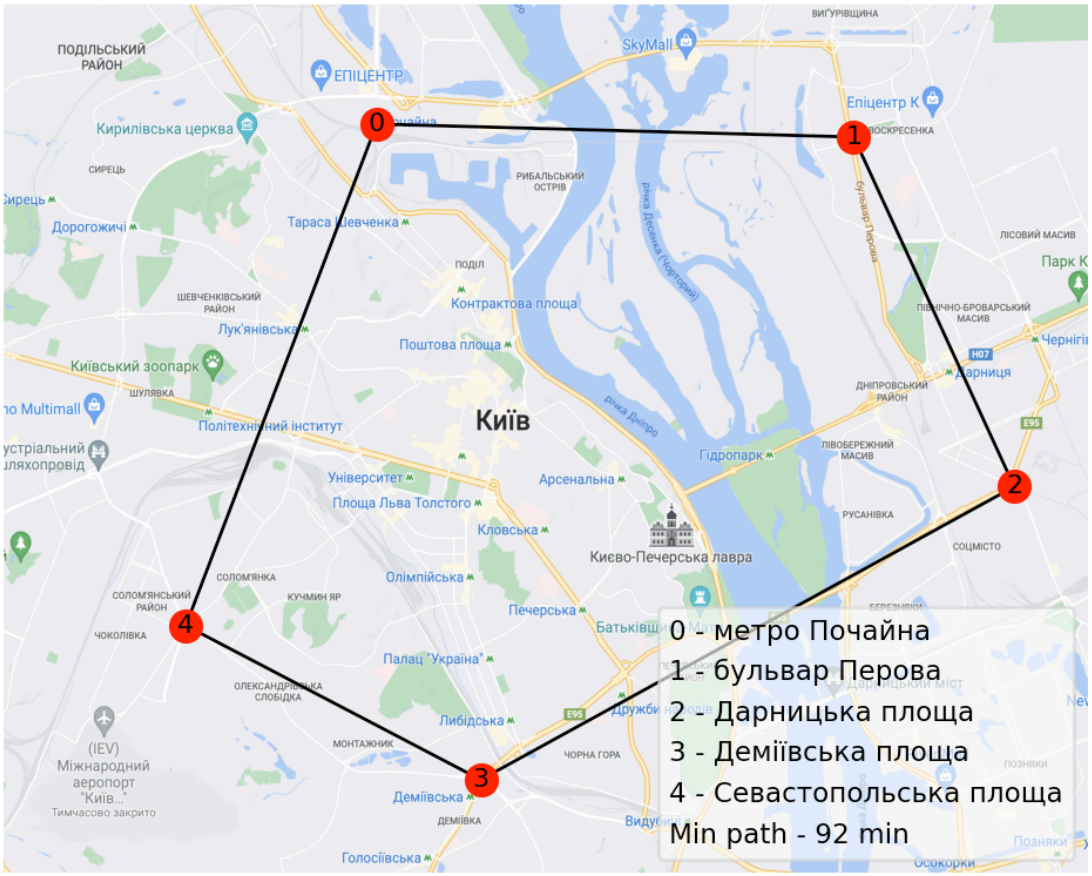


Рис. 14

# Висновки

В магістерській роботі запропоновано підхід до розв’язання нечіткої задачі комівояжера як однієї з сучасних проблем логістики, реалізовано спосіб накопичення та розподілу потоків даних. Досліджено методи розв’язання задачі комівояжера, зосереджену увагу на використанні методу гілок та меж, здійснено навчання лінгвістичної моделі розпізнавання сутностей. На основі підходу до формалізації суб’єктивних оцінок часових інтервалів розв’язано нечітку задачу комівояжера та розроблено програмну реалізацію запропонованого підходу.

В теоретичній частині магістерської роботи був представлений аналіз сучасних проблем логістики, основними з яких було виділено:

1. Збільшення транспортних витрат
2. Невідповідність трекінгу
3. Обмежена видимість відправлень
4. Фрагментована комунікація
5. Порожні милі
6. Затримки доставки
7. Форс-мажорні обставини

Визначено перелік аналітичних методів для розв’язання даних проблем: лінійне програмування, оптимізація мережі, теорія масового обслуговування, імітаційне моделювання, евристика та метаевристика, генетичні алгоритми, аналіз рішень. Основну увагу приділено методам лінійного програмування.

Проведено аналіз сучасних проблем здобуття, накопичення та розподілу потоків даних, основними з яких було виділено: нерівномірний розподіл даних, асиметрія даних, розміщення даних, реплікація даних. Розглянуто способи вирішення, що знайшли відбиття у вирішенні задачі накопичення даних та у створенні програмної реалізації.

Розглянуто нечіткі задачі оптимізації, теорія нечітких множин та їх застосування, розглянуто методику нечіткого лінійного програмування та її використання при розв’язанні задач з нечіткою інформацією, наприклад, поданою у вигляді текстів. Розглянуто постановку задачі комівояжера, її математичне формулювання та різні підходи до розв’язання даної задачі, з їх перевагами та недоліками.

Практичний аспект роботи спрямовано на реалізацію методики розв’язання нечіткої задачі комівояжера з урахуванням суб’єктивних часових оцінок тривалості переміщень, базуючись на використанні елементів машинного навчання. Розглянуто обробка природньої мови та розпізнавання іменованих сутностей. Як один з найкращих методів навчання моделей розпізнавання іменованих сутностей обрано метод опорних векторів, наведено його сутність та характеристика. Описано програмну реалізацію методу меж та гілок для розв’язання нечіткої задачі комівояжера, розглянуто спосіб формування нечітких суб’єктивних оцінок тривалості переміщень за ділянками маршруту.

# Список використаних джерел


1. **Martin Christopher** (February 26, 2016). “Logistics and Supply Chain Management”, *FT Publishing International*; 5th edition
2. **Alan Harrison, Remko van Hoek** (February 28, 2005). “Logistics Management and Strategy”, *Financal Times Management*; 2nd edition
3. **Gianpaolo Ghiani, Gilbert Laporte, Roberto Musmanno** (January 9, 2004). “Introduction to Logistics Systems Planning and Control”, *John Wiley & Sons, Ltd*
4. **Ajay D. Kshemkalyani, Mukesh Singhal** (2008). "Distributed Computing: Principles, Algorithms, and Systems", *Cambridge University Press*
5. **Larissa T. Moss, Shaku Atre** (March 7, 2003). “Business Intelligence Roadmap: The Complete Project Lifecycle for Decision-Support Applications”, *Addison-Wesley Professional*; 1st edition
6. **Zimmermann, H.-J.** (2001). “Fuzzy Set Theory – and Its Applications”, *Kluwer Academic Publishers*, fourth edition
7. **Зайченко Ю.П** (2008). “Нечеткие модели и методы в интеллектуальных системах”, *Киев, Слово*, с. 222-262, 298-332
8. **Zimmermann, H. J.** (1991). “Fuzzy mathematical programming”, *Computers & operations research*, 18(4), 343-359.
9. **Vanderbei R. J.** (2014). “Linear programming: Foundations and extensions”, *Springer*.
10. **Dantzig G., Fulkerson R., Johnson S.** (November 1954). “Solution of a Large-Scale Traveling-Salesman Problem”, *Journal of the Operations Research Society of America*. **2** (4): 393–410.
11. **Richard Bellman** (1962). “Dynamic programming treatment of the travelling salesman problem”, *Journal of Assoc. Computing Mach.* 9.

12. **John D. C. Little, Katta G. Murty, Dura W. Sweeney, Caroline Karel** (1963). “An algorithm for the traveling salesman problem”, *M.I.T.*
13. **Steven Bird, Ewan Klein, Edward Loper** (2009). “Natural Language Processing with Python”, *O'Reilly Media, Inc.*
14. **Cortes C., Vapnik V.** (1995). “Support-vector networks”, 273-297.
15. **Schölkopf B., Smola A.** (2002). “Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond”, *MIT Press.*
16. **L. X. Wang** (1997). “A course in fuzzy systems and control”, *Prentice Hall, Upper Saddle River, NJ, 1997.*
17. **W.J.M. Kickert, E.H. Mamdani** (1978). “Analysis of a fuzzy logic controller”, *Fuzzy Sets and Systems*

Електронні ресурси:

1. Анотовані данні для тренування: https://www.lang.org.ua/en/about/
2. Документація бібліотеки MITIE: https://github.com/mit-nlp/MITIE
3. Нечіткий регулятор: https://pythonhosted.org/scikit-fuzzy/auto_examples/plot_tipping_problem_newapi.html
4. https://www.geeksforgeeks.org/
5. https://uk.wikipedia.org/
6. https://github.com
7. https://www.researchgate.net

# Додаток А

Код програми тренування NER моделі

```
import time
from mitie import *
from ner_utils_func import BsfInfo, parse_bsf, read_train_test_split
import os
from tqdm import tqdm



def prepare_mitie_training_data(dev_files):
    # convert char offset in ner-uk markup to token based MITIE markup
    base_path= "/Applications/CS_diploma_code/train_data/" #folder with all
training data
    samples = []
    for f_name in dev_files:
        # read ann
        with open(base_path + f_name + '.ann', 'r') as f:
            annotations = parse_bsf(f.read())
        # read tokens
        with open(base_path + f_name + '.txt', 'r') as f:
            tok_txt = f.read()

        tokens = tok_txt.split()

        # convert char offset to token offset
        tok_ann = []
        tok_idx = 0

        ann: BsfInfo
        for ann in annotations:
            tok_start = 0
            in_token = False
            tok_end = 0
            for i in range(tok_idx, len(tokens)):
                tok_idx = i + 1
                if not in_token and ann.token.startswith(tokens[i]):
                    tok_start = i
                    tok_end = i + 1
                    in_token = (len(ann.token) != len(tokens[i]))
                    if len(ann.token) == len(tokens[i]):
                        break
                elif in_token and ann.token.endswith(tokens[i]):
                    tok_end = i + 1
                    in_token = False
                    break
            tok_ann.append(BsfInfo(ann.id, ann.tag, tok_start, tok_end, ann.token))

        # Create MITIE sample
```

```
            sample = ner_training_instance(tokens)
            for t_ann in tok_ann:
                sample.add_entity(xrange(t_ann.start_idx, t_ann.end_idx), t_ann.tag)
            samples.append(sample)

        print(f'Converted to MITIE format. Sample documents {len(samples)}')
        return samples


def run_training(cpu_threads, config_path, feature_extractor_path):
    dev_files, test_files = read_train_test_split(config_path)
    print(f'Loaded corpus file split configuration (documents):
DEV={len(dev_files)}, TEST={len(test_files)}')

    samples = prepare_mitie_training_data(dev_files)

    # check for workspace folder existence
    workspace_folder = os.path.join('workspace', 'mitie')

    """
    if not os.path.exists(workspace_folder):
        os.makedirs(workspace_folder)
    """

    # Training
    trainer = ner_trainer(feature_extractor_path)

    for s in samples:
        trainer.add(s)

    trainer.num_threads = cpu_threads

    print("Launching training process...")
    # takes long here
    ner = trainer.train()

    model_path = os.path.join(workspace_folder, "mitie_ner_model_ver1.dat")
    ner.save_to_disk(model_path)
    print(f'Training finished. Model saved to "{model_path}"')


if __name__ == '__main__':
    t1=time.perf_counter()
    if not os.path.exists('/Applications/CS_diploma_code/train_data'):
        print("Error: data folder not found")
    else:
        run_training(8, 'dev-test-split.txt', 'total_word_feature_extractor.dat')
    t2=time.perf_counter()
    print(f'Finished in {t2-t1} seconds')
```

# Додаток Б

## Головний код програмної реалізації даної роботи

```
#Made by Kiptyk Kirill

import os
import time
import difflib
import math
import numpy as np
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib.patches as mpl_patches
import skfuzzy as fuzz
from skfuzzy import control as ctrl
from mitie import *
from sys import maxsize
from urllib.request import urlopen, Request
from urllib.error import HTTPError, URLError
from bs4 import BeautifulSoup
from GoogleNews import GoogleNews
from itertools import permutations
from difflib import SequenceMatcher


# extracting all urls of news sources by the "Ремонт доріг Київ" request (1 month
period)
def extracting_urls(request):
    googlenews = GoogleNews(period='1m')
    googlenews.set_lang('ua')
    googlenews.search(request)
    result=googlenews.result()

    pd.set_option('display.max_colwidth', None)
    df=pd.DataFrame(result)

    for i in range(2,3):
        googlenews.getpage(i)
        result=googlenews.result()
        df=pd.DataFrame(result)

    links=df['link']

    with open('urls.txt', 'a') as f:
        links = links.to_string(header=False, index=False)
        f.write(links)

    with open('urls.txt', 'r') as f:
        urls=f.readlines()
```

```
        urls = [line.replace(' ', '') for line in urls]

    with open('urls.txt', 'w') as f:
        f.writelines(urls)

    df.to_excel("data/articles.xlsx")

    return(urls)

# Scraping news sources with urls from extracting_urls()
def news_scraper(urls):
    for i, url in enumerate(urls):
        print(urls[i])
        response = Request(urls[i])
        response.add_header('user-agent', 'Mozilla/5.0 (Macintosh; Intel Mac OS X
10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.6.1 Safari/605.1.15')
        try:
            client = urlopen(response)
            page = client.read()
            client.close()

            soup = BeautifulSoup(page, 'html.parser')
            text=soup.get_text()
            text=text.split()
            text=' '.join(str(e) for e in text)

            with open(f'data/news_data.txt','a') as f:
                f.write(text+'\n')

        except HTTPError:
            print('HTTP error')
        except URLError:
            print('URL error')
        else:
            print ('ok')

# using pretrained model to find all locations from news text file compiled in
news_scraper()
def ner_locs(dat_file_loc,text_file_loc):
    print ("loading NER model...")
    ner = named_entity_extractor(dat_file_loc)

    #print ("\nTags output by this NER model:", ner.get_possible_ner_tags())

    # Load a text file and convert it into a list of words.
    file=load_entire_file(text_file_loc).decode('utf8')

    tokens=file.split()

    #print ("Tokenized input:", tokens)
    entities = ner.extract_entities(tokens)
```

```
    #print ("\nEntities found:", entities)
    #print ("\nNumber of entities detected:", len(entities))

    # entities is a list of tuples, each containing the entity tag and a xrange
that indicates which tokens are part of the entity.
    locations=[]

    for i,e in enumerate(entities):
        range = e[0]
        tag = e[1]
        accuracy = e[2]
        #print(i)

        if tag=='LOC':
            entity_text = " ".join(tokens[i] for i in range)
            if accuracy >= 0:
                locations.append(entity_text)

    return(locations)

# calculating time for every path using fuzzy control
def road_repair_status_check(loc_entities,routes,initial_time):
    # saving all locations that visits salesman into one list
    valuable_locs=[]
    for route in routes:
        for loc in route:
            if loc not in valuable_locs:
                valuable_locs.append(loc)

    #
    loc_rat=[]
    for loc_ent in loc_entities:
        l={}
        for loc in valuable_locs:
            ratio=difflib.SequenceMatcher(None,loc_ent,loc).ratio()
            l[loc]=ratio

        max_r=max(l,key=l.get)
        if l[max_r]>0.5:
            loc_rat.append([max_r,l[max_r]])
        l.clear()

    #
    unique_locs=[]
    loc_number_ratio=[]
    for loc in loc_rat:
        if loc[0] not in unique_locs:
            unique_locs.append(loc[0])
            loc_number_ratio.append([loc[0]])

    #
```

```
n_mentions=[]
for i,loc in enumerate(unique_locs):
    s=sum(x.count(loc) for x in loc_rat)

    r=[]
    for loc_r in loc_rat:
        if loc_r[0]==loc:
            r.append(loc_r[1])

    loc_number_ratio[i].append(s)
    n_mentions.append(s)
    loc_number_ratio[i].append(sum(r)/s)

#
def detect_outlier(data):
    # find q1 and q3 values
    q1, q3 = np.percentile(sorted(data), [25, 75])
    # compute IRQ
    iqr = q3 - q1

    upper_bound = q3 + (1.5 * iqr)
    #lower_bound = q1 - (1.5 * iqr)

    outliers = [x for x in data if x >= upper_bound]
    return outliers

outliers=detect_outlier(n_mentions)

for out in outliers:
    if out in n_mentions:
        n_mentions.remove(out)

for outlier in outliers:
    for ent in loc_number_ratio:
        if ent[1]==outlier:
            ent[1]=max(n_mentions)

#print(loc_number_ratio)

# New Antecedent/Consequent objects hold variables and membership functions
mentions= ctrl.Antecedent(np.arange(0, max(n_mentions)+1, 1), 'mentions')
similarity = ctrl.Antecedent(np.arange(0.5, 1.01, 0.01), 'similarity')
delay = ctrl.Consequent(np.arange(0, 15, 1), 'delay')

mentions.automf(3)
similarity.automf(3)
delay.automf(3)

#mentions.view()
#similarity.view()
#delay.view()
```

```
    #plt.show()

    rule1 = ctrl.Rule(mentions['poor'] & similarity['poor'] , delay['poor'])
    rule2 = ctrl.Rule(mentions['average'] & similarity['poor'], delay['average'])
    rule3 = ctrl.Rule(mentions['good'] & similarity['good'], delay['good'])
    rule4 = ctrl.Rule(mentions['poor'] & similarity['good'], delay['average'])
    rule5 = ctrl.Rule(mentions['good'] & similarity['average'], delay['good'])
    rule6 = ctrl.Rule(mentions['poor'] & similarity['average'], delay['average'])

    time_delay_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5,
rule6])

    time_delay_sim = ctrl.ControlSystemSimulation(time_delay_ctrl)

    # Pass inputs to the ControlSystem using Antecedent labels with Pythonic API
    locs_time_delays={}

    for ent in loc_number_ratio:
        time_delay_sim.input['mentions']=ent[1]
        time_delay_sim.input['similarity']=ent[2]
        time_delay_sim.compute()
        locs_time_delays[ent[0]]=int(time_delay_sim.output['delay'])

    #print(locs_time_delays)


    for i,route in enumerate(routes):
        for loc in route:
            if loc in locs_time_delays.keys():
                initial_time[i]+=locs_time_delays[loc]


    print('\nLocations where repair works are in progress:')
    print(list(locs_time_delays.keys()))
    print('\nUpdated travel time for every route:')
    print(initial_time)

    return(initial_time)

# reading salesmans input file that consist of all available routes and initial
travel time for them
def reading_input_data(salesman_routes_file):

    with open(salesman_routes_file, 'r') as f:
        text=f.read().splitlines()

    locations=[]
    estimate_time=[]
    for route in text:

        if route=='' or route==' ':
```

```
            continue

        route1=route.split(',')
        locations.append(route1[:-1])
        estimate_time.append(route.split(';')[1])
    #print(locations)
    n_dest=np.roots([1/2,-1/2,-len(estimate_time)])[0]

    if n_dest.is_integer()==False:
        print('Wrong input data')
        return 0


    time_matrix=[[0 for i in range(int(n_dest))] for i in range(int(n_dest))]

    for i in range(len(estimate_time)):
        estimate_time[i]=int(estimate_time[i])

    n_dest=int(n_dest)

    time_matrix=updating_time_matrix(time_matrix,n_dest,estimate_time)

    destinations=[]
    i=n_dest-2
    n=0
    destinations.append(locations[i][0])

    while i!=len(locations)-1:
        destinations.append(locations[i+n_dest-2-n][0])
        i+=n_dest-2-n
        n+=1

    destinations.append(locations[-1][-1])

    return(time_matrix,locations,estimate_time,n_dest,destinations)

# creating and updating matrix with weights (travel time) for every route, it is
used for travelling salesman problem
def updating_time_matrix(time_matrix,n_dest,estimate_time):
    i=0
    for x in range(n_dest):
        for y in range(n_dest):
            if x>=y:
                continue
            time_matrix[x][y]=time_matrix[y][x]=estimate_time[i]
            i+=1
    return time_matrix

# implementation of Travelling Salesman Problem, naive approach
def travellingSalesmanProblem(graph,  V):
    # store all vertex apart from source vertex
```

```
    vertex = []
    #starting point
    s=0

    for i in range(V):
        if i != s:
            vertex.append(i)

    # store minimum weight Hamiltonian Cycle
    min_pathweight = maxsize
    res_path=()
    all_permutations=permutations(vertex)

    for next_permutation in all_permutations:
        #print(i)
        # store current Path weight(cost)
        current_pathweight = 0

        # compute current path weight
        k = s
        for j in next_permutation:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]

        # update minimum
        min_pathweight = min(min_pathweight, current_pathweight)
        if min_pathweight==current_pathweight:
            res_path=next_permutation

    return(res_path,min_pathweight)

# implementation of Travelling Salesman Problem, branch and bound method
def branch_and_bound_TSP(adj,N,final_path):
    maxsize = float('inf')
    def copyToFinal(curr_path):
        final_path[:N + 1] = curr_path[:]
        final_path[N] = curr_path[0]


    # Function to find the minimum edge cost
    # having an end at the vertex i
    def firstMin(adj, i):
        min = maxsize
        for k in range(N):
            if adj[i][k] < min and i != k:
                min = adj[i][k]
        return min

    # function to find the second minimum edge
    # cost having an end at the vertex i
```

```
def secondMin(adj, i):
    first, second = maxsize, maxsize
    for j in range(N):
        if i == j:
            continue
        if adj[i][j] <= first:
            second = first
            first = adj[i][j]

        elif(adj[i][j] <= second and
            adj[i][j] != first):
            second = adj[i][j]

    return second

# function that takes as arguments:
# curr_bound -> lower bound of the root node
# curr_weight-> stores the weight of the path so far
# level-> current level while moving
# in the search space tree
# curr_path[] -> where the solution is being stored
# which would later be copied to final_path[]
def TSPRec(adj, curr_bound, curr_weight,
           level, curr_path, visited):

    global final_res
    # base case is when we have reached level N
    # which means we have covered all the nodes once
    if level == N:

        # check if there is an edge from
        # last vertex in path back to the first vertex
        if adj[curr_path[level - 1]][curr_path[0]] != 0:

            # curr_res has the total weight
            # of the solution we got
            curr_res = curr_weight + adj[curr_path[level - 1]]\
                                        [curr_path[0]]
            if curr_res < final_res:
                copyToFinal(curr_path)
                final_res = curr_res
        return

    # for any other level iterate for all vertices
    # to build the search space tree recursively
    for i in range(N):

        # Consider next vertex if it is not same
        # (diagonal entry in adjacency matrix and
        #  not visited already)
        if (adj[curr_path[level-1]][i] != 0 and
```

```
                        visited[i] == False):
            temp = curr_bound
            curr_weight += adj[curr_path[level - 1]][i]

            # different computation of curr_bound
            # for level 2 from the other levels
            if level == 1:
                curr_bound -= ((firstMin(adj, curr_path[level - 1]) +
                                firstMin(adj, i)) / 2)
            else:
                curr_bound -= ((secondMin(adj, curr_path[level - 1]) +
                                firstMin(adj, i)) / 2)

            # curr_bound + curr_weight is the actual lower bound
            # for the node that we have arrived on.
            # If current lower bound < final_res,
            # we need to explore the node further
            if curr_bound + curr_weight < final_res:
                curr_path[level] = i
                visited[i] = True

                # call TSPRec for the next level
                TSPRec(adj, curr_bound, curr_weight,
                    level + 1, curr_path, visited)

            # Else we have to prune the node by resetting
            # all changes to curr_weight and curr_bound
            curr_weight -= adj[curr_path[level - 1]][i]
            curr_bound = temp

            # Also reset the visited array
            visited = [False] * len(visited)
            for j in range(level):
                if curr_path[j] != -1:
                    visited[curr_path[j]] = True

# This function sets up final_path
def TSP(adj):

    # Calculate initial lower bound for the root node
    # using the formula 1/2 * (sum of first min +
    # second min) for all edges. Also initialize the
    # curr_path and visited array
    curr_bound = 0
    curr_path = [-1] * (N + 1)
    visited = [False] * N

    # Compute initial bound
    for i in range(N):
        curr_bound += (firstMin(adj, i) +
                    secondMin(adj, i))
```

```
        # Rounding off the lower bound to an integer
        curr_bound = math.ceil(curr_bound / 2)

        # We start at vertex 1 so the first vertex
        # in curr_path[] is 0
        visited[0] = True
        curr_path[0] = 0

        # Call to TSPRec for curr_weight
        # equal to 0 and level 1
        TSPRec(adj, curr_bound, 0, 1, curr_path, visited)

    TSP(adj)

    return final_path,final_res

# visualizing output graph
def visualize_graph(path,destinations,ans):

    pos=[(172,345),(392,339),(466,178),(220,43),(84,114)]

    img = plt.imread("background.png")
    fig,ax = plt.subplots()
    ax.imshow(img,extent=[0,500,0,400])

    G = nx.Graph()
    labels=[]
    handles = [mpl_patches.Rectangle((0, 0), 1, 1, fc="white", ec="white", lw=0,
alpha=0)] * (len(destinations)+1)

    for i,s in enumerate(destinations):
        G.add_node(i,pos=pos[i])

        if i<len(path)-1:
            G.add_edge(path[i],path[i+1])

        labels.append(f'{i} - {destinations[i]}')

    labels.append(f'Min path - {ans} min')

    plt.legend(handles, labels, loc='best', fontsize='small', fancybox=True,
framealpha=0.6, handlelength=0, handletextpad=0)

    nx.draw(G,pos,node_size=90,node_color='red',with_labels=True,font_size=8)

    plt.show()


# Drive Code
def main():
```

```
    # matrix for testing TSP
    #dist = [[0, 10, 15, 20],[10, 0, 35, 25],[15, 35, 0, 30],[20, 25, 30, 0]]

    input_file='data/routes.txt' #'data/data_test.txt'
    try:

estimate_time_matrix,routes,initial_time,number_of_destinations,destinations=readin
g_input_data(input_file)
    except FileNotFoundError:
        print('Wrong file or file path')
        return 0

    print(f'\nDestinations salesman want to visit:\n{destinations}')

    # final_path[] stores the final solution
    # i.e. the // path of the salesman.
    final_path = [None] * (number_of_destinations + 1)

    # visited[] keeps track of the already
    # visited nodes in a particular path
    visited = [False] * number_of_destinations

    # Stores the final minimum weight
    # of shortest tour.
    global final_res
    final_res = maxsize

    path,ans=branch_and_bound_TSP(estimate_time_matrix,
number_of_destinations,final_path)

    #print(f"\nPath of the most efficient tour:\n{' -> '.join([destinations[i] for
i in path])}")
    #print(f"Estimated cost of the most efficient tour: {ans} min\n")

    final_path = [None] * (number_of_destinations + 1)
    final_res = maxsize


    print('\n-----------------------------------------------------------------------
-----------')

    print('\nInitial travel time for every route (from input file):')
    print(initial_time)

    print('\nInitial travel time matrix (from input file):')
    for i in estimate_time_matrix:
        print('\t'.join(map(str,i)))

    print('\n-----------------------------------------------------------------------
-----------')
```

```
    # next two functions are time consuming, so be patient or just skip them this
time.
    # news_data.txt file created by this two functions is already in 'data'
directory

    '''
    #urls=extracting_urls('Ремонт доріг київ') #'Ремонт доріг київ'
    #news_scraper(urls)
    '''

    # uk_model.dat - NER model from lang-uk github with their training set
    # workspace/mitie/mitie_ner_model_ver1.dat - NER model trained by the author of
this program with training data set taken from lang-uk repository


loc_entities=ner_locs('workspace/mitie/mitie_ner_model_ver1.dat','data/news_data.tx
t') #uk_model.dat #workspace/mitie/mitie_ner_model_ver1.dat

    new_time=road_repair_status_check(loc_entities,routes,initial_time)

new_time_matrix=updating_time_matrix(estimate_time_matrix,number_of_destinations,ne
w_time)

    path,ans=branch_and_bound_TSP(new_time_matrix,
number_of_destinations,final_path)

    print('\nUpdated travel time matrix:')
    for i in new_time_matrix:
        print('\t'.join(map(str,i)))

    print('\n----------------------------------------------------------------------
-----------')

    print(f"\nPath of the most efficient tour:\n{' -> '.join([destinations[i] for i
in path])}")
    print(f"Estimated cost of the most efficient tour: {ans} min\n")
    visualize_graph(path,destinations,ans)


if __name__ == "__main__":
    t1=time.perf_counter()
    main()
    t2=time.perf_counter()
    print(f'Finished in {t2-t1} seconds')
```