Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра автоматизації та комп'ютерно-інтегрованих технологій

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

АВТОМАТИЗОВАНА СИСТЕМА РОЗПОДІЛУ ТА МАРШРУТИЗАЦІЇ ВОЛОНТЕРІВ ПІД ЧАС ПРИРОДНО ТЕХНІЧНИХ КАТАСТРОФ (NATECH)

AUTOMATED SYSTEM FOR THE ALLOCATION AND ROUTING OF VOLUNTEERS DURING NATURAL-TECHNOLOGICAL DISASTERS (NATECH)

Спеціальність 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка
освітня програма «Автоматизація та комп'ютерно-інтегровані технології»

Виконав: здобувач вищої освіти
групи АВм - 21
Шишковський Станіслав Олександрович
_____
(підпис)

Керівник:
к. т. н., доцент
Гуменюк Лариса Олександрівна
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Гуменюк П. О.

_____
(підпис)

Луцьк – 2025

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних та інформаційних технологій
Кафедра автоматизації та комп'ютерно-інтегрованих технологій
Ступінь вищої освіти: *магістр*
Галузь знань: *17 Електроніка, автоматизація та електронні комунікації*
Спеціальність: *174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка*
Освітня програма: *«Автоматизація та комп'ютерно-інтегровані технології»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О. Ю. Повстяной
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Шишковського Станіслава Олександровича
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: *Автоматизована система розподілу та маршрутизації волонтерів під час природно технічних катастроф (NaTech)*
Керівник роботи: *к.т.н., доцент Гуменюк Лариса Олександрівна*
затверджені наказом закладу вищої освіти від « 27 » 06 2025 року N 304/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи: « 1 » 12 2025 року
3. Вихідні дані до роботи: дані з Internet

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
Аналіз ризиків техногенних катастроф NaTech у міському середовищі. Методи та моделі розподілу та маршрутизації волонтерів під час катастроф. Використання Python у реалізації алгоритмів розподілу та маршрутизації волонтерів. Порівняльний аналіз алгоритмів пошуку сусідів у заданому радіусі. Алгоритм пошуку волонтерів у заданій області та його програмна реалізація. Алгоритми пошуку маршруту з мінімальною довжиною до цілі. Сервіси OSRM, GraphHopper та OpenRouteService пошуку маршруту волонтерів до цілі. Автоматизована система розподілу та маршрутизації волонтерів.
5. Перелік графічного матеріалу :
графічний матеріал виконано у вигляді презентації, яка складається з 17 слайдів

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1	*Гуменюк Л. О.*		
Розділ 2	*Гуменюк Л. О.*		
Розділ 3	*Гуменюк Л. О.*		
Розділ 4	*Гуменюк Л. О.*		
Розділ 5	*Гуменюк Л. О.*		
Розділ 6	*Гуменюк Л. О.*		
Розділ 7	*Гуменюк Л. О.*		
Розділ 8	*Гуменюк Л. О.*		
Нормоконтроль	*Лапченко Ю. С.*		
Показник запозичень тексту			
Академічна доброчесність	*Федік Л. Ю.*		

7. Дата видачі завдання _____*27.06.2025 р.*_____

КАЛЕНДАРНИЙ ПЛАН

N з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми за темою роботи та постановка задач	01.09.2025 р.	
2	Аналіз і вибір напрямків дослідження	10.09.2025 р.	
3	Теоретичне дослідження та практична реалізація	20.09.2025 р.	
4	Опис засобів розробки об’єкта проєктування	01.10.2025 р.	
5	Загальні висновки та рекомендації	20.10.2025 р.	
6	Оформлення роботи	10.11.2025 р.	
7	Оформлення презентації	20.11.2025 р.	
8	Здача чистового варіанту кваліфкаційної роботи на	01.12.2025 р.	
	кафедру		

Здобувач вищої освіти _____ _____Шишковський С. О._____
(підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи _____ _____Гуменюк Л. О._____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Шишковський С. О. Автоматизована система розподілу та маршрутизації волонтерів під час природно технічних катастроф (NaTech). Рукопис.

Кваліфікаційна робота магістра ОП «Автоматизація та комп'ютерно-інтегровані технології» спеціальності 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка. Луцький національний технічний університет, Луцьк 2025.

Кваліфікаційна робота магістра складається з вступу, восьми розділів, загальних висновків та рекомендацій, переліку використаних джерел та додатків.

У кваліфікаційній роботі здійснено дослідження розподілу та маршрутизації волонтерів у контексті техногенних катастроф типу NaTech. З огляду на характер таких катастроф обґрунтовано необхідність створення інструментів, здатних адаптуватися до швидкоплинної, непередбачуваної ситуації в умовах високої щільності міського середовища, зруйнованої інфраструктури та каскадного розвитку подій.

Розроблена автоматизована система розподілу та маршрутизації волонтерів під час катастроф NaTech виступає як гнучкий інструмент для ухвалення рішень у надзвичайних ситуаціях, забезпечуючи оперативне залучення ресурсів завдяки інтеграції методів геоаналітики, алгоритмічної оптимізації та інтуїтивно зрозумілого візуального середовища.

Результати дослідження можуть бути використані для покращення координації волонтерської допомоги в умовах криз, інтеграції у цифрові платформи реагування на надзвичайні події.

Об'єм графічної частини магістерської роботи складає 17 слайдів. Обсяг пояснювальної записки становить 91 друкована сторінка.

Ключові слова: природно технічні катастрофи (NaTech), волонтерська координація, маршрутизація ресурсів, моделювання, автоматизована система.

ANNOTATION

Shishkovsky S. Automated system for the allocation and routing of volunteers during natural-technological disasters (NaTech). Manuscript.

Master's qualification work of OP "Automation and computer-integrated technologies" specialty 174 Automation, computer-integrated technologies and robotics. Lutsk National Technical University, Lutsk 2025.

The master's qualification work consists of an introduction, eight chapters, general conclusions and recommendations, a list of references and appendices.

The thesis examines the distribution and routing of volunteers in the context of man-made disasters such as NaTech. Given the nature of such disasters, the need to create tools capable of adapting to rapidly changing, unpredictable situations in high-density urban environments with destroyed infrastructure and cascading events is justified.

The automated system developed for the distribution and routing of volunteers during disasters, NaTech, acts as a flexible tool for decision-making in emergency situations, ensuring the rapid mobilization of resources through the integration of geanalytics methods, algorithmic optimization, and an intuitive visual environment.

The results of the study can be used to improve the coordination of volunteer assistance in crisis situations and integration into digital emergency response platforms.

The volume of the graphic part of the master's thesis is 17 slides. The volume of the explanatory note is 91 printed pages.

Keywords: natural-technological disasters (NaTech), volunteer coordination, resource routing, modeling, automated system.

ЗМІСТ

ВСТУП

За період з 2020 по 2025 рік у світі було зафіксовано кілька значних техногенних катастроф, зокрема в урбанізованих районах, які мали серйозні наслідки для людства та навколишнього середовища.

Серед наймасштабніших техногенних трагедій останніх років – вибух складу аміачної селітри в Бейруті у 2020 році. Подія забрала життя 218 людей, спричинила масштабні руйнування в центрі міста та позбавила житла понад 300 тисяч мешканців. До цього ряду також належать: катастрофа на Каховській ГЕС (2023), яка призвела до затоплення понад 600 км² Херсонщини; вибух на хімічному підприємстві в Кельні (2021), що спричинив смерть однієї особи та поранення ще 31; масштабна залізнична катастрофа в Індії (2023), внаслідок якої загинуло 288 осіб. Ці події свідчать про нагальну потребу в скоординованих діях для зменшення наслідків подібних катастроф.

У 2024-2025 роках спостерігається зростання техногенних аварій через зміни клімату, старіння інфраструктури та недостатнє дотримання стандартів безпеки. Так, аварія на хімічному заводі в Х'юстоні (2024) призвела до витоку токсичних речовин, що викликало евакуацію понад 50 000 осіб. Обвал будівлі в Стамбулі (2025), що спричинив десятки постраждалих, ще раз підтверджує необхідність швидкої і ефективної організації допомоги постраждалим.

Ці події продемонстрували високий рівень уразливості сучасних інфраструктур та важливість ефективного реагування на надзвичайні ситуації. У цьому контексті особливо актуальним стає питання організації системи розподілу та маршрутизації волонтерів під час катастроф NaTech (техногенно-природних катастроф).

Метою дослідження є розробка автоматизованої системи розподілу та маршрутизації волонтерів, здатної ефективно функціонувати в умовах катастроф типу NaTech, шляхом поєднання алгоритмічних, просторових і логістичних

рішень для підтримки оперативного реагування на надзвичайні ситуації у міському середовищі.

Об’єктом дослідження виступає процес організації волонтерської допомоги під час комплексних катастроф, що виникають на перетині природних і техногенних загроз.

Предметом дослідження є моделі, методи та програмні реалізації для розподілу та маршрутизації волонтерів з урахуванням географічних та інфраструктурних факторів у динамічному середовищі катастроф.

Наукова новизна полягає у створенні адаптивної системи, яка здатна поєднувати алгоритмічний пошук волонтерів за просторовими критеріями, динамічну маршрутизацію з використанням сервісів на базі відкритих картографічних даних, а також забезпечувати інтегровану геопросторову візуалізацію для оперативної підтримки рішень у ситуаціях, що швидко змінюються.

Практична новизна: результати дослідження можуть бути використані для покращення координації волонтерської допомоги в умовах криз, інтеграції у цифрові платформи реагування на надзвичайні події.

У процесі виконання роботи необхідно вирішити такі задачі.

1. Дослідити ризики, що виникають внаслідок поєднання природних і техногенних катастроф у містах.

2. Проаналізувати існуючі методи і моделі координації волонтерських ресурсів, враховуючи спонтанний характер їх появи та логістичні обмеження.

3. Розробити та реалізувати алгоритми пошуку волонтерів у заданому радіусі із застосуванням структур просторового розподілу (KD-Tree, Ball-Tree, Quadtree).

4. Провести порівняльний аналіз алгоритмів маршрутизації (Дейкстра, A*, Флойд-Уоршелл) та сервісів побудови маршрутів (OSRM, GraphHopper, ORS) для забезпечення швидкого доступу волонтерів до місць потреби.

5. Розробити автоматизовану систему, яка включає візуалізацію на основі бібліотек Python, інтеграцію з базами даних та підтримку прийняття рішень.

6. Провести тестування ефективності розроблених підходів на модельних сценаріях із застосуванням симуляцій.

Апробація результатів дослідження. Опублікована стаття Шишковський С. О., Гуменюк Л. О. Методи та моделі розподілу та маршрутизації волонтерів під час катастроф. *Перспективні технології та прилади. Збірник статей*. Луцьк: ЛНТУ, 2025. Випуск 27. С. 84-89.

РОЗДІЛ 1
АНАЛІЗ РИЗИКІВ ТЕХНОГЕННИХ КАТАСТРОФ NATECH У МІСЬКОМУ СЕРЕДОВИЩІ

У сучасному світі, де спостерігається безпрецедентна концентрація населення та критичної інфраструктури в міських агломераціях, зростає потенціал значних наслідків як від техногенних катастроф, так і від подій, спричинених природними небезпеками. Техногенна катастрофа визначається як подія, що виникає внаслідок технологічних або промислових аварій, небезпечних процедур, відмов інфраструктури або певних видів людської діяльності. Міське середовище, зі свого боку, характеризується високою щільністю населення та складними взаємопов’язаними системами інфраструктури. Природна небезпека являє собою подію або процес природного походження, здатний завдати шкоди. Особливу увагу привертають катастрофи NaTech (Natural Hazard-triggered Technological Disaster), які є технологічними аваріями, спровокованими природними небезпеками [1].

Зростаюча актуальність техногенних катастроф, а особливо NaTech у міських умовах, зумовлена кількома ключовими факторами.

По-перше, висока концентрація населення та критичної інфраструктури в містах означає, що будь-яка аварія може призвести до значних людських та економічних втрат.

По-друге, сучасні міські системи є надзвичайно взаємопов’язаними, де відмова в одній галузі може швидко призвести до каскадних відмов в інших, потенційно спричиняючи або посилюючи техногенні катастрофи.

По-третє, концепція катастроф NaTech підкреслює складний та часто непередбачуваний характер ризиків, де вплив природної небезпеки посилюється через відмову технологічних систем [2].

1.1 Класифікація міських техногенних катастроф

Техногенні катастрофи, які можуть виникнути в міському середовищі, можна класифікувати за різними критеріями, зокрема за джерелом та за характером їхнього впливу.

За джерелом (сферою виникнення) можна виділити такі основні категорії.

- Промислові аварії: включають хімічні витоки, вибухи на заводах, пожежі на промислових об’єктах.
- Аварії в енергетиці: охоплюють інциденти на атомних електростанціях, розриви нафто- та газопроводів, масштабні відмови в електромережах.
- Транспортні аварії: включають великомасштабні дорожньо-транспортні пригоди, залізничні аварії, авіакатастрофи або морські інциденти, особливо ті, що пов’язані з перевезенням небезпечних вантажів або значними пошкодженнями інфраструктури.
- Аварії в будівництві та інфраструктурі: охоплюють обвалення будівель, руйнування мостів, прориви дамб, обвали тунелів.
- Кіберінциденти: включають атаки на системи управління критичною інфраструктурою, що призводять до фізичних пошкоджень або порушень функціонування.

За масштабами наслідків розрізняють:

– локальні – пошкодження об’єкта або невеликої території (наприклад, витік хімічних речовин на заводі),

– регіональні – впливають на велику територію (аварія на АЕС, велика залізнична катастрофа),

– глобальні – наслідки відчутні у всьому світі (наприклад, ядерна війна, масштабні кібератаки на критичну інфраструктуру).

За характером впливу техногенні катастрофи можна поділити на:

- негайний вплив: включає загибель людей, травмування, безпосередні матеріальні збитки, забруднення навколишнього середовища,

– вторинний вплив: охоплює переміщення населення, економічні збої, кризи в галузі охорони здоров’я, довгострокові екологічні наслідки [3].

Важливо зазначити, що різні типи техногенних катастроф можуть бути взаємопов’язані та спричиняти каскадні ефекти. Наприклад, вибух може призвести до пожежі, а відключення електроенергії може спричинити відмову критично важливих служб.

Класифікація техногенних катастроф не завжди є чіткою та однозначною. Одна подія може одночасно належати до кількох категорій. Наприклад, сходження з рейок потяга, що перевозить хімічні речовини, є одночасно транспортною аварією та промисловою небезпекою з потенційним хімічним забрудненням.

Зростаюча залежність сучасних міст від взаємопов’язаних цифрових систем призводить до появи нової категорії техногенних катастроф, пов’язаних з кібератаками. Ці атаки можуть мати далекосяжні фізичні наслідки, підкреслюючи вразливість сучасної міської інфраструктури до цифрових загроз. Успішна кібератака на системи управління критичною інфраструктурою, такі як електромережі або водоочисні споруди, може призвести до значних фізичних пошкоджень та порушень, стираючи межі між традиційними технологічними відмовами та катастрофами, спричиненими кіберінцидентами [4-5].

1.2 Катастрофи NaTech у міському середовищі

Катастрофи NaTech виникають на перетині природних небезпек та технологічних вразливостей. Різноманітні природні явища, такі як землетруси, повені, шторми, екстремальні температури та лісові пожежі, можуть спровокувати відмови в технологічних системах. Наприклад, землетрус може пошкодити промислові об’єкти, призводячи до витоку небезпечних речовин, повінь може затопити електропідстанції, спричиняючи масштабні відключення

електроенергії, а сильний шторм може пошкодити нафто- та газопроводи, викликаючи витоки.

Катастрофи NaTech у міському середовищі мають унікальні характеристики, які відрізняють їх від звичайних техногенних катастроф. Вища щільність населення в містах призводить до більшого потенціалу жертв та травмувань. Концентрація критичної інфраструктури збільшує ризик широкомасштабних порушень. Складні взаємозалежності між міськими системами можуть посилити вплив подій NaTech, спричиняючи каскадні відмови. Крім того, висока щільність забудови та пошкодження інфраструктури можуть ускладнити проведення рятувальних робіт та евакуацію населення.

У сценаріях NaTech часто спостерігається ефект доміно, коли первинна природна небезпека ініціює ланцюг технологічних відмов. Наприклад, землетрус може пошкодити хімічний завод, що призведе до викиду токсичних речовин та подальших пожеж. Цей каскадний характер подій значно ускладнює прогнозування та управління наслідками катастроф NaTech.

Міські райони часто стикаються з підвищеним базовим ризиком як природних небезпек (наприклад, прибережні міста та урагани, сейсмоактивні зони), так і технологічних аварій через концентрацію промислових та інфраструктурних об’єктів. Таке поєднання вразливостей значно підвищує ризик катастроф NaTech у містах порівняно з менш розвиненими або сільськими районами. Економічна активність у містах зумовлює високу концентрацію промислових підприємств та інфраструктури, що в поєднанні з розташуванням у зонах, схильних до певних природних небезпек, робить міські території особливо вразливими до подій NaTech.

Час та інтенсивність природної небезпеки можуть суттєво впливати на ймовірність та серйозність техногенної катастрофи. Наприклад, землетрус, що стався в години пікової промислової активності, може призвести до більш тяжких наслідків, ніж той, що стався вночі, коли більшість підприємств не працює. Операційний стан та присутність людей на технологічних об’єктах під

час природної небезпеки є критичними факторами, що можуть посилити наслідки технологічної відмови, спровокованої природною подією [6].

1.3 Вразливість міської інфраструктури до техногенних та NaTech катастроф

Міська інфраструктура є основою функціонування сучасних міст. Вона охоплює критично важливі сектори. Одним із ключових напрямків є енергетика. До неї входять об’єкти генерації та передачі електроенергії, а також мережі транспортування нафти й газу.

Транспортна система також є важливою складовою. Вона включає дороги, мости, залізниці, аеропорти, порти та громадський транспорт. Водопостачання та водовідведення забезпечують життєдіяльність міста. Ці системи складаються з дамб, водосховищ, очисних споруд та каналізаційної інфраструктури.

Засоби зв’язку відіграють ключову роль у функціонуванні міста. Вони забезпечують телекомунікації, доступ до інтернету та екстрений зв’язок. Охорона здоров’я є невід’ємною частиною міської інфраструктури. Її представляють лікарні, клініки та служби швидкої медичної допомоги.

Вразливість міської інфраструктури до техногенних та природно-техногенних (NaTech) катастроф формується сукупністю факторів, які взаємодіють між собою. Якщо інфраструктура є застарілою або технічно зношеною, її здатність протистояти аваріям значно зменшується – конструкції можуть втрачати міцність, інженерні мережі стають ненадійними, а системи безпеки – малоефективними. Порушення або недосконаліть проєктних і будівельних норм створюють умови, за яких об’єкти не витримують навантажень, що виникають під час землетрусів, повеней чи інших екстремальних подій. Відсутність резервних або дублюючих систем, таких як альтернативне електроживлення чи водопостачання, збільшує час відновлення після аварій і сприяє поширенню наслідків на великі території.

Важливу роль відіграє й взаємозалежність між різними секторами – наприклад, вихід з ладу енергомережі може паралізувати транспорт, зв'язок, водопостачання й охорону здоров'я. Географічне положення інфраструктурних об'єктів у районах із підвищеною небезпекою – поблизу зон підтоплення, зсувів, розломів чи промислових зон – додатково підвищує ризик виникнення та масштаби катастроф. Усі ці чинники разом створюють складний і вразливий техно-природний ландшафт, у якому навіть локальна подія може мати масштабні наслідки.

Існують численні приклади вразливостей міської інфраструктури. Електромережі можуть бути пошкоджені сильними вітрами або землетрусами, що призводить до масштабних відключень електроенергії та впливає на роботу інших служб. Транспортні мережі вразливі до повеней або зсувів ґрунту, що може порушити ланцюги постачання та ускладнити доступ для служб екстреної допомоги. Водоочисні споруди можуть бути забруднені під час повеней, що призводить до криз у галузі охорони здоров'я.

Зростаюча залежність міських економік від своєчасних поставок та ощадливих ланцюгів постачання робить їх особливо вразливими до порушень у транспортній інфраструктурі, спричинених як техногенними, так і NaTech катастрофами. Навіть локалізовані пошкодження можуть мати значний ланцюговий вплив на доступність основних товарів та послуг. Перебої в роботі транспортних мереж, незалежно від того, чи є вони результатом транспортної аварії, чи пошкодження інфраструктури природною небезпекою, можуть швидко призвести до дефіциту продовольства, палива та інших критично важливих товарів, впливаючи на повсякденне життя міських мешканців та ускладнюючи зусилля з відновлення.

Вразливість міської інфраструктури часто посилюється через відсутність інтегрованого планування та координації між різними секторами інфраструктури та відповідальними органами. Розрізнені підходи до управління можуть призвести до ігнорування взаємозалежностей та неврахованих вразливостей.

Різні сектори інфраструктури (енергетика, транспорт, водопостачання, зв’язок) часто управляються окремими організаціями зі своїми пріоритетами та планами. Відсутність цілісного, інтегрованого планування може призвести до нездатності розпізнати та усунути критичні взаємозалежності, що робить міську систему в цілому більш вразливою до каскадних відмов під час катастрофи.

1.4 Природні і техногенні катастроф в період з 2020 по 2025 рік

У період з 2020 по 2025 рік зафіксовано помітне зростання кількості природних катастроф, що часто спричиняли й техногенні інциденти, сягнувши, за даними звіту «Global Natural Disaster Assessment Report» [6], 432 подій у 2021 році. У 2020 році, за різними оцінками, відбулося понад 300 великих природних катастроф, серед яких найчастішими були повені (193 випадки) та шторми (69 випадків). Екологічний реєстр загроз зафіксував 396 інцидентів ще у 2019 році, що вказує на значне зростання порівняно з попередніми десятиліттями. Дані за 2025 рік свідчать про продовження цієї тенденції, з повідомленнями про понад 400 значних природних катастроф по всьому світу [7].

Економічні наслідки цих катастроф досить відчутні, досягнувши, за оцінками, від 151,6 млрд доларів до 268 млрд доларів у 2020 році. У 2021 році економічні втрати зросли до 252,1 млрд доларів, а за даними звіту [6], прямі економічні втрати були на 82 % вищими за середній показник за 30 років. У 2024 році глобальні втрати від природних катастроф сягнули 320 млрд доларів, а за оцінками Gallagher Re, ця цифра може сягати 417 млрд доларів у 2025 році. У США кількість погодних та кліматичних катастроф, збитки від кожної з яких перевищили 1 мільярд доларів, досягла рекордних 28 випадків у 2023 році [2].

Зміна клімату продовжувала відігравати важливу роль у формуванні цих подій. 2025 рік, ймовірно, стане найтеплішим за всю історію спостережень. Збільшення температури поверхні океану також сприяло посиленню тропічних

циклонів. У 2020 році UNDRR зафіксувало 389 подій, пов’язаних зі зміною клімату [1].

Щодо техногенних катастроф, за даними IFRC, у 2020 році сталося 151 таке лихо, а у 2021 році – 101. Природні катастрофи часто призводили до техногенних аварій. Наприклад, у квітні 2021 року в індійському штаті Уттаракханд зсув, спричинений обвалом льодовика, пошкодив гідроенергетичну інфраструктуру, що призвело до загибелі близько 234 осіб. Землетрус у Японії у 2022 році спричинив витік води з АЕС Фукусіма, а землетрус на півострові Ното у 2024 році призвів до збоїв у роботі промислових підприємств.

Дані за період з 2020 по 2025 рік підтверджують зростання кількості та руйнівності природних катастроф, а також збільшення економічних втрат, особливо у 2021 та 2024 роках. Зростання кількості мільярдних катастроф у США, де у 2023 році було зафіксовано рекордні 28 таких подій, також підкреслює цю тенденцію. Річні економічні втрати коливалися від 151,6 млрд доларів до 417 млрд доларів, що значно перевищує середні показники попередніх десятиліть. Отже, аналіз даних за період з 2020 по 2025 рік підтверджує тенденцію до зростання кількості та руйнівності природних катастроф, що призводить до значних економічних втрат [6].

Протягом цього періоду спостерігалися приклади ефективного управління ризиками стихійних лих. Індонезія розробила інтегровану систему раннього попередження про повені в Джакарті, а місто Макаті на Філіппінах створило Центр управління надзвичайними ситуаціями. Світовий банк використовував фінансові інструменти, такі як Cat DDO [8], для допомоги країнам, постраждалим від лих, наприклад, у Малаві у 2024 році. Японія успішно застосувала систему для доставки перевіреної інформації про лихо після землетрусу 2024 року [9]. Такі практики доводять, що адаптація та стратегічне планування можуть знижувати втрати, навіть на тлі посилення катастроф.

Висновок до розділу 1

Висока концентрація населення та інфраструктури посилює вразливість перед природними й техногенними загрозами, особливо коли вони взаємодіють між собою, утворюючи складні каскадні ланцюги подій із непередбачуваними наслідкам. Катастрофи NaTech вирізняються складністю реагування через залежності в інфраструктурі та її деградацію.

Період 2020-2025 років засвідчує зростання природних катаклізмів, що часто спричиняють техногенні аварії. У цьому контексті розробка автоматизованої системи координації волонтерів є актуальним і практичним кроком для покращення організації допомоги та швидшого реагування під час надзвичайних ситуацій у містах.

РОЗДІЛ 2
МЕТОДИ ТА МОДЕЛІ РОЗПОДІЛУ ТА МАРШРУТИЗАЦІЇ ВОЛОНТЕРІВ ПІД ЧАС КАТАСТРОФ

Управління наслідками катастроф та надання гуманітарної допомоги є складним і багатогранним завданням, у якому волонтери відіграють надзвичайно важливу роль. Їхня здатність швидко мобілізуватися та надавати різноманітну підтримку, від спеціалізованої допомоги до виконання простих завдань, робить їх безцінним ресурсом у кризових ситуаціях. Волонтери здатні забезпечити значне збільшення потенціалу реагування, яке не завжди можуть забезпечити лише офіційні служби. Вони можуть виконувати широкий спектр завдань, полегшуючи навантаження на персонал служб надзвичайних ситуацій та дозволяючи їм зосередитися на більш спеціалізованих видах діяльності [10].

У зв’язку зі зростаючою частотою та інтенсивністю масштабних катастроф, зростає і науковий інтерес до оптимізації управління волонтерами. Більша кількість катастроф призводить до більшої потреби в ефективному реагуванні, що, у свою чергу, підкреслює велику роль волонтерів.

Хоча волонтери визнаються цінним ресурсом, їхня ефективна інтеграція створює значні логістичні проблеми, які іноді називають «катастрофою всередині катастрофи» [11].

Волонтери можуть бути спонтанними або зареєстрованими. Спонтанні волонтери, як правило, допомагають без офіційної реєстрації, діючи ситуативно в кризових ситуаціях. Хоча вони можуть не мати офіційного статусу, їхня допомога часто є надзвичайно цінною. З іншого боку, зареєстровані волонтери оформлюються через офіційні організації (благодійні фонди, волонтерські центри), що дає їм доступ до інструкцій, ресурсів та підтримки.

Основні відмінності між спонтанними та зареєстрованими волонтерами полягають у їхній організації, підтримці та відповідальності. Зареєстровані волонтери діють у системному порядку, що дозволяє ефективно інтегрувати їх у

робочі процеси, тоді як спонтанні волонтери зазвичай залучаються ситуативно, залежно від нагальної потреби.

Офіційно зареєстровані волонтери мають доступ до необхідних ресурсів, таких як харчування, засоби захисту та інші матеріали, що підвищує їхню ефективність. Спонтанні волонтери ж часто не мають такої підтримки. Крім того, зареєстровані волонтери мають чітко визначені обов'язки та проходять інструктаж, що дозволяє їм діяти більш злагоджено і в рамках організованих заходів, в той час як спонтанні волонтери не завжди мають таку підготовку.

2.1 Класифікація методів та моделей розподілу волонтерів

Для оптимізації розподілу та маршрутизації волонтерів під час катастроф використовується широкий спектр моделей та методів. Основні категорії включають оптимізаційні моделі, моделі черг, агентно-орієнтовані симуляції та методи прийняття рішень.

2.1.1 Оптимізаційні моделі

Оптимізаційні моделі спрямовані на пошук найкращого можливого рішення для розподілу волонтерів з урахуванням певних обмежень та цілей.

А. Моделі лінійного та цілочисельного програмування

Моделювання задач призначення волонтерів на завдання під час ліквідації наслідків катастроф часто базується на лінійному та цілочисельному програмуванні, що дозволяє оптимізувати розподіл ресурсів за визначеними критеріями.

У [12] запропоновано модель для підвищення продуктивності та утримання волонтерів, враховуючи змінні потреби постраждалих і розвиток навичок волонтерів. Автори [13] розробили модель цілочисельного програмування для оптимального розподілу завдань між волонтерами та працівниками неприбуткових організацій, акцентуючи на задоволеності волонтерів, що є критичним для їхнього утримання. В роботі [14] розроблено

модель для призначення рятувальників в різні райони катастроф з урахуванням їхніх кваліфікацій та уподобань, що дозволяє ефективно розподіляти ресурси в різних зонах. Ряд інших моделей розглядають оптимізацію початкового розподілу як професійних рятувальників, так і волонтерів під час катастроф, що дозволяє забезпечити ефективне реагування на надзвичайні ситуації. Також пропонується зосередилися на попередньому розподілі завдань для мінімізації часу реагування, що є критично важливим у кризових умовах [14-15].

Перевага цих моделей полягає в здатності знаходити оптимальні рішення для чітко визначених проблем. Однак вони можуть бути складними для розв’язання в масштабних ситуаціях і не завжди враховують динаміку змін.

Б. Багатокритеріальні моделі оптимізації

У [16] запропоновано багатокритеріальну модель оптимізації для управління волонтерами в гуманітарних організаціях, яка враховує переваги волонтерів та вимоги до завдань, забезпечуючи баланс між ефективністю виконання завдань і збереженням волонтерів. Для підвищення ефективності застосовано нечітку логіку (fuzzy logic). Модель враховує компроміси між потенційно конфліктуючими цілями, такими як ефективність виконання завдань і задоволеність волонтерів. Автори [10] розробили модель багатоцільової оптимізації для координації волонтерів у динамічному й невизначеному середовищі, спрямовану на одночасну оптимізацію кількох цілей в умовах катастроф. В [17] автори використали багатокритеріальну оптимізацію з нечіткими системами виводу для покращення управління волонтерами після катастроф, акцентуючи на обробці невизначеності через нечітку логіку.

Розробка таких моделей демонструє, що реагування на катастрофи передбачає збалансування різних важливих факторів, при цьому враховується мінімізація незадоволених потреб, максимізація задоволеності волонтерів та забезпечення справедливості. Використання нечіткої логіки підкреслює наявність невизначеності в управлінні волонтерами, зокрема в контексті суб’єктивних переваг та неточних вимог до завдань.

В. Моделі робастної оптимізації

Модель на основі стійкої оптимізації [18] призначена для динамічного управління волонтерами під час гуманітарних криз, мінімізуючи невиконані завдання з урахуванням переваг та навичок волонтерів в умовах змін. Ця модель явно враховувала невизначеність у вимогах до завдань, використовуючи підхід робастної оптимізації.

В роботі [19] автори застосували теорію Демпстера–Шафера для оптимального призначення рятувальників за невизначеності. Це дослідження використовувало теорію Демпстера-Шафера для обробки невизначеності при призначенні рятувальників.

Катастрофи завжди пов’язані з невизначеністю – змінюється попит на допомогу та кількість доступних волонтерів. Робастна оптимізація допомагає знайти рішення, які залишаються ефективними навіть у складних і нестабільних умовах.

2.1.2 Моделі черг. Багатоканальні системи черг

Моделі черг використовуються для аналізу систем, де об’єкти (у цьому випадку волонтери) прибувають випадковим чином і потребують обслуговування (призначення завдань).

Автори [20] змоделювали ситуацію, коли спонтанні волонтери приходять у різний час і в різній кількості, як багатоканальну систему черг. У такій системі волонтери прибувають випадково, можуть не дочекатися призначення і піти (тобто «відмовитися»). Для оцінки варіантів розподілу вони використали марковські процеси прийняття рішень – математичний інструмент, який дозволяє вибрати найкращу стратегію, враховуючи випадкові зміни у середовищі. Модель дозволяє зрозуміти, як зміна політики розподілу (наприклад, кого призначати першим) впливає на загальну ефективність.

В роботі [21] розроблено схожу модель черг, де також застосовуються марковські процеси, але додатково використовуються комп’ютерні симуляції. Це дозволило протестувати різні стратегії у змодельованих умовах катастроф, де

ситуація постійно змінюється. Основна мета – знайти способи швидко приймати, перевіряти та ефективно призначати спонтанних волонтерів на відповідні завдання.

Автори [22] також використали модель черг, орієнтовану на роботу в умовах невизначеності. Вони застосували марковські моделі для вирішення проблем призначення волонтерів, коли точно невідомо, скільки волонтерів надійде і які завдання будуть доступні. Це дозволило аналізувати ситуації, коли є обмежені ресурси та необхідно приймати рішення на основі неповної інформації.

Загалом, усі три дослідження трактують управління спонтанними волонтерами як задачу потоків: волонтери надходять у систему, можливо чекають, і врешті – обслуговуються (призначаються на завдання). Теорія черг дозволяє оцінювати ключові показники – тривалість очікування, довжину черги, завантаженість координаторів.

2.1.3 Агентно-орієнтовані симуляції (АОС)

Агентно-орієнтовані симуляції (АОС) використовуються для моделювання поведінки окремих волонтерів і їхньої взаємодії в умовах катастроф. У [23] застосували АОС для оцінки евристичних стратегій призначення спонтанних волонтерів у центрах допомоги з метою зменшення часу перебування як волонтерів, так і отримувачів допомоги. Автори [24] використали АОС для аналізу управління напливом волонтерів і формування стратегічних рішень у режимі невизначеності. Автори [25] створили децентралізовану модель рекомендацій завдань для волонтерів на основі їхніх переваг, що відповідає принципам АОС.

Використання таких симуляцій дозволяє враховувати індивідуальні особливості, мотивації та поведінкові правила волонтерів. Це розширює розуміння не лише механічного розподілу завдань, як у класичних моделях, а й соціальної динаміки участі у волонтерстві. АОС дає змогу дослідити причини залучення волонтерів, особливості їхніх рішень, а також наслідки різних

управлінських стратегій, що робить цей підхід особливо цінним для покращення комунікації, залучення та утримання волонтерів у кризових умовах.

2.1.4 Моделі та методи прийняття рішень

Ця категорія включає різні підходи, які допомагають особам, що приймають рішення, ефективно розподіляти та спрямовувати волонтерів.

А. Еврістичний підхід

Знання того, що прості евристики можуть працювати порівняно з складними оптимальними стратегіями, має важливі практичні наслідки для управління надзвичайними ситуаціями. Це свідчить про те, що організації з обмеженими ресурсами можуть не завжди потребувати впровадження складних моделей для досягнення хороших результатів. Хоча оптимізаційні моделі можуть забезпечити теоретично оптимальні рішення, вони часто вимагають значних даних та обчислювальних ресурсів. Той факт, що прості правила можуть бути майже такими ж ефективними, робить управління волонтерами більш доступним та практичним для організацій з обмеженими можливостями [23, 26].

Б. Двосторонні моделі відповідності

Двосторонні моделі відповідності враховують переваги як волонтерів, так і координаторів завдань, щоб забезпечити ефективне та взаємовигідне призначення. Вони дозволяють працювати навіть за неповної або неточної інформації.

Окремі підходи також враховують емоційні аспекти, як-от радість і розчарування волонтерів, або поєднують сучасні методи оптимізації (наприклад, DEA, диференціальну еволюцію, EBRB) для підвищення точності призначень. Загалом цей напрямок відображає зсув до більш людоцентричного управління волонтерами – з урахуванням їх мотивацій, очікувань і задоволеності [27-28].

2.2 Аналіз факторів, що впливають на розподіл та маршрутизацію волонтерів

Різноманітні моделі [10-28], що застосовуються в контексті розподілу та маршрутизації волонтерів, характеризуються комплексним врахуванням широкого спектра факторів. Зокрема, моделі систематично враховують компетентності та кваліфікації волонтерів з метою оптимізації виконання завдань та підвищення ймовірності успішного досягнення поставлених цілей, що також сприяє підвищенню рівня задоволеності та утримання волонтерського корпусу. Крім того, важливим елементом є врахування індивідуальних умінь та часової доступності волонтерів для забезпечення високого рівня мотивації та залученості, прагнучи до узгодження особистих схильностей з актуальними потребами.

Центральним аспектом багатьох моделей є пріоритетність потреб постраждалого населення та специфіка завдань, орієнтуючись на задоволення найнагальніших потреб осіб, які постраждали внаслідок кризової ситуації, шляхом пріоритезації завдань на основі оцінки їхньої критичності.

Управління волонтерськими ресурсами часто відбувається в умовах часових обмежень, що зумовлює необхідність їхнього врахування в моделях, а динамічний характер катастроф вимагає від моделей здатності до адаптації у відповідь на зміни в умовах та потребах.

Моделювання процесів розподілу волонтерів повинно враховувати притаманну кризовим ситуаціям невизначеність щодо кількості доступних волонтерів, їхніх компетенцій та еволюціонуючих потреб, а також розглядати потенційні ризики, пов’язані з виконанням завдань у зоні катастрофи.

Оптимізація розподілу та маршрутизації волонтерів передбачає аналіз геопросторових характеристик та логістики переміщення, включаючи географічне розташування волонтерів, місць виконання завдань та транспортної

інфраструктури, де мінімізація часу на пересування є критично важливим фактором.

2.3 Методи ефективного залучення та призначення завдань спонтанним волонтерам

Спонтанні волонтери є учасниками реагування на надзвичайні ситуації без попередньої підготовки чи організаційної приналежності. Проте їхня непередбачуваність щодо прибуття, різноманітність навичок та відсутність попередньої координації створюють значні управлінські виклики. Це потребує спеціалізованих моделей розподілу та маршрутизації для ефективного використання цього ресурсу.

Для оптимізації процесу призначення волонтерів у ситуаціях обмежених ресурсів та високої динаміки потреб використовуються моделі черг, тоді як агентно-орієнтоване моделювання дозволяє імітувати поведінку волонтерів у динамічному середовищі для тестування різних стратегій управління. Моделі рекомендацій надають волонтерам можливість самостійно обирати завдання відповідно до їхніх навичок та інтересів, що сприяє підвищенню ефективності та мотивації. У складних умовах, де необхідно балансувати між різними потребами організацій та можливостями волонтерів у змінному середовищі, застосовується багатоцільова оптимізація.

Існують різні стратегії для залучення та призначення завдань спонтанним волонтерам.

Централізоване адміністрування передбачає створення єдиної системи координації для зменшення хаосу та затримок у розподілі завдань. Волонтерські центри прийому забезпечують місце для реєстрації, навчання та інструктажу, що дозволяє ефективніше зіставляти навички волонтерів з наявними потребами.

Використання технологій, таких як SMS-розсилки, мобільні додатки та онлайн-платформи, спрощує комунікацію та процес розподілу завдань.

В умовах обмеженого часу можуть бути ефективними евристичні методи розподілу, що базуються на простих правилах, наприклад, призначення найближчого доступного волонтера. Крім того, гнучкі рекомендаційні системи, які пропонують завдання замість прямого призначення, сприяють самостійності волонтерів та покращують їхню мотивацію.

2.4 Врахування динамічного характеру катастроф

Реагування на катастрофи є не статичним процесом, а розвивається з часом, оскільки потреби змінюються, а наявність волонтерів коливається. Це зумовлює необхідність використання моделей, які можуть адаптуватися до цих мінливих обставин.

Динамічне управління волонтерами передбачає застосування методів, що враховують змінність потреб і характеристик учасників процесу. Розроблені моделі оптимізації дозволяють ефективно спрямовувати волонтерів на завдання, враховуючи їхні навички та ймовірність виконання завдань.

У непередбачуваних умовах надзвичайних ситуацій особливо важливо мати рішення, стійкі до невизначеності. Прогнозування потреб у волонтерах ґрунтується на аналізі історичних даних із застосуванням машинного навчання. Такі підходи дозволяють заздалегідь оцінити обсяги необхідної допомоги, спираючись на досвід попередніх катастроф, що сприяє більш ефективній підготовці та розподілу ресурсів.

Адаптація до змін у ході реагування включає оновлення розкладів, перепризначення завдань і повторну оцінку ресурсів. Стійкі моделі забезпечують збереження ефективності навіть у разі суттєвих відхилень від початкових умов, дозволяючи системі реагування залишатися гнучкою та стійкою.

Узагальнюючи, сучасні підходи до управління волонтерами в умовах катастроф базуються на трьох ключових принципах: динамічності,

прогнозуванні та стійкості. Це забезпечує ефективність реагування навіть в умовах високої невизначеності та обмежених ресурсів.

Висновок до розділу 2

Проведено аналіз існуючих методів та моделей, що застосовуються для розподілу та маршрутизації волонтерів під час ліквідації наслідків катастроф. Огляд літератури підтвердив важливість волонтерів як ресурсу для підсилення спроможностей офіційних служб реагування та показав логістичні виклики, пов’язані з їх ефективною інтеграцією, особливо у випадку спонтанних волонтерів.

Управління волонтерськими ресурсами базується на різних підходах, що враховують як ефективність, так і людський фактор.

Оптимізаційні моделі дають змогу знаходити найкращі рішення щодо розподілу завдань з урахуванням продуктивності, справедливості, задоволеності та інших критеріїв, навіть в умовах невизначеності.

Теорія черг дозволяє аналізувати потоки волонтерів, оцінювати завантаження системи та перевіряти політики призначення за допомогою симуляцій.

Агентно-орієнтовані моделі моделюють поведінку окремих волонтерів, виявляючи мотиваційні та соціальні фактори.

Евристичні методи та моделі відповідності забезпечують швидке прийняття рішень і враховують переваги як волонтерів, так і завдань, формуючи людиноцентричні стратегії управління.

Аналіз показав, що сучасні моделі прагнуть враховувати широкий спектр факторів, включаючи компетентності та вподобання волонтерів, пріоритетність потреб постраждалих, специфіку завдань, часові та геопросторові обмеження, логістику, динамічність ситуації, невизначеність та ризики.

Розглянуто методи залучення та призначення завдань спонтанним волонтерам, з акцентом на централізовану координацію, використання технологій (онлайн-платформи, мобільні додатки) та гнучких підходів, таких як рекомендаційні системи.

Враховано необхідність застосування адаптивних моделей, методів прогнозування на основі даних та розробки рішень, стійких до змін умов, через динамічний характер катастроф.

Огляд методів та моделей демонструє інтерес до розробки інструментів для оптимізації управління волонтерами. Виявлено тенденцію до інтеграції різних підходів, врахування невизначеності та динаміку кризових ситуацій. Результати цього аналізу створюють теоретичну основу для розробки алгоритмів та функціоналу автоматизованої системи розподілу та маршрутизації волонтерів.

РОЗДІЛ 3
ВИКОРИСТАННЯ PYTHON У РЕАЛІЗАЦІЇ АЛГОРИТМІВ РОЗПОДІЛУ ТА МАРШРУТИЗАЦІЇ ВОЛОНТЕРІВ

Розробка системи розподілу та маршрутизації волонтерів потребує ефективних алгоритмів для оптимізації часу, відстаней та ресурсів. Python, як універсальна мова програмування, надає широкий спектр бібліотек для реалізації таких алгоритмів.

Застосування цієї мови дозволяє створювати системи, які здатні адаптуватися до змінних умов, масштабуватися при збільшенні кількості волонтерів чи завдань і підтримувати інтеграцію з різними джерелами даних.

Однією з основ ефективної роботи з просторовими та числовими даними є бібліотека NumPy. Вона забезпечує високопродуктивну обробку координат та матричних операцій, необхідних для розрахунку відстаней, оцінки ефективності маршрутів та обчислень при оптимізації розподілу. Використання векторизованих обчислень дозволяє суттєво зменшити час виконання порівняно зі звичайними ітеративними методами.

Для точного визначення географічних відстаней між об’єктами широко застосовується бібліотека Geopy, яка надає доступ до кількох геокодувальних сервісів і дозволяє використовувати різні методи вимірювання, включно з формулою Haversine. Це особливо важливо у випадках, коли волонтери та завдання розташовані на великій території, де просте евклідове наближення не забезпечує достатньої точності.

Проблема знаходження найближчих волонтерів до певної задачі вирішується за допомогою scikit-learn. Завдяки алгоритмам KDTree та BallTree можливо здійснювати швидкий пошук найближчих сусідів у високовимірному просторі координат. Це дозволяє знаходити оптимальних кандидатів для виконання конкретних завдань.

У процесі інтеграції з зовнішніми сервісами – наприклад, базами даних волонтерських центрів або платформами координування – активно використовується бібліотека Requests. Вона забезпечує стабільну взаємодію з API, дозволяє обробляти формати JSON, XML та інші, а також гарантує безпечну передачу даних між клієнтом і сервером.

Візуалізація маршрутів є важливою частиною аналізу ефективності системи. Бібліотека Folium дозволяє створювати інтерактивні карти з маркерами, маршрутами та зонами покриття. У поєднанні з Matplotlib, яка дозволяє будувати числові графіки продуктивності, та NetworkX, що моделює транспортні або логістичні мережі у вигляді графів, система отримує потужний аналітичний модуль для виявлення вузьких місць та оптимізації топології маршрутів.

Керування чергами завдань у динамічному середовищі, де постійно змінюється пріоритет або статус завдань, вирішується за допомогою модуля queue. Він дозволяє формувати пріоритетні черги, визначати порядок обробки заявок і синхронізувати дії між різними потоками виконання. У комплексі з локальною базою даних SQLite3, яка не потребує налаштування серверної частини, система отримує можливість швидкого зберігання та доступу до інформації без затримок.

У практиці застосування Python для автоматизації розподілу волонтерів важливе місце посідають прикладні веб-рішення на основі Django, які дозволяють впроваджувати алгоритми планування в реальних умовах.

Як приклад розглянемо два підходи до розподілу волонтерів.

1. Automated Volunteer Scheduler for Sporting Events Using First Come First Serve Approach реалізує автоматичне призначення волонтерів за принципом «перший прийшов – перший отримав». Користувачі реєструються на завдання, і система автоматично закріплює їх за змінами у порядку надходження запитів. Це мінімізує втручання організаторів і дозволяє швидко формувати команди при обмежених ресурсах.

2. A Round-Robin Algorithm Approach впроваджує розподіл волонтерів на основі алгоритму Round-Robin. Такий підхід циклічно розподіляє учасників між завданнями, знижуючи ризик перенавантаження одних і тих самих осіб. Це особливо корисно при плануванні подій із великою кількістю змін або паралельних обов'язків, де критично важливий баланс навантаження.

Незалежно від конкретного алгоритму, обидві системи мають спільні риси: повну автоматизацію розподілу, використання Python для реалізації логіки та гнучкість щодо адаптації до різних типів заходів. Це доводить, що Python є ефективним інструментом не лише для побудови алгоритмів, а й для їхньої інтеграції у веб-сервіси, які працюють у реальному часі з живими даними

Таким чином, Python виступає не просто мовою програмування, а платформою для побудови складних, масштабованих і адаптивних рішень у сфері розподілу та маршрутизації волонтерських ресурсів. Його гнучкість дозволяє поєднувати класичні алгоритми з сучасними методами аналізу даних, машинного навчання та графової аналітики.

Висновок до розділу 3

Використання Python у реалізації алгоритмів розподілу та маршрутизації волонтерів забезпечує ефективність, масштабованість і гнучкість розроблюваних систем. Завдяки широкому спектру бібліотек – від обчислювальних (NumPy) до візуалізаційних (Folium, Matplotlib), від аналітичних (scikit-learn, NetworkX) до інфраструктурних (Requests, queue, SQLite3) – Python дозволяє створювати повноцінні рішення, здатні адаптуватися до різноманітних умов і сценаріїв використання.

РОЗДІЛ 4
ПОРІВНЯЛЬНИЙ АНАЛІЗ АЛГОРИТМІВ ПОШУКУ СУСІДІВ У ЗАДАНОМУ РАДІУСІ

Задача пошуку найближчих сусідів полягає у знаходженні в заданому наборі даних точок, які є найближчими до певної точки запиту. Ця фундаментальна проблема є ключовою у багатьох сферах комп'ютерних наук та обчислювальної геометрії, знаходячи застосування в різноманітних областях, таких як машинне навчання, комп'ютерний зір, геоінформаційні системи (ГІС) та інтелектуальний аналіз даних. Особливо важливим є ефективний пошук найближчих сусідів у задачах, що включають багатовимірні ключі пошуку, такі як пошук діапазону та власне пошук найближчих сусідів [29].

Пошук сусідів у заданому радіусі, також відомий як пошук сусідів з фіксованим радіусом або запит за радіусом [30], є специфічною задачею пошуку найближчих сусідів, яка полягає у знаходженні всіх точок даних, що знаходяться в межах певної відстані (радіуса) від заданої точки запиту. На відміну від пошуку k найближчих сусідів, де метою є знаходження фіксованої кількості найближчих точок, пошук у заданому радіусі корисний у сценаріях, де важливим є визначення всіх об'єктів, що знаходяться на певній відстані, наприклад, знаходження всіх магазинів у радіусі 5 км від поточного місцезнаходження користувача або ідентифікація всіх точок даних у межах певної відстані для оцінки локальної щільності або виявлення аномалій.

Проведемо порівняльний аналіз чотирьох алгоритмів, які використовуються для пошуку сусідів у заданому радіусі: K-D Tree, Ball Tree, Brute-force та QuadTree. Кожен з цих алгоритмів має свій основний принцип:

– двійкове дерево (K-D Tree) розділяє простір за допомогою гіперплощин, паралельних осям координат [31],

– ієрархічна структура даних (Ball Tree) розділяє простір за допомогою вкладених гіперсфер (куль) [32],

– прямий метод (Brute-force) обчислює відстань від точки запиту до кожної точки в наборі даних [33],

– деревоподібна структура даних (QuadTree) рекурсивно поділяє двовимірний простір на чотири квадранти [34].

K-D Tree.

K-D Tree є просторовою структурою даних, що представляє собою двійкове дерево, спеціально розроблене для організації точок у k-вимірному просторі, полегшуючи такі операції, як пошук найближчих сусідів та діапазонний пошук. У випадку двовимірного простору (k = 2), кожен вузол дерева представляє точку, а розділення простору відбувається за допомогою вертикальних або горизонтальних ліній, що проходять через ці точки. Листкові вузли дерева містять самі точки даних. Основна ідея K-D дерева полягає у рекурсивному поділі простору на дві півплощини на кожному рівні дерева, використовуючи для цього одну з координат.

Для пошуку найближчих точок у межах заданого радіуса методом K-D Tree, починають з кореневого вузла. Для кожного вузла визначається, в яку підгілку (ліву чи праву) слід перейти, порівнюючи координату точки запиту з розділяючою координатою поточного вузла. Рекурсивний пошук продовжується у вибраній підгілці. Коли досягається листковий вузол, обчислюється відстань між точкою запиту та точкою, що зберігається у цьому вузлі. Якщо ця відстань не перевищує заданий радіус, точка додається до списку результатів. Важливим аспектом алгоритму є те, що під час повернення з рекурсивних викликів перевіряється, чи може інша підгілка також містити точки в межах радіуса. Це відбувається, якщо відстань між точкою запиту та розділяючою гіперплощиною поточного вузла менша за радіус пошуку. У цьому випадку, інша підгілка також рекурсивно досліджується. Цей крок дозволяє уникнути пропуску потенційних сусідів, які можуть знаходитися в іншій гілці дерева, але все ще в межах заданого радіуса від точки запиту. Ефективність пошуку K-D Tree значною мірою

залежить від здатності алгоритму швидко виключати з розгляду цілі піддерева, які не можуть містити релевантних точок.

Для реалізації K-D Tree застосуємо існуючу функцію KDTree() бібліотеки sklearn.

За викликом функції tree = KDTree(points) створюється K-D дерево з масиву points (це набір точок у просторі). Це дерево зберігає просторову структуру, щоб можна було швидко шукати сусідів.

Далі запит indices = tree.query_radius([query_point], r=radius)[0] виводить всі точки з points, які знаходяться в межах відстані radius від query_point».

Ball Tree.

Ball Tree – це двійкове дерево, де кожен вузол визначає гіперсферу (у 2D коло), що містить підмножину точок. Побудова Ball Tree також є рекурсивним процесом. Замість розбиття простору на осі, вона групує точки в «кулі» (сфери), щоб ефективно відкидати ті, що точно далеко.

На кожному кроці набір точок розділяється на дві підмножини, і для кожної підмножини визначається найменше коло, що її охоплює. Для пошуку найближчих сусідів у межах радіуса за допомогою Ball Tree починають з кореневого вузла. Обчислюється відстань між точкою запиту та центром кола поточного вузла. Якщо коло повністю знаходиться за межами кола пошуку, підгілка не досліджується. Інакше, рекурсивно досліджуються дочірні вузли. У листкових вузлах перевіряється відстань до кожної точки. Ball Trees часто є ефективнішими для пошуку найближчих сусідів у просторах високої розмірності, а також добре підходять для запитів з круглим радіусом.

Для пошуку застосуємо існуючу функцію BallTree() бібліотеки sklearn. Створюємо Ball Tree з масиву points ball_tree = BallTree(points). Дерево організовує ці точки у вигляді вкладених «куль», щоб потім можна було швидко шукати сусідів.

Запит indices = ball_tree.query_radius([query_point], r=radius)[0] повертає список індексів знайдених точок.

QuadTree.

QuadTree є деревоподібною структурою даних, яка розділяє двовимірний простір на чотири рівні квадранти. Кожен внутрішній вузол QuadTree має чотири дочірні вузли, що відповідають цим квадрантам. Побудова QuadTree починається з кореневого вузла, що охоплює всю досліджувану область. Якщо квадрант містить більше ніж задану кількість точок або досягнуто певну глибину дерева, він рекурсивно поділяється на чотири менші квадранти. Для пошуку точок у межах заданого радіуса навколо точки запиту в QuadTree, починають з кореневого вузла. Перевіряється, чи перетинається коло пошуку з областю, представленою поточним вузлом. Якщо так, то рекурсивно досліджуються дочірні вузли. Якщо поточний вузол є листковим, відстань між точкою запиту та кожною точкою у цьому листі перевіряється. QuadTrees особливо ефективні для даних з нерівномірним розподілом, оскільки вони адаптивно поділяють простір, забезпечуючи більшу деталізацію в густонаселених областях.

Програмна реалізація QuadTree наступна. Створюється об’єкт класу QuadTree, який охоплює певну прямокутну ділянку – це змінна boundary. Вона задається як список з чотирьох чисел: мінімум і максимум по x, і мінімум і максимум по y. У вузлі також визначається capacity – максимальна кількість точок, які він може зберігати локально, до того як буде розділений. Усі вхідні points одразу вставляються через виклик функції insert.

Функція insert(pt) перевіряє, чи координати точки pt потрапляють у межі boundary. Якщо так і кількість точок ще менше, ніж capacity, точка додається до списку self.points. Якщо ж ліміт перевищено, і вузол ще не був розділений (self.divided – False), викликається функція subdivide.

Функція subdivide() обчислює центр меж (mx, my) і створює чотири нові об’єкти QuadTree, кожен з межами відповідного підпрямокутника. Вони зберігаються в списку self.children. Усі точки, що вже були в вузлі (self.points), перерозподіляються в ці дочірні вузли через виклик insert для кожного з них.

Після цього список точок очищується, а прапорець self.divided встановлюється в True.

Функція query_circle(center, radius) призначена для пошуку точок, що потрапляють у коло з центром center та радіусом radius. Вона спочатку перевіряє, чи взагалі це коло перетинається з межами вузла (boundary). Якщо ні – повертається порожній список. Якщо так – і вузол вже розділений (self.divided == True), запит рекурсивно передається всім дочірнім вузлам. Якщо вузол не поділений, функція перебирає всі точки в self.points і обчислює відстань до center (за допомогою distance.euclidean). Якщо точка потрапляє в коло, вона додається до результату.

Після побудови дерева через qt = QuadTree(points, boundary) можна викликати query_circle(query_point, radius) для пошуку сусідів, і потім зіставити знайдені точки з індексами у вихідному масиві points.

Brute-force.

Brute-force search – це найпростіший підхід, який полягає у послідовному переборі всіх точок у наборі даних та перевірці відстані між кожною точкою та точкою запиту. Для кожної точки p у наборі даних S обчислюється евклідова відстань до точки запиту q. Якщо ця відстань менша або дорівнює заданому радіусу r, точка p додається до результату.

Для реалізації Brute-force search використовується метод повного перебору для пошуку точок у межах заданого радіуса radius від контрольної точки query_point.

Спочатку змінна dists зберігає відстані від кожної точки з масиву points до query_point. Це обчислюється за допомогою функції np.linalg.norm, яка знаходить евклідову норму векторної різниці між усіма точками та query_point. В результаті утворюється масив відстаней, кожен елемент якого відповідає певній точці.

Після цього за допомогою np.where(dists <= radius)[0] отримуються індекси точок, що знаходяться на відстані не більшій за radius. Ці індекси зберігаються у змінній indices.

Насамкінець, викликається функція plot_result, яка візуалізує ці точки на графіку.

Результати пошуку найближчих точок у межах заданого радіуса алгоритмами K-D Tree, Ball Tree, QuadTree та Brute-force подібні і представлені в Додатку А.

Повний скрипт реалізації алгоритмів пошуку сусідів у заданому радіусі методами K-D Tree, Ball Tree, QuadTree та Brute-force приведений в Додатку Б.

Висновок до розділу 4

Приведено алгоритм та програмна реалізація чотирьох алгоритмів пошуку сусідів у заданому радіусі: K-D Tree, Ball Tree, QuadTree та Brute-force.

Для алгоритмів K-D Tree та Ball Tree була використана бібліотека sklearn, що дозволяє швидко організувати просторові структури та проводити ефективний пошук у багатовимірних просторах. В обох випадках швидкість пошуку покращується завдяки рекурсивному поділу простору, що дозволяє значно зменшити кількість перевірок. QuadTree реалізований через об’єктно-орієнтований підхід, а Brute-force методом повного перебору.

Порівняльна характеристика показала, що немає універсального алгоритму, оптимального для всіх випадків.

Таким чином, ефективне застосування алгоритмів пошуку сусідів у заданому радіусі вимагає гнучкого підходу до вибору методів, з урахуванням контексту задачі. Для подальшої реалізації доцільним є використання гібридної стратегії: поєднання дерева-подібних структур для пришвидшення пошуку із застосуванням Brute-force як контрольного механізму для перевірки коректності результатів.

РОЗДІЛ 5
АЛГОРИТМ ПОШУКУ ВОЛОНТЕРІВ У ЗАДАНІЙ ОБЛАСТІ ТА ЙОГО ПРОГРАМНА РЕАЛІЗАЦІЯ

Визначення найближчих волонтерів у конкретній географічній області є важливою задачею в рамках оперативного управління під час надзвичайних ситуацій, зокрема катастроф NaTech. Такі алгоритми широко застосовуються в службах екстреної допомоги, системах медичної координації та різних громадських ініціативах, де час є критичним ресурсом. Ефективне вирішення цієї задачі дозволяє забезпечити швидке та точне направлення волонтерів до місць, що потребують допомоги. Зважаючи на велику кількість волонтерів та необхідність у високій оперативності, розробка та програмна реалізація таких алгоритмів є ключовим елементом в управлінні ресурсами в умовах кризових ситуацій.

Зважаючи на важливість оперативного реагування та точності пошуку волонтерів, розробка програми для автоматизованого пошуку є наступним етапом у реалізації цієї задачі. Програма використовує кілька ключових компонентів, які забезпечують її ефективність. Зокрема, це перетворення координат, обчислення відстаней, а також різні алгоритми пошуку найближчих сусідів. Система також інтегрується з базою даних для отримання актуальних даних про волонтерів і візуалізує результати на карті.

Розглянемо програмну реалізацію, яка включає різноманітні алгоритми, такі як повний перебір, K-D Tree, Ball Tree та QuadTree, що дозволяють порівняти їхню продуктивність та придатність для даної задачі.

5.1 Огляд основних компонентів програми

Програма складається з набору функцій та класів, які спільно забезпечують функціональність пошуку найближчих волонтерів. Загальний потік виконання

програми включає етапи генерації або завантаження даних про волонтерів, виконання пошуку з використанням одного з реалізованих алгоритмів та візуалізацію знайдених результатів на карті.

Ключовими алгоритмами, реалізованими в програмі, є:

- алгоритм повного перебору (brute_force_search),
- алгоритм пошуку з використанням K-D Tree (kd_tree_search),
- алгоритм пошуку з використанням Ball Tree (ball_tree_search),
- алгоритм пошуку з використанням QuadTree (quadtree_search).

Для обчислення відстаней між географічними координатами використовуються дві різні метрики: геодезична відстань та відстань Haversine. Геодезична відстань, що враховує кривизну Землі, забезпечує більш точні результати, особливо на великих відстанях. Відстань Haversine є спрощеною формулою для розрахунку відстані на сфері, яка може бути менш обчислювально затратною.

Програма також інтегрована з базою даних SQLite для зберігання інформації про волонтерів та бібліотекою Folium для візуалізації результатів пошуку на інтерактивній карті. Це дозволяє не лише знаходити найближчих волонтерів, але й отримувати доступ до їхніх даних та наочно відображати їхнє розташування.

5.2 Опис функцій перетворення координат та радіуса

5.2.1 Функція abstract_radius_to_meters

Призначенням функції abstract_radius_to_meters є перетворення абстрактного значення радіуса в реальну відстань у метрах. Скрипт використовує початкову абстрактну систему координат для генерації даних, і для виконання розрахунків відстаней, що мають сенс у реальному світі (наприклад, для визначення радіуса пошуку), ця абстрактна одиниця потребує переведення в стандартну одиницю вимірювання відстані, таку як метр.

Функція приймає одне вхідне значення – абстрактний радіус. На виході функція повертає значення радіуса, виражене в метрах.

Формула перетворення, що використовується в цій функції, включає коефіцієнт масштабування, пов’язаний з абстрактною системою координат та географічною областю інтересу (наприклад, місто Луцьк). Необхідність цієї функції виникає через те, що абстрактні координати, є довільними одиницями, які потрібно прив’язати до реального масштабу. Оскільки кінцевий радіус пошуку має бути в метрах для практичного використання (наприклад, «знайти лікарів у радіусі 1 км»), це перетворення є важливим першим кроком. Коефіцієнт масштабування, залежить від щільності або діапазону згенерованих абстрактних точок відносно фактичної географічної області.

5.2.2 Функції transform_points_to_coords та transform_single_point_to_coords

Функції transform_points_to_coords та transform_single_point_to_coords відповідають за перетворення абстрактних координат (значень x, y в абстрактному просторі) в географічні координати (широту та довготу). Це є важливим етапом, оскільки дозволяє використовувати реальні методи розрахунку відстаней та відображати результати на карті.

Перетворення координат ґрунтується на заданій центральній точці (місті Луцьк) та коефіцієнті масштабування. Спрощене представлення координат в абстрактному просторі дозволяє ефективно зіставляти їх з фактичними широтою та довготою, використовуючи обрану точку початку координат та масштаб. Це підхід дозволяє уникнути складнощів, пов’язаних із безпосередньою роботою з географічними координатами на початкових етапах генерації даних та внутрішніх обчислень.

Функція transform_points_to_coords приймає на вхід список абстрактних координат та повертає список відповідних географічних координат (широти та довготи). Функція transform_single_point_to_coords виконує аналогічне перетворення для однієї абстрактної точки.

Логіка перетворення, що лежить в основі цих функцій, передбачає зсув абстрактних координат відносно центру та їх масштабування відповідно до заданого коефіцієнта. Точність отриманих географічних координат безпосередньо залежить від правильності визначення центральної точки та коефіцієнта масштабування.

5.3 Функції розрахунку відстаней

5.3.1 Функція geodesic_distance

Функція geodesic_distance використовується для обчислення геодезичної відстані між двома точками на поверхні Землі, враховуючи кривизну планети. Це забезпечує точніший розрахунок на великих відстанях у порівнянні з іншими методами, оскільки враховується реальна форма Землі. Геодезична відстань є найкоротшою відстанню між двома точками на поверхні сфери або еліпсоїда, що забезпечує найвищу точність для географічних розрахунків.

Функція приймає два набори координат у форматі широти та довготи для кожної точки (наприклад, point1 і point2), і використовує бібліотеку geopy, яка інтегрує методи для точних географічних обчислень. В результаті отримуємо відстань між точками у метрах.

5.3.2 Функція haversine_distance

Функція haversine_distance використовується для обчислення відстані між двома точками на сфері методом Haversine. Це формула для обчислення відстані по великому колу між двома точками на сфері, що дозволяє враховувати кривизну Землі.

Відстань між двома точками на поверхні сфери можна розрахувати за формулою 5.1:

$$a = sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1) \cdot \cos(\varphi_2) \cdot sin^2\left(\frac{\Delta\lambda}{2}\right), \quad (5.1)$$

де φ_1, φ_2 – широти двох точок в радіанах,

λ_1, λ_2 – довготи двох точок в радіанах,

$\Delta\varphi = \varphi_2 - \varphi_1$ – різниця між широтами двох точок,

$\Delta\lambda = \lambda_2 - \lambda_1$ – різниця між довготами двох точок.

Відстань d між точками можна обчислити за за формулою 5.2:

$$d = 2 \cdot R \cdot atan2(\sqrt{a}, \sqrt{1-a}), \quad (5.2)$$

де R – радіус Землі (приблизно 6371000 м),

atan2 – двоступінчата арктангенс функція, яка дозволяє коректно враховувати напрямок для всіх чотирьох чвертей координатної системи.

Формула є дещо спрощеною, порівняно з геодезичними розрахунками, але є обчислювально менш затратною, що робить її корисною для швидких пошуків.

У лістингу 5.1 описано розрахунок відстані між точками, де x, y: кортежі з координатами двох точок (широта, довгота) в градусах.

Лістинг 5.1 – Розрахунок відстані між точками

```
def haversine_distance(x, y):
    lat1, lon1 = x
    lat2, lon2 = y
    dlat, dlon = lat2 - lat1, lon2 - lon1
    a = sin(dlat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dlon / 2) ** 2
    return 6371000 * 2 * atan2(sqrt(a), sqrt(1 - a))
```

кінець лістингу 5.1

В результаті отримуємо відстань між точками у метрах.

Ця функція є альтернативою геодезичній відстані, яку можна використовувати для швидших, але менш точних розрахунків, особливо на меншій відстані або при великих обсягах даних.

Геодезична відстань (geodesic_distance) є більш точною, особливо для великих відстаней, де вплив кривизни Землі є значущим.

Відстань за формулою Haversine (haversine_distance) є менш точною, але швидшою у виконанні, оскільки не враховує деталі еліпсоїдної форми Землі.

Використовуємо обидві функції для досягнення балансу між точністю та швидкістю залежно від вибраного методу пошуку точок у заданому радіусі.

Функція geodesic_distance застосовується в методах, таких як brute-force та quadtree_search, де точність відіграє важливу роль, а обчислювальні ресурси не є критичними.

Функція haversine_distance використовується у ball_tree_search, де швидкість пошуку є пріоритетною, а точність може бути менш важливою в контексті відстаней у межах кількох десятків або сотень кілометрів.

Цей компроміс дозволяє адаптувати програму для різних умов, залежно від вимог до швидкості та точності.

5.4 Алгоритми пошуку найближчих сусідів

5.4.1 Функція brute_force_search

Функція brute_force_search реалізує алгоритм повного перебору для пошуку сусідів у заданому радіусі. Він полягає в послідовному переборі всіх наявних точок, обчисленні відстані між кожною точкою та точкою запиту, та відборі тих точок, відстань до яких не перевищує заданий радіус. Для обчислення відстані між точками використовується функція geodesic_distance, що забезпечує точність вимірювань.

Алгоритм приймає на вхід список усіх точок, координати точки запиту та радіус пошуку (в метрах).

На виході функція повертає список індексів (або самі точки), які знаходяться в межах заданого радіуса від точки запиту.

5.4.2 Функція kd_tree_search

Функція kd_tree_search реалізує алгоритм пошуку найближчих сусідів за допомогою KD-дерева.

Процес пошуку з використанням KD-дерева включає кілька етапів. Спочатку необхідно перетворити географічні координати в метричну систему координат, яка є більш підходящою для роботи з KD-деревами, оскільки вони найкраще працюють з евклідовими відстанями в декартовій системі координат.

Далі будується KD-дерево на основі перетворених координат з використанням класу sklearn.neighbors.KDTree з бібліотеки scikit-learn, яка надає оптимізовану реалізацію KD-дерев. Після побудови дерева використовується метод query_radius для знаходження всіх сусідів, що знаходяться в межах заданого радіуса від точки запиту.

Функція приймає на вхід список усіх точок (у географічних координатах), координати точки запиту (у географічних координатах) та радіус пошуку (в метрах). На виході функція повертає список індексів точок, що знаходяться в межах заданого радіуса.

5.4.3 Функція ball_tree_search

Функція ball_tree_search реалізує алгоритм пошуку найближчих сусідів за допомогою Ball-дерева.

Процес пошуку з використанням Ball-дерева починається з перетворення географічних координат (градусів) у радіани, оскільки функція haversine_distance оперує з радіанами. Далі будується Ball-дерево з використанням координат у радіанах та функції haversine_distance як метрики. Після побудови дерева використовується метод query_radius для знаходження сусідів у заданому радіусі.

Функція приймає на вхід список усіх точок (у географічних координатах), координати точки запиту (у географічних координатах) та радіус пошуку (в

метрах). На виході функція повертає список індексів точок, що знаходяться в межах заданого радіуса. Використання відстані Haversine з Ball-деревом дозволяє безпосередньо працювати зі сферичними відстанями, що забезпечує кращу точність.

5.4.4 Клас Quadtree та функція quadtree_search

Клас Quadtree реалізує структуру даних Quadtree, яка є ієрархічною структурою, що використовується для ефективного зберігання та пошуку даних у двовимірному просторі.

Конструктор класу (__init__) Quadtree приймає параметри boundary (об’єкт, що визначає межі квадранта), capacity (максимальна кількість точок, яку може містити вузол, перш ніж буде поділений). Він ініціалізує порожній список points для зберігання точок у поточному вузлі, булеву змінну divided зі значенням False, яка вказує, чи був вже поділений поточний вузол, та порожній список children для зберігання дочірніх квадрантів.

Метод insert використовується для вставки точки в Quadtree. Логіка вставки включає перевірку, чи знаходиться точка в межах поточного квадранта. Якщо кількість точок у поточному вузлі не досягла ліміту capacity, точка додається до списку points цього вузла. Якщо ліміт досягнуто і вузол ще не був поділений, викликається метод subdivide для поділу поточного квадранта на чотири дочірні квадранти. Після поділу точка рекурсивно вставляється в один з дочірніх квадрантів, до якого вона належить.

Метод subdivide відповідає за поділ поточного квадранта на чотири рівні дочірні квадранти: північно-західний, північно-східний, південно-західний та південно-східний. Для кожного дочірнього квадранта створюється новий об’єкт Quadtree з відповідними межами. Потім точки з поточного вузла розподіляються між дочірніми вузлами.

Метод query використовується для пошуку точок у заданому колі (визначеному центром та радіусом). Для кожної точки в поточному вузлі перевіряється відстань до центру кола. Якщо точка знаходиться в межах радіуса,

вона додається до результатів пошуку. Потім рекурсивно викликається метод query для дочірніх вузлів, якщо коло пошуку перетинається з межами цих вузлів. Ефективність Quadtree полягає в здатності швидко відкидати великі області, які не містять кола запиту. Рекурсивно переходячи лише до відповідних дочірніх квадрантів, алгоритм уникає перевірки кожної окремої точки. Перевірка перетину між колом запиту та межами квадранта є ключовою для цієї ефективності.

Функція quadtree_search використовує клас Quadtree для пошуку сусідів. Спочатку визначаються межі для кореневого вузла Quadtree на основі координат усіх точок. Потім створюється об’єкт Quadtree з визначеними межами та заданою ємністю вузла. Усі точки вставляються в Quadtree за допомогою методу insert. Нарешті, викликається метод query з координатами точки запиту та радіусом пошуку для знаходження сусідів.

5.5 Отримання даних про волонтерів та візуалізація

5.5.1 Функція get_volunteer_data

Функція get_volunteer_data призначена для отримання детальної інформації про волонтерів з бази даних SQLite на основі індексів найближчих сусідів, знайдених алгоритмами пошуку. Ця функція забезпечує зв’язок між результатами просторового пошуку та фактичними даними про волонтерів.

Процес отримання даних включає підключення до бази даних SQLite за допомогою бібліотеки sqlite3. Потім формується SQL-запит, який вибирає записи з таблиці волонтерів, де навичка відповідає, наприклад, «Медсестра», а ім’я (або інший ідентифікатор) волонтера відповідає одному зі знайдених індексів (або міток). Після виконання SQL-запиту функція повертає список знайдених волонтерів разом з їхніми координатами.

Функція приймає на вхід список індексів (або міток) знайдених сусідів. На виході функція повертає список даних про волонтерів, включаючи їхні імена та координати.

5.5.2 Функція plot_on_map

Функція plot_on_map відіграє значну роль у візуалізації результатів пошуку на інтерактивній карті за допомогою бібліотеки Folium.

Процес візуалізації включає створення об’єкта карти Folium, центрованої на місті Луцьк. На карту додаються маркери для всіх точок даних. Окремими кольорами можуть бути позначені волонтери-медсестри та знайдені найближчі сусіди. Також на карту додається маркер для точки запиту, щоб візуально позначити місце, від якого здійснювався пошук. Для наочного відображення радіуса пошуку на карту додається коло з центром у точці запиту та радіусом, що відповідає заданому значенню. Згенерована інтерактивна карта зберігається у вигляді HTML-файлу, який можна відкрити у веб-браузері для перегляду та взаємодії.

Функція приймає на вхід список усіх точок, список знайдених сусідів, координати точки запиту та радіус пошуку. На виході функція створює та зберігає HTML-файл, що містить інтерактивну карту з візуалізованими результатами.

5.5.3 Основна частина програми

Основна частина програми координує виконання всіх описаних вище функцій для пошуку найближчих волонтерів. Послідовність дій в основній частині скрипту виглядає наступним чином.

1. Спочатку відбувається генерація випадкових абстрактних точок, які представляють потенційних волонтерів. Також визначаються координати точки запиту (місцезнаходження, для якого потрібно знайти найближчих медсестер) та значення радіуса пошуку. Генерування випадкових точок дозволяє протестувати алгоритми без необхідності використання попередньо існуючого набору даних.

2. Абстрактні координати всіх згенерованих точок, а також координати точки запиту, перетворюються в географічні координати (широту та довготу) за допомогою функцій transform_points_to_coords та transform_single_point_to_coords. Це необхідно для використання реальних методів розрахунку відстаней та відображення на карті.

3. Абстрактне значення радіуса пошуку перетворюється в реальну відстань у метрах за допомогою функції abstract_radius_to_meters. Це забезпечує практичну інтерпретацію радіуса пошуку.

4. Для кожної згенерованої точки створюється мітка (label), яка може використовуватися як ідентифікатор волонтера та для зв’язку з даними в базі даних.

5. Далі послідовно викликається кожна з функцій пошуку найближчих сусідів (brute_force_search, kd_tree_search, ball_tree_search, quadtree_search). Кожна функція отримує на вхід список географічних координат усіх точок, координати точки запиту та радіус пошуку в метрах. Результатом виконання кожної функції є список індексів точок, які знаходяться в межах заданого радіуса. Це дозволяє безпосередньо порівняти результати роботи різних алгоритмів на одному й тому ж наборі даних та запиті.

6. Для кожного набору індексів, отриманих від різних алгоритмів пошуку, викликається функція get_volunteer_data. Ця функція використовує індекси для запиту до бази даних SQLite та отримання детальної інформації про волонтерів, які відповідають знайденим точкам.

7. На завершення для результатів, отриманих кожним алгоритмом пошуку, викликається функція plot_on_map. Ця функція візуалізує знайдені найближчі волонтери (медсестри), вихідні дані, точку запиту та радіус пошуку на окремій інтерактивній карті, яка зберігається у вигляді HTML-файлу.

Після завершення обчислень кожен алгоритм повертає перелік індексів точок, що потрапили в зону пошуку. Ці індекси використовуються для отримання детальної інформації про відповідних волонтерів із бази даних. Далі

результати кожного методу виводяться у вигляді списків знайдених волонтерів, які супроводжуються географічною візуалізацією на інтерактивній карті. Результати пошуку волонтерів-медсестер відображені на рисунку 5.1.

```
 Волонтери (Медсестри) у межах кола:
Волонтер_6, координати: (50.74894, 25.33999)
Волонтер_15, координати: (50.74323, 25.32649)
Волонтер_94, координати: (50.75200, 25.32718)
Карту збережено як: Brute-force_Lutsk_map.html

 Волонтери (Медсестри) у межах кола:
Волонтер_6, координати: (50.74894, 25.33999)
Волонтер_15, координати: (50.74323, 25.32649)
Волонтер_94, координати: (50.75200, 25.32718)
Карту збережено як: KD-Tree_Lutsk_map.html

 Волонтери (Медсестри) у межах кола:
Волонтер_6, координати: (50.74894, 25.33999)
Волонтер_15, координати: (50.74323, 25.32649)
Волонтер_94, координати: (50.75200, 25.32718)
Карту збережено як: Ball-Tree_Lutsk_map.html

 Волонтери (Медсестри) у межах кола:
Волонтер_6, координати: (50.74894, 25.33999)
Волонтер_15, координати: (50.74323, 25.32649)
Волонтер_94, координати: (50.75200, 25.32718)
Карту збережено як: Quadtree_Lutsk_map.html
```

Рисунок 5.1 – Результати пошуку волонтерів-медсестер у місті Луцьк в заданому радіусі

Таким чином, користувач може не лише побачити імена та контакти волонтерів, а й оцінити просторове охоплення кожного алгоритму та порівняти їхню точність і продуктивність.

Результати просторового пошуку медсестер у Луцьку за допомогою різних алгоритмів подібні і представлені в Додатку В.

Повний скрипт реалізації алгоритмів пошуку сусідів приведений у Додатку Г.

5.6 Результати експериментального дослідження ефективності алгоритмів пошуку

Проведено порівняльний аналіз ефективності чотирьох алгоритмів пошуку найближчих сусідів: Brute-force, KD-Tree, Ball-Tree та Quadtree. Основною метою дослідження було оцінити масштабованість кожного з алгоритмів залежно від обсягу даних, що подається на вхід, у межах діапазону від 0 до 20000 точок з кроком 2500. Вимірювання виконувалися за метрикою часу пошуку (в секундах) для фіксованого запиту.

Результати порівняльного аналізу ефективності чотирьох алгоритмів пошуку найближчих сусідів представлені в Додатку Д.

Результати експерименту представлено у вигляді графіків, що демонструють залежність часу пошуку від кількості точок. Згідно з отриманими даними:

– Brute-force демонструє лінійне зростання часу пошуку. При 20000 точках час пошуку сягає ~ 3,37 с, що свідчить про низьку придатність цього підходу для великих наборів даних;

– KD-Tree виявився найефективнішим алгоритмом серед досліджених, демонструючи практично сталий час пошуку в межах 0,002-0,013 с навіть при максимальній кількості точок. Попри високу ефективність, алгоритм KD-Tree може повертати помилкові результати в окремих випадках – зокрема, коли просторові дані мають нерівномірний розподіл або значну кількість точок поблизу меж радіуса пошуку;

– Ball-Tree також демонструє добру масштабованість, хоча і дещо нижчу порівняно з KD-Tree. Час пошуку зростає повільно, досягаючи ~ 0,70 с на 20000 точках. Така поведінка узгоджується з конструкцією алгоритму, який застосовує ієрархічну кластеризацію у вигляді вкладених гіперсфер;

– QuadTree, незважаючи на свою теоретичну ефективність у 2D-просторі, показав продуктивність, подібну до Brute-force. Час пошуку при 20000 точках

становив ~ 3,86 с, що свідчить про значну чутливість цього алгоритму до геометричного розподілу даних. У загальному випадку, Quadtree не продемонстрував переваг над наївним пошуком.

Висновок до розділу 5

У програмі реалізовано та порівняно кілька алгоритмів пошуку, кожен з яких має свої переваги та недоліки. Алгоритм повного перебору є найпростішим, але неефективним для великих наборів даних. KD-Tree та Ball-Tree пропонують більш ефективні рішення, особливо для великих обсягів інформації, хоча їхня продуктивність може залежати від розподілу даних та обраної метрики відстані. QuadTree є ефективним для просторово кластеризованих даних та запитів діапазону.

Важливим аспектом програми є використання функцій перетворення координат та розрахунку відстаней, які забезпечують точність результатів у географічному контексті. Візуалізація результатів на інтерактивній карті за допомогою бібліотеки Folium є цінним інструментом для розуміння та представлення результатів пошуку.

Результати дослідження однозначно вказують на перевагу KD-Tree як базового методу для ефективного пошуку найближчих сусідів у великих множинах точок. Ball-Tree може слугувати альтернативою в умовах складного або багатовимірного розподілу даних. Натомість Brute-force і QuadTree не забезпечують достатнього рівня масштабованості й можуть розглядатися лише в якості контрольних або допоміжних методів.

РОЗДІЛ 6
АЛГОРИТМИ ПОШУКУ МАРШРУТУ З МІНІМАЛЬНОЮ ДОВЖИНОЮ ДО ЦІЛІ

Пошук маршруту з найкоротшою відстанню до цілі є базовою задачею у сфері логістики, навігації та управління волонтерськими ресурсами. Оптимізація саме за довжиною шляху дозволяє мінімізувати витрати зусиль та часу переміщення. Для вирішення таких задач застосовуються алгоритми на графах, де вузли представляють точки призначення, а ребра – відстані між ними. Серед найбільш поширених методів – Дейкстри, алгоритм A* та алгоритм Флойда-Уоршелла. Ці алгоритми є основою для практичного застосування графових моделей у популярних сервісах маршрутизації, таких як OSRM (Open Source Routing Machine), GraphHopper та OpenRouteService, що дозволяють вирішувати складні логістичні задачі в реальному часі.

6.1 Алгоритм Дейкстри

Алгоритм Дейкстри є одним з найбільш відомих та корисних алгоритмів у галузі інформатики, призначений для знаходження найкоротших шляхів від заданої початкової вершини до всіх інших вершин у зваженому графі [35]. Основний принцип роботи алгоритму полягає в ітеративному дослідженні графа, починаючи з початкової вершини. На кожному кроці алгоритм вибирає невідвідану вершину з найменшою відомою відстанню від початкової вершини та оновлює відстані до її сусідів.

Процес роботи алгоритму Дейкстри включає кілька ключових етапів. Спочатку алгоритм ініціалізує відстані до всіх вершин як нескінченність, за винятком початкової вершини, відстань до якої встановлюється рівною нулю. Підтримується набір відвіданих та невідвіданих вершин. На кожній ітерації алгоритм вибирає з набору невідвіданих вершин ту, яка має найменшу поточну

відстань від початкової вершини. Ця вершина стає поточною, і її позначають як відвідану. Далі алгоритм розглядає всіх невідвіданих сусідів поточної вершини та обчислює відстань до них через поточну вершину. Якщо знайдена відстань є меншою за поточну відому відстань до сусіда, то відстань до сусіда оновлюється. Для ефективного вибору вершини з найменшою відстанню алгоритм часто використовує чергу з пріоритетом [36].

Для застосування алгоритму Дейкстри для пошуку найшвидшого маршруту, необхідно, щоб вага кожного ребра в графі представляла час, необхідний для подолання відповідної ділянки шляху. Вершини графа в такому випадку відповідають точкам на карті, таким як перехрестя, а ребра – дорогам між ними. Алгоритм знаходить шлях з найменшою сумарною вагою ребер, що відповідає мінімальному загальному часу в дорозі. Наприклад, алгоритм може бути використаний для знаходження найшвидшого пішохідного маршруту між двома будівлями в університетському містечку, де ваги ребер визначаються часом.

6.2 Алгоритм A*

Алгоритм A* є розширенням алгоритму Дейкстри, яке використовує евристичну функцію для спрямування пошуку до цільової вершини [37]. Цей алгоритм оцінює кожну вершину за допомогою функції $f(n) = g(n) + h(n)$, де $g(n)$ представляє вартість шляху від початкової вершини до поточної вершини n, а $h(n)$ є евристичною оцінкою вартості шляху від n до цільової вершини. Алгоритм використовує чергу з пріоритетом для вибору вершини з найнижчим значенням $f(n)$ для подальшого дослідження.

Евристична функція $h(n)$ є ключовим фактором, що визначає ефективність алгоритму A*. Якщо $h(n)$ є точною оцінкою вартості шляху до цілі, A* знайде найкоротший шлях дуже швидко, досліджуючи мінімальну кількість вузлів. Якщо $h(n)$ недооцінює фактичну вартість, A* гарантовано знайде найкоротший

шлях, але може дослідити більше вузлів, ніж необхідно. Якщо h(n) переоцінює фактичну вартість, A* може знайти неоптимальний шлях, але працюватиме швидше, оскільки буде менше досліджувати вузли.

Алгоритм A* часто ефективніший за Дейкстру, оскільки використовує евристику, яка допомагає швидше знаходити вузли, що наближають до цілі.

Розробка ефективної евристики може бути складною і залежить від конкретної задачі. Невдало підібрана евристика може не забезпечити очікуваного прискорення або навіть призвести до гіршої продуктивності, ніж у Дейкстри.

Для пошуку найшвидшого маршруту на карті, евристикою часто використовують пряму відстань (наприклад, евклідову або манхеттенську) до цілі, поділену на максимальну можливу швидкість. Це забезпечує допустимість евристики, оскільки фактичний час у дорозі ніколи не буде меншим за час, розрахований за прямою відстанню та максимальною швидкістю.

Існує компроміс між точністю евристики та швидкістю роботи A*. Точніші евристики можуть призвести до повільнішого пошуку, але гарантують оптимальність, тоді як менш точні евристики можуть прискорити пошук, але ризикують знайти неоптимальний шлях. Вибір евристики залежить від вимог до точності та часу виконання для конкретного застосування. У реальних навігаційних системах часто використовують евристики, які є досить точними, щоб значно прискорити пошук, але при цьому гарантують знаходження результату близького до оптимального шляху.

6.3 Алгоритм Флойда-Уоршелла

Алгоритм Флойда-Уоршелла є класичним алгоритмом динамічного програмування, який використовується для знаходження найкоротших шляхів між усіма парами вершин у зваженому графі [38].

Принцип його роботи полягає в ітеративному розгляді всіх можливих проміжних вершин k для кожної пари вершин (i, j). На кожній ітерації алгоритм перевіряє, чи є шлях від вершини i до вершини j через вершину k коротшим за поточний відомий шлях між i та j, і якщо так, то оновлює відстань.

Для зберігання найкоротших відстаней між усіма парами вершин алгоритм використовує матрицю відстаней, де елемент dist[i][j] представляє поточну найкоротшу відстань між вершинами i та j. Початково матриця відстаней ініціалізується прямими вагами ребер між вершинами (або нескінченністю, якщо ребра немає, та нулем для відстані від вершини до самої себе). Алгоритм Флойда-Уоршелла може працювати з графами, що мають ребра з від’ємною вагою, але він не призначений для графів, що містять від’ємні цикли.

У контексті дорожньої мережі, алгоритм Флойда-Уоршелла може бути застосований для обчислення матриці найкоротшого часу в дорозі між усіма можливими парами точок (вершин) у мережі. У цьому випадку вага кожного ребра графа представляє час, необхідний для проїзду між двома сусідніми точками. Після завершення роботи алгоритму, елемент [i][j] у отриманій матриці відстаней буде містити мінімальний час, необхідний для подорожі з точки i до точки j. Така інформація може бути особливо корисною для завдань, де необхідно швидко отримувати час подорожі між будь-якими двома точками в мережі без необхідності повторно запускати алгоритм пошуку шляху для кожної окремої пари.

Алгоритм Флойда-Уоршелла надає повну інформацію про найкоротші шляхи між усіма парами вершин в графі, що може бути корисно для аналізу загальної зв’язності та ефективності мережі, а не лише для пошуку шляху між двома конкретними точками. Знаючи найкоротші відстані між усіма парами вершин, можна легко визначити, наприклад, діаметр графа (найбільша з усіх найкоротших відстаней) або центральні вершини (вершини з найменшою максимальною відстанню до інших вершин).

6.4 Оптимізації та розширення алгоритмів

Для підвищення ефективності алгоритмів пошуку, особливо в контексті великих дорожніх мереж та реальних умов, існує ряд оптимізацій та розширень.

Однією з ключових оптимізацій є використання Contraction Hierarchies (CH). Цей метод попередньої обробки графа додає ярлики (shortcut) для обходу менш важливих вершин під час виконання запиту на маршрут, що значно прискорює пошук найкоротшого шляху [39]. CH особливо ефективні для статичних дорожніх мереж, де топологія змінюється рідко. Іншим підходом є використання Landmark-based A (ALT)*. Ця оптимізація алгоритму A* використовує попередньо обчислені відстані до набору спеціальних «орієнтирів» для покращення евристичної оцінки, що спрямовує пошук більш безпосередньо до цілі [40].

Для алгоритму Дейкстри також існують оптимізації, такі як використання черг з пріоритетом з більш ефективними внутрішніми структурами даних, наприклад, бінарних куп або куп Фібоначчі, що може покращити часову складність алгоритму.

Крім того, при пошуку найкоротшого шляху між двома конкретними вузлами, виконання алгоритму Дейкстри можна зупинити одразу після знаходження шляху до цільової вершини, що дозволяє уникнути непотрібних обчислень.

Оптимізації, такі як Contraction Hierarchies та Landmarks, є важливими для практичного використання алгоритмів пошуку найшвидшого маршруту в великих дорожніх мережах, оскільки вони дозволяють досягти прийнятного часу відповіді для користувачів. Без таких оптимізацій базові алгоритми, такі як Дейкстра та A*, можуть бути занадто повільними для обробки запитів у великих мережах, особливо в реальному часі.

Для підвищення точності та швидкості знаходження маршрутів з мінімальним часом у реальних умовах, важливим є врахування динамічних

факторів, таких як трафік у реальному часі. Це може бути досягнуто шляхом інтеграції даних про трафік у реальному часі як динамічно змінюваних ваг ребер у графі. Такий підхід дозволяє алгоритмам враховувати поточну завантаженість доріг при розрахунку найшвидшого маршруту. Також використовується історичні дані про трафік для прогнозування майбутніх умов дорожнього руху та вибору маршрутів на основі цих прогнозів. Розробка та використання евристичних функцій в алгоритмі A*, які враховують поточні умови трафіку або прогнозовані затримки, також сприяють підвищенню точності.

У реальних системах навігації часто застосовується комбінування різних алгоритмів та підходів для досягнення кращої точності та швидкості. Наприклад, Contraction Hierarchies можуть бути використані для швидкого знаходження попереднього маршруту, який потім коригується на основі даних.

6.5 Програмна реалізація алгоритмів пошуку найкоротшого шляху

Розглянемо програмну реалізацію трьох алгоритми пошуку найкоротшого шляху (Дейкстри, A*, Флойда-Уоршелла) для зваженого графа.

Початково було створено двовимірну сітку розміром 5×5 за допомогою методу networkx.grid_2d_graph, що моделює карту перехресть (вузлів) і доріг (ребер). Для спрощення обробки мітки вузлів були перетворені на цілі числа. Кожному ребру призначалась випадкова вага у межах від 1 до 10, що імітує час переміщення між вузлами, а координати для візуалізації були згенеровані через spring_layout, що забезпечує просторову розкладку графа.

Алгоритм Дейкстри реалізований як класичний метод знаходження найкоротших шляхів від початкового вузла до всіх інших. Вхідні значення відстаней до вузлів ініціалізуються нескінченністю, за винятком стартового вузла, для якого відстань дорівнює нулю. Основна логіка полягає у використанні пріоритетної черги, з якої на кожній ітерації витягується вузол із найменшою відстанню. Далі відбувається оновлення відстаней до його сусідів за формулою

dist[neighbor] = dist[current] + weight, якщо нова відстань менша за попередню. Для кожного оновлення фіксується попередник у словнику, що дозволяє згодом відтворити найкоротший маршрут. Завершується алгоритм після обробки всіх вузлів, а шлях відновлюється ітеративно – від цільового вузла до початкового.

У випадку алгоритму A* основна логіка доповнена евристичною функцією, що оцінює наближеність вузла до цілі. Фактична вартість переміщення від початку зберігається у словнику g_score, а функція оцінки f_score обчислюється як сума g_score + heuristic.

Евристикою виступає евклідова відстань між координатами вузлів, що забезпечує ефективність і допустимість пошуку.

Алгоритм використовує відкриту множину вузлів, з якої на кожному кроці вибирається вузол із найменшим f_score. При кожному переході перевіряється можливість поліпшення g_score для сусідів. У разі успіху оновлюються значення f_score і g_score, а також попередники. Алгоритм завершується, щойно досягається цільовий вузол, після чого шлях відновлюється за аналогією з алгоритмом Дейкстри.

Алгоритм Флойда-Уоршелла має іншу природу і не потребує стартової точки. Його основна логіка полягає в ітеративному оновленні матриці відстаней між усіма парами вузлів через проміжні вершини. Якщо для деякої пари вузлів i та j існує такий вузол k, що шлях через k коротший, ніж безпосередній шлях від i до j, то відстань оновлюється.

Використання nx.floyd_warshall_predecessor_and_distance з бібліотеки NetworkX, дозволило одразу отримати матрицю найкоротших відстаней і попередників. Це дозволяє швидко відновити шлях між будь-якою парою вузлів.

Візуалізація результатів реалізована через окрему функцію, яка відображає граф із урахуванням знайденого шляху. Шлях позначається суцільними ребрами, що наочно виділяє обрану траєкторію. Всі вузли зберігають свої позиції, згенеровані за допомогою spring_layout, що створює зручну для інтерпретації

картину. Для кожного ребра виводиться його вага, що дозволяє користувачеві оцінити кількісні характеристики маршруту.

Результати пошуку маршруту з мінімальною довжиною до цілі за алгоритмами Дейкстри, A*, Флойда-Уоршелла представлені в Додатку Е. Результат ідентичний для всіх трьох алгоритмів.

Повний скрипт пошуку мінімальної віддалі представлено в Додатку Ж.

Для оцінювання ефективності трьох класичних алгоритмів пошуку найкоротшого шляху – Дейкстри, A* та Флойда-Уоршелла – було проведено серію експериментів, у яких кожен алгоритм запускався 100 разів на графі у вигляді сітки розміром 5×55, з випадковими вагами ребер, згенерованими у діапазоні [1;10].

Середній час виконання кожного з алгоритмів було зафіксовано та подано у вигляді стовпчикової діаграми в Додатку К. Результати демонструють суттєві відмінності в обчислювальній складності та продуктивності обраних методів.

Середній час виконання алгоритму Дейкстри склав приблизно 0,0002 с, що є найнижчим показником серед усіх розглянутих. Його продуктивність пояснюється сприятливою структурою графа (рідкісні з’єднання, відсутність негативних ваг) та відсутністю необхідності додаткових обчислень, як у A*.

Хоча A* використовує евристичну функцію для пришвидшення пошуку, в даному випадку вона не забезпечила перевагу в часі через невеликий розмір графа. Середній час виконання склав 0,0003 с, що на ~ 50 % більше, ніж у Дейкстри. Збільшення обумовлено додатковими обчисленнями евристики на кожному кроці, хоча в більших або геометрично складніших графах A* потенційно виграє за рахунок скорочення простору пошуку.

Алгоритм Флойда-Уоршелла виявився найменш ефективним, з середнім часом виконання 0,0030 с, що приблизно у 10-15 разів повільніше, ніж у попередніх. Причина полягає в його глобальному характері – алгоритм розраховує найкоротші шляхи між усіма парами вершин, тоді як Дейкстра і A* – лише між фіксованими start і goal.

Для вивчення масштабованості алгоритмів пошуку найкоротшого шляху було проведено експериментальну оцінку часу виконання трьох алгоритмів – Дейкстри, A* та Флойда-Уоршелла – на графах, розмір яких варіювався від 10 до 100 вузлів – Додаток Л. Кожна точка на графіку відповідає середньому часу виконання за 100 запусків на випадково згенерованих зважених графах сіткового типу.

З графіка чітко видно три окремі траєкторії зростання. Дейкстра та A* демонструють майже лінійну або близьку до лінійної залежність, а алгоритм Флойд-Уоршелл демонструє експоненціальне зростання часу виконання.

Поведінка алгоритму Дейкстри стабільна. Час виконання зростає повільно навіть при збільшенні розміру графа до 100 вузлів.

Результати роботи A* майже ідентичні до Дейкстри, з невеликим додатковим часом на обчислення евристики.

Найбільш обчислювально затратний час показав алгоритм Флойда-Уоршелла. Навіть при 40-50 вузлах час виконання суттєво перевищує конкурентів. На 100 вузлах середній час виконання наближається до 0,15 с, що в десятки разів більше, ніж у Дейкстри та A*.

Висновок до розділу 6

Алгоритм Дейкстри показав найвищу ефективність з середнім часом виконання 0,0002 с на графі 5×5, що пояснюється його прямолінійним підходом і відсутністю складних обчислень. Це найшвидший алгоритм для розв’язку задачі пошуку найкоротшого шляху між фіксованими точками.

Алгоритм A*, який додає евристичну функцію для пришвидшення пошуку, працює трохи повільніше, з середнім часом 0,0003 с. При великих або складніших графах A* може забезпечити значне прискорення порівняно з Дейкстрою, але на малих графах виграш мінімальний через невеликий наклад на обчислення евристики.

Алгоритм Флойда-Уоршелла виявився найменш ефективним, з середнім часом 0,0030 с, оскільки його глобальна природа вимагає обчислення найкоротших шляхів для всіх пар точок. Це значно уповільнює його роботу порівняно з іншими алгоритмами.

Експерименти з масштабованістю показали, що час виконання алгоритмів Дейкстри та A* зростає лінійно, тоді як Флойда-Уоршелла – експоненційно. На графах зі 100 вузлами час виконання алгоритму Флойда-Уоршелла досягав 0,15 с, що в десятки разів більше, ніж у Дейкстри та A*.

Таким чином, Дейкстра та A* підходять для швидкого пошуку в малих і середніх графах, тоді як Флойд-Уоршелл зручний для знаходження всіх найкоротших шляхів у великих графах, але не підходить для задач з конкретною парою точок через високу обчислювальну складність.

РОЗДІЛ 7
СЕРВІСИ OSRM, GRAPHHOPPER ТА OPENROUTESERVICE ПОШУКУ МАРШРУТУ ВОЛОНТЕРІВ ДО ЦІЛІ

OpenStreetMap (OSM) – це відкритий проєкт зі створення детальної та вільної для використання карти світу. Дані в OSM збираються та оновлюються спільнотою волонтерів, які використовують GPS-пристрої, супутникові знімки, а також місцеві знання. Цей підхід дозволяє отримувати актуальні та точні географічні дані, особливо корисні в регіонах, де комерційні карти застарілі або відсутні. OSM активно використовується в урбаністиці, логістиці, гуманітарних місіях, навігаційних додатках та аналітиці просторових даних.

OpenStreetMap (OSM) став основою для численних сучасних систем побудови маршрутів, зокрема OSRM, GraphHopper та OpenRouteService – рішень, що активно застосовуються у волонтерських, гуманітарних та логістичних задачах. Завдяки відкритим даним, гнучкості та широкому охопленню, ці сервіси дозволяють здійснювати точний та оперативний пошук маршрутів навіть у складних або слабо картографованих регіонах.

Суть задачі полягає у знаходженні оптимального шляху у великомасштабному графі дорожньої мережі, який може містити мільйони вузлів і ребер. Класичні підходи на кшталт алгоритму Дейкстри або навіть A*, хоч і забезпечують коректні результати, виявляються надто повільними для практичного застосування в режимі реального часу, особливо на континентальних відстанях. У відповідь на це були розроблені високопродуктивні спеціалізовані алгоритми та індексаційні структури, які лежать в основі згаданих сервісів.

7.1 Open Source Routing Machine (OSRM)

Open Source Routing Machine (OSRM) – це високопродуктивна система побудови маршрутів, що спеціалізується на забезпеченні максимально швидкої обробки запитів [41, 42].

Основою її алгоритмічної архітектури є метод Contraction Hierarchies (CH), який використовується майже ексклюзивно. Даний підхід забезпечує ефективне скорочення графа дорожньої мережі шляхом поступового усунення менш важливих вузлів із збереженням коротких шляхів через додавання стислих сполучень. Вся програмна архітектура OSRM оптимізована під CH, що дозволяє досягати відповідей на запити в межах мілісекунд навіть на масштабних графах.

Реалізація OSRM виконана на мові C++, що забезпечує низькорівневе управління пам’яттю та високу продуктивність. Окрім CH, система може також підтримувати альтернативний ієрархічний метод – Multi-Level Dijkstra (MLD). У цьому підході граф ієрархічно ділиться на рівні (локальні, регіональні, національні), причому пошук виконується насамперед на верхніх рівнях, переходячи до нижчих лише поблизу початкової та кінцевої точок маршруту. Хоча MLD забезпечує гнучкість, саме CH лишається основним інструментом для досягнення максимальної швидкодії.

Особливе місце займає питання інтеграції даних про трафік у реальному часі. У випадку CH це створює суттєві складнощі, оскільки зміна ваг ребер може вимагати часткової або повної перебудови ієрархії. OSRM дозволяє частково оновлювати ваги та структуру графа, однак такі операції є компромісом між актуальністю інформації та складністю і швидкістю обчислень.

Програмна реалізація OSRM використовує засоби візуалізації на базі бібліотеки Folium. Реалізація базується на програмному використанні HTTP API OSRM через функцію get_osrm_route(start, end, profile="car"), яка виконує звернення до публічного сервера http://router.project-osrm.org.

Функція get_osrm_route приймає три параметри: координати початкової точки start, координати цільової точки end, а також тип маршруту profile, де за замовчуванням використовується “car”, що трансформується у “driving”.

Для побудови URL використовується змінна coordinates, яка форматує координати у вигляді рядка, розділеного крапкою з комою.

Параметри запиту зберігаються у змінній params, яка визначає тип геометрії (“geojson”), рівень деталізації (“full”) та вимикає покрокову навігацію (“false”). Відповідь API парсується у змінну data, з якої витягується перший маршрут у data[‘routes’][0]. Геометрія маршруту зберігається в route[‘geometry’][‘coordinates’], а тривалість пересування обчислюється в хвилинах через route[‘duration’] / 60. Функція повертає маршрут у вигляді списку координат ([(lat, lon) for lon, lat in coords]) та тривалість поїздки.

У головній частині програми задається список п’яти стартових точок у змінній start_points, кожна з яких подана у вигляді кортежу (широта, довгота). Спільна кінцева точка задається окремо у змінній end_point.

Для візуалізації використовується об’єкт m, створений із допомогою folium.Map(), що центрований на end_point із масштабом zoom_start=14. Для диференціації маршрутів використовується список кольорів colors, який циклічно призначається кожному маршруту.

Далі реалізовано цикл for, у якому перебираються всі координати із start_points. У кожній ітерації викликається функція get_osrm_route(start, end_point), результати якої присвоюються змінним route і time_min. Отриманий маршрут додається до об’єкта карти m через folium.PolyLine, а початкова точка маркується за допомогою folium.Marker з відповідними стилями. У разі винятку (наприклад, помилка з’єднання або недоступність маршруту), виконується обробка через конструкцію try-except, яка виводить повідомлення про помилку, але не припиняє виконання програми.

Після побудови всіх маршрутів до карти додається маркер фінальної точки з описом «Кінцева точка» та іконкою ‘flag’, що позначає завершення всіх

маршрутів. Фінально карта зберігається у HTML-файлі за допомогою методу m.save(“osrm_5_routes.html”), що дозволяє інтерактивно переглядати маршрути у браузері.

Таким чином, запропонований алгоритм, реалізований через набір змінних та функцій (get_osrm_route, start_points, end_point, route, time_min, m), забезпечує ефективну інтеграцію з OSRM, дозволяючи будувати високошвидкісні маршрути та візуалізувати їх у геопросторовому середовищі.

Результат роботи OSRM для 5 точок приведені на рисунку 7.1.

Отримані маршрути з використанням OSRM приведені в Додатку М.1.

```
Маршрут 1: 4.9 хв від (50.4501, 30.5234) до (50.4547, 30.5166)
Маршрут 2: 2.5 хв від (50.4525, 30.527) до (50.4547, 30.5166)
Маршрут 3: 3.4 хв від (50.449, 30.52) до (50.4547, 30.5166)
Маршрут 4: 3.5 хв від (50.451, 30.532) до (50.4547, 30.5166)
Маршрут 5: 2.4 хв від (50.453, 30.518) до (50.4547, 30.5166)
```

Рисунок 7.1 – Результат роботи OSRM для 5 точок

7.2 API GraphHopper Directions

GraphHopper – це система побудови маршрутів, яка поєднує високу продуктивність із широкими можливостями кастомізації, орієнтуючись на гнучкість обробки запитів при збереженні ефективності [43]. На відміну від рішень, що строго дотримуються єдиного підходу, GraphHopper підтримує декілька алгоритмічних стратегій, серед яких Contraction Hierarchies (CH) та ALT (A, Landmarks, Triangle inequality). Обидва методи можуть бути активовані через параметри конфігурації системи. Типово, CH застосовується для профілів з критеріями найшвидшого або найкоротшого шляху, тоді як ALT може використовуватись для інших оптимізаційних сценаріїв.

Особливістю GraphHopper є підтримка гнучкого режиму (Flex Mode), що дозволяє відмовитися від заздалегідь побудованих ієрархій (CH/ALT) на користь динамічних алгоритмів, таких як класичний Дейкстра або A*. Цей режим особливо важливий при роботі з динамічними вагами графа – зокрема, коли

потрібно враховувати реальний трафік, локальні обмеження, аварійні перекриття доріг або персоналізовані вподобання користувача. Гнучкий режим працює або на повному графі, або на помірно скороченому, дозволяючи адаптувати пошук маршруту до поточних умов.

GraphHopper реалізований на мові Java, що забезпечує кросплатформність, легку інтеграцію з іншими Java-додатками та широке використання в мобільних і серверних рішеннях. Завдяки відкритому вихідному коду, система активно модифікується та розширюється спільнотою.

Ще однією важливою перевагою є підтримка кастомних моделей ваг. Користувачі можуть задавати складні правила маршрутизації, де на вагу ребра впливають не лише геометричні характеристики (довжина, час проїзду), а й контекстуальні – тип покриття, категорія дороги, обмеження на повороти, пріоритетні або заборонені зони. Це робить GraphHopper ефективним інструментом як для типових навігаційних завдань, так і для спеціалізованих застосунків, наприклад, у логістиці чи волонтерських операціях.

Програмна реалізація використовує API платформи GraphHopper, що спеціалізується на швидкому й масштабованому обчисленні маршрутів на основі OpenStreetMap. Основна функція get_graphhopper_route(start, end, api_key) забезпечує комунікацію з віддаленим сервісом маршрутизації через HTTP-запит, де початкова й кінцева точки передаються у вигляді координат, а автентифікація здійснюється через унікальний ключ доступу api_key.

В межах функції формуються параметри запиту до URL https://graphhopper.com/api/1/route. Вони включають координати точок у вигляді двох елементів масиву “point”, транспортний профіль “vehicle”: “car”, мову локалізації “locale”: “en”, прапорець для побудови маршруту “calc_points”: “true”, вимогу до неенкодованих координат “points_encoded”: “false”, а також ключ доступу. У відповідь повертається структура даних JSON, з якої витягується найкоротший шлях у data[‘paths’][0]. Координати маршруту містяться у вкладеній структурі path[‘points’][‘coordinates’], а тривалість руху

визначається у мілісекундах і перетворюється у хвилини через поділ на 60000. Після перетворення координат у формат (широта, довгота), функція повертає їх разом із часом пересування у хвилинах.

Основна частина скрипта містить список стартових точок starts, кожна з яких подана у форматі (lat, lon), та кінцеву точку end, розташовану в тій самій географічній області. Об’єкт карти m, створений за допомогою folium.Map(), ініціалізується на цільовій позиції з відповідним масштабом (zoom_start=14). Для візуального розрізнення маршрутів використовується список кольорів colors, який застосовується по модулю до кожної траєкторії.

У циклі for, що перебирає індекс та координати зі списку starts, кожна стартова точка обробляється за допомогою функції get_graphhopper_route. Отримані результати призначаються змінним route_points та time_min, які далі використовуються для відображення маршруту на карті у вигляді folium.PolyLine, де кожна лінія має відповідний колір і підпис із тривалістю подорожі. Також для кожної стартової точки створюється folium.Marker з відповідною іконкою ‘user’, що відображає її положення. У разі винятків, пов’язаних із відсутністю маршруту або помилкою з’єднання, виконання переходить у блок except, де фіксується номер стартової точки та опис проблеми.

Завершальним кроком є додавання маркера для цільової точки за допомогою folium.Marker, що позначається іконкою ‘flag’ зеленого кольору. Весь результат інтерактивної побудови маршруту фіксується у HTML-файлі graphhopper_multi_start.html, який може бути відкритий у браузері для подальшого аналізу. Таким чином, структура алгоритму, що використовує змінні starts, end, api_key, m, а також функцію get_graphhopper_route, демонструє практичне застосування GraphHopper для задач просторової маршрутизації з декількох точок походження до однієї мети.

Результат роботи GraphHopper для 5 точок приведені на рисунку 7.2.

Отримані маршрути з використанням GraphHopper приведені в Додатку М.2.

```
Маршрут 1: 5.6 хв від (50.4501, 30.5234) до (50.4547, 30.5166)
Маршрут 2: 2.7 хв від (50.452, 30.52) до (50.4547, 30.5166)
Маршрут 3: 3.4 хв від (50.448, 30.525) до (50.4547, 30.5166)
Маршрут 4: 10.8 хв від (50.4515, 30.53) до (50.4547, 30.5166)
Маршрут 5: 3.8 хв від (50.4495, 30.519) до (50.4547, 30.5166)
```

Рисунок 7.2 – Результат роботи GraphHopper для 5 точок

7.3 REST API OpenRouteService (ORS)

OpenRouteService (ORS) – це комплексна геопросторова платформа, яка надає широкий спектр сервісів маршрутизації та просторового аналізу на базі відкритих даних OpenStreetMap. Основою ORS є рушій GraphHopper, завдяки чому система успадковує його алгоритмічну гнучкість та продуктивність. Однак на відміну від самого GraphHopper, ORS орієнтований передусім на надання готових API-сервісів для різноманітних користувацьких і аналітичних сценаріїв.

Для типових запитів типу A-до-B (один маршрут між двома точками), ORS застосовує Contraction Hierarchies (CH), що дозволяє досягти високої швидкості відповіді. Для складніших задач, таких як побудова зон досяжності або генерація матриць відстаней, ORS використовує адаптовані варіанти класичних алгоритмів (Дейкстра, A*) в поєднанні з можливими оптимізаціями CH або ALT.

Зони доступності реалізуються через пошук графа «вшир» від джерела до досягнення заданого обмеження (наприклад, 30 хвилин пішки або 10 км). Такий підхід вимагає сканування великої частини графа без визначеної цільової точки, тож CH застосовується лише частково або не використовується взагалі. Матриці відстаней – це ще одна типова задача, яка передбачає обчислення парних маршрутів між багатьма точками. Для цього ORS виконує оптимізовані багатопоточні запити, використовуючи попередньо підготовлені ієрархії або інші прискорювачі.

З функціональної точки зору, ORS надає великий набір маршрутизаційних профілів, зокрема: автомобіль, вантажівка, велосипед, пішохід, користувач на інвалідному візку тощо. Кожен профіль має відповідні налаштування обмежень, допустимих типів доріг, швидкісних режимів і додаткових параметрів

(наприклад, уникнення платних доріг, поромів або тунелів). Завдяки цьому ORS зручний як для звичайних користувачів, так і для спеціалізованих сервісів з високим рівнем адаптації до контексту – від гуманітарної логістики до мобільності маломобільних груп населення.

Програмну побудову маршрутів між кількома стартовими точками та однією фіксованою кінцевою, використовуючи хмарний сервіс OpenRouteService (ORS), що базується на OpenStreetMap і забезпечує маршрутизацію через REST API. Основною функцією є get_ors_route(start, end, api_key), яка відповідає за надсилання HTTP POST-запиту до ендпойнту https://api.openrouteservice.org/v2/directions/driving-car/geojson. Вона приймає координати початкової та кінцевої точок разом із ключем доступу api_key, який необхідний для авторизації користувача в системі.

Передача запиту виконується через бібліотеку requests, із зазначенням заголовків headers, які містять параметри Authorization (зі значенням API-ключа) та Content-Type зі значенням ‘application/json’. Тіло запиту, представлене у вигляді словника body, включає координати у форматі [довгота, широта], що є стандартом GeoJSON. Відповідь з API обробляється у форматі JSON, і після перевірки наявності ключа “features” та його вмісту, з неї вилучаються координати маршруту із вкладеного об’єкта data[‘features’][0][‘geometry’][‘coordinates’], а також тривалість маршруту з data[‘features’][0][‘properties’][‘summary’][‘duration’], яка конвертується з секунд у хвилини.

Функція повертає пару – список географічних точок маршруту у вигляді (lat, lon) і відповідну тривалість маршруту в хвилинах. У головному тілі скрипта задано список стартових точок starts, кожна з яких є координатною парою (широта, довгота). Цільова точка end задається аналогічно, а ключ API api_key використовується для автентифікації при кожному зверненні до ORS.

Інтерактивна карта створюється за допомогою folium.Map(), яка центрується на кінцевій точці з рівнем масштабування zoom_start=14. Для

кольорового відображення маршрутів визначено список colors, з якого кольори обираються по черзі відповідно до індексу. Цикл for i, start in enumerate(starts) проходить по кожній стартовій точці, викликає функцію get_ors_route для отримання маршруту, а далі додає на карту лінію маршруту через folium.PolyLine з підказкою, що містить номер старту та час подорожі у хвилинах.

Кожна стартова точка додатково позначається маркером folium.Marker, з іконкою користувача (параметр icon='user') червоного кольору. У разі помилки запиту або відсутності маршруту, винятки перехоплюються через конструкцію try-except, і результат логування виводиться у консоль. Кінцева точка маршруту позначена окремим зеленим маркером із іконкою прапора.

У фіналі, створена карта з усіма маршрутами зберігається у файл ors_multi_start.html, який містить усю побудовану просторову інформацію. Таким чином, змінні starts, end, api_key, m, а також функція get_ors_route формують повноцінний модуль для інтерактивного відображення мультистартової маршрутизації з використанням OpenRouteService.

Результат роботи OpenRouteService для 5 точок приведені на рисунку 7.3.

Отримані маршрути з використанням OpenRouteService приведені в Додатку М.3.

```
Маршрут 1: 5.0 хв від (50.4501, 30.5234) до (50.4547, 30.5166)
Маршрут 2: 3.6 хв від (50.452, 30.52) до (50.4547, 30.5166)
Маршрут 3: 5.0 хв від (50.448, 30.525) до (50.4547, 30.5166)
Маршрут 4: 5.5 хв від (50.4515, 30.53) до (50.4547, 30.5166)
Маршрут 5: 4.1 хв від (50.4495, 30.519) до (50.4547, 30.5166)
```

Рисунок 7.3 – Результат роботи OpenRouteService для 5 точок

Скрипти реалізації Open Source Routing Machine (OSRM), GraphHopper та OpenRouteService (ORS) представлені у Додатках Н, П, Р відповідно.

У контексті порівняльного аналізу систем маршрутизації (Open Source Routing Machine (OSRM), GraphHopper та OpenRouteService (ORS)) увагу приділено оцінці ефективності побудови маршрутів у межах одного міського

середовища. Критерієм порівняння слугувала тривалість маршруту від п’яти різних стартових точок до однієї фіксованої кінцевої точки, що дозволяє встановити як середню ефективність сервісів, так і їхню стабільність.

За результатами обчислень, OSRM продемонструвала найкращу продуктивність. Середній час маршруту, розрахований за всіма п’ятьма початковими координатами, становив 3,34 хвилини, що є найнижчим показником серед трьох протестованих систем. Максимальна тривалість маршруту в межах OSRM не перевищувала 4,9 хвилини, а мінімальна – 2,4 хвилини, що свідчить про високу узгодженість алгоритму Contraction Hierarchies, покладеного в основу цієї системи. Така стабільність є ознакою ефективної оптимізації вуличного графа з урахуванням локальних особливостей міського руху.

У свою чергу, GraphHopper продемонстрував найменшу стабільність: середній час склав 5,26 хвилини, а один з маршрутів перевищив 10 хвилин, що є майже вдвічі більше за середнє значення інших маршрутів у тій самій системі. Це вказує на певні недоліки в алгоритмі Multi-Level Dijkstra або обмеження у локальній деталізації дорожнього графа. Незважаючи на те, що GraphHopper підтримує багаторівневу ієрархію, вона може втрачати ефективність при побудові коротких міських маршрутів.

OpenRouteService забезпечив більш рівномірний, але все ж посередній результат – середній час маршруту становив 4,64 хвилини, максимальний – 5,5 хвилин, а мінімальний – 3,6 хвилини. Це свідчить про достатню, проте менш агресивну оптимізацію маршрутів у порівнянні з OSRM. Враховуючи використання OpenStreetMap як єдиного джерела геоданих, відмінності у результатах, ймовірно, зумовлені саме відмінностями в алгоритмічній обробці графа та евристиках, які застосовуються на серверному рівні.

У межах повторного порівняльного дослідження трьох сучасних маршрутних сервісів було здійснено аналіз їх роботи у міському середовищі району Газіосманпаша, розташованого на європейській стороні Стамбула.

OSRM демонструє загалом найкращі результати. Середня тривалість маршруту становить 8,76 хвилини, з мінімальним значенням у 3,5 хвилини та максимальним у 13,0 хвилин.

GraphHopper, у порівнянні з OSRM, демонструє погіршену продуктивність: середній час маршруту складає 10,36 хвилини. Хоча мінімальний показник лише на одну хвилину більший (4,4 хвилини), максимальний сягає 15,0 хвилин, що вказує на наявність маршрутів з менш оптимальними евристиками.

OpenRouteService (ORS) посів проміжне положення між двома іншими системами. Середній маршрутний час становить 9,22 хвилини, що є кращим за GraphHopper, але дещо гіршим за OSRM. ORS виявився особливо ефективним у маршруті з найкоротшим часом (3,6 хвилини), а його максимальний показник – 13,3 хвилини – свідчить про стриману, але узгоджену евристичну політику.

Загалом, у межах району Газіосманпаша OSRM знову підтвердив свою лідерську позицію.

Порівняння тривалості п’яти маршрутів у Києві та Стамбулі для трьох сервісів побудови маршрутів – OSRM, GraphHopper та OpenRouteService (ORS), наведено в Додатку С.

Таким чином результати експериментального моделювання дозволяють стверджувати, що OSRM є найбільш ефективним інструментом для побудови маршрутів у щільній міській інфраструктурі. Його алгоритмічна архітектура забезпечує як швидкодію, так і стабільність, що є визначальними факторами для задач навігації в реальному часі.

Висновок до розділу 7

OSRM є найефективнішим сервісом маршрутизації в цьому порівнянні. Він забезпечує найменший середній час проїзду, найшвидші маршрути та стабільні результати без різких коливань тривалості. Його використання

алгоритму Contraction Hierarchies виявляється вигідним у щільних міських умовах. GraphHopper і ORS демонструють гірші результати як у середньому, так і в крайніх значеннях, при цьому GraphHopper показує найбільшу нестабільність.

У практичному застосуванні це означає, що OSRM – оптимальний вибір для задач реального часу з кількома стартовими точками та вимогою до високої точності та швидкодії.

РОЗДІЛ 8
АВТОМАТИЗОВАНА СИСТЕМА РОЗПОДІЛУ ТА МАРШРУТИЗАЦІЇ ВОЛОНТЕРІВ

Автоматизація процесів розподілу та маршрутизації дозволяє значно скоротити час реагування й забезпечити цільове використання людських ресурсів. Розроблений скрипт являє собою автоматизовану систему розподілу та маршрутизації волонтерів під час катастроф. Його основна мета – оптимізація процесу виявлення волонтерів із необхідними навичками в межах заданого радіусу від визначеної географічної точки (на прикладі м. Стамбул). Програма здійснює пошук таких волонтерів, аналізує їхню доступність та відображає оптимальні маршрути до місць потреби на інтерактивній карті.

8.1 Опис функціональності скрипта

Вхідні дані скрипта формують основу для реалізації процесів пошуку, фільтрації та візуалізації волонтерів із необхідними навичками в межах постраждалого регіону. Одним з ключових елементів є набір абстрактних координат points_abstract, що моделює просторовий розподіл волонтерів. Ці координати формуються у вигляді масиву з 100 двовимірних точок у нормалізованому масштабі [0, 1], які згодом трансформуються у географічні широти та довготи відносно центральної координати – міста Стамбул (ISTANBUL_COORDS) за допомогою функції transform_points_to_coords().

Центральна точка (ISTANBUL_COORDS = (41,0786; 28,8968)) використовується як опорна позиція для геопросторових обчислень.

Абстрактний радіус пошуку radius_abstract, що задається у відносних одиницях (наприклад, 0,4), трансформується у метри через масштабуючий коефіцієнт COORDINATE_SCALE_FACTOR за допомогою функції

abstract_radius_to_meters(). Результат зберігається у змінній radius_meters, яка визначає фактичну зону пошуку.

Координата запиту (тобто точка, від якої ведеться пошук) задається змінною query_point_abstract, а її географічне представлення формується у query_point_coords за допомогою функції transform_single_point_to_coords().

Ідентифікатори волонтерів формуються у вигляді списку міток labels, які відповідають позиціям у масиві координат. Для фільтрації волонтерів за визначеним вмінням (наприклад, «Лікар», змінна skill_to_search) використовується функція get_volunteers_by_skill(), яка звертається до бази даних volonter.db. Вона обробляє результати пошуку за індексами final_indices, сформованими на основі просторового аналізу, та повертає список імен релевантних волонтерів volunteers_found.

8.2 Функції трансформації координат

Функція abstract_radius_to_meters перетворює абстрактні величини радіуса пошуку в конкретну відстань у метрах. Це досягається шляхом множення абстрактного радіуса на заданий коефіцієнт масштабування. Використання абстрактного радіуса на початковому етапі спрощує подання та обробку відстаней, дозволяючи працювати з умовними значеннями, не переходячи одразу до географічних координат. Це дає змогу легко змінювати масштаб пошуку без складних обчислень на початку. Однак для точного визначення відстаней на місцевості та використання бібліотек, які оперують фізичними одиницями вимірювання, такими як метри, необхідне перетворення до реальних одиниць. Такий підхід забезпечує гнучкість у налаштуванні радіуса пошуку, а потім узгоджує його з конкретними геопросторовими розрахунками.

Функція transform_points_to_coords відповідає за перетворення масиву абстрактних координат у їхні географічні аналоги, представлені широтою та довготою. Процес трансформації базується на визначених центральних

координатах, у даному випадку – координатах Стамбула, та вже згаданому коефіцієнті масштабування. Кожна абстрактна координата, являє собою відносне зміщення відносно центральної точки у певній системі координат. Такий метод спрощує генерацію та первинну обробку координат волонтерів.

Функція transform_single_point_to_coords виконує аналогічне завдання з перетворення абстрактної координати в географічну, але вже для окремої точки. Її призначення полягає в обробці одиничних координат, що може бути необхідно в різних частинах скрипта, наприклад, для перетворення координат запиту або центральної точки пошуку.

8.3 Метрики відстані

Функція geodesic_distance використовується для обчислення геодезичної відстані між двома точками, розташованими на земній кулі. Для реалізації цього завдання скрипт використовує можливості бібліотеки geopy.

Геодезична відстань є найбільш точним способом вимірювання відстані між двома точками на поверхні еліпсоїда, яким є Земля, оскільки вона враховує кривизну планети. Застосування цієї метрики особливо важливе для точних вимірювань на значних відстанях, де ігнорування сферичності Землі може призвести до суттєвих похибок.

Функція haversine_distance реалізує обчислення відстані між двома точками на сфері за допомогою формули Гаверсина. Формула Гаверсина є тригонометричною формулою, яка дозволяє розрахувати відстань великого кола між двома точками на сфері за їхніми координатами (широтою та довготою).

8.4 Алгоритми пошуку

Функція brute_force_search реалізує прямий перебір усіх волонтерів, послідовно обчислюючи геодезичну відстань до кожної точки та перевіряючи,

чи входить вона у заданий радіус. Цей підхід простий у реалізації, але неефективний при великому обсязі даних. Функція kd_tree_search використовує структуру KD-дерева для пришвидшеного пошуку в евклідовому просторі, де координати попередньо масштабуються з урахуванням широти. Це дозволяє значно зменшити час пошуку, хоча й вносить деяку похибку через спрощення просторової метрики. Функція ball_tree_search, на відміну від KD-підходу, працює без масштабування, застосовуючи Ball-дерево разом із гаверсиновою відстанню. Такий підхід краще відповідає географічній природі даних, забезпечуючи більшу точність у метричному просторі.

8.5 Пошук волонтерів за вмінням

Функція get_volunteers_by_skill відповідає за фільтрацію знайдених волонтерів за їхніми вміннями. Вона приймає як вхідні параметри індекси волонтерів, які були знайдені в заданому радіусі за допомогою одного з алгоритмів пошуку, список доступних міток, назву вміння, яку необхідно знайти, а також шлях до бази даних SQLite, де зберігається інформація про волонтерів.

Першим кроком функція встановлює з’єднання з базою даних SQLite. Потім формується та виконується SQL-запит, який вибирає імена волонтерів з таблиці бази даних, чиї індекси відповідають переданим індексам знайдених точок і які мають зазначене вміння.

Результатом виконання запиту є список імен волонтерів, які відповідають обом критеріям – знаходяться в заданому радіусі та володіють потрібним вмінням.

8.6 Отримання маршруту за допомогою OSRM

Функція get_osrm_route відповідає за отримання маршруту між двома географічними точками через API сервісу OSRM. Вона приймає координати

початку й кінця маршруту та транспортний профіль (типово – “car”), формує HTTP-запит і обробляє відповідь у форматі GeoJSON. У результаті повертаються координати маршруту та орієнтовна тривалість у хвилинах. Така інтеграція дозволяє використовувати зовнішній сервіс для точного та зручного розрахунку маршрутів без реалізації власного алгоритму навігації.

8.7 Основний блок скрипта

Основний блок скрипта виконує послідовність дій для пошуку та відображення волонтерів. На початковому етапі генерується набір випадкових абстрактних координат для 100 волонтерів. Потім задаються абстрактна координата, що представляє точку запиту (наприклад, місцезнаходження користувача), та радіус пошуку навколо цієї точки. Після цього абстрактні координати як волонтерів, так і точки запиту трансформуються в географічні координати (широту та довготу) за допомогою відповідних функцій.

Далі виконується пошук волонтерів, що знаходяться в межах заданого радіуса від точки запиту, використовуючи всі три реалізовані алгоритми: прямий перебір, пошук за KD-деревом та пошук за Ball-деревом. Після завершення пошуку результати, отримані за допомогою різних алгоритмів, порівнюються. Якщо всі три алгоритми повертають ідентичний набір індексів волонтерів, скрипт продовжує роботу, використовуючи один із цих результатів. У випадку виявлення розбіжностей між результатами алгоритмів виводиться попередження, що може свідчити про потенційні проблеми з реалізацією або налаштуванням одного з алгоритмів.

На наступному кроці серед знайдених волонтерів здійснюється пошук осіб з певним вмінням, яка за замовчуванням встановлена як «Лікар». На завершення основний блок скрипта виводить імена та географічні координати волонтерів, які були знайдені в заданому радіусі та володіють вказаною навичкою.

Завершальним етапом роботи скрипта є побудова інтерактивної карти за допомогою бібліотеки folium, яка відображає знайдених волонтерів і маршрути до них. Центр карти встановлюється на координати Стамбула. Для кожного волонтера формується маршрут до центру за допомогою функції get_osrm_route, а отримані шляхи додаються у вигляді кольорових ліній. Також розміщуються маркери: для кожного волонтера – з його іменем, а для центру – з відповідною позначкою. Готова карта зберігається у файлі volunteers_routes_map.html, що дає змогу переглядати її в браузері без додаткового ПЗ.

Результати пошуку в терміналі волонтерів з навиками «Будівельник» приведені на рисунку 8.1.

Отримані маршрути цих волонтерів до місця події приведені в Додатку Т.1.

```
Знайдено волонтерів (навичка: Будівельник) у радіусі: 2
Волонтер_1 – (41.10012, 28.90168)
Волонтер_73 – (41.09899, 28.92742)
Маршрут 1: 7.8 хв від Волонтер_1
Маршрут 2: 11.1 хв від Волонтер_73
```

Рисунок 8.1 – Результат пошуку в терміналі

Розглянемо всі етапи роботи системи розподілу та маршрутизації волонтерів на прикладі вміння «Лікар» – візуалізація етапів представлена Додатках Т.2-Т.5.

Повний скрипт автоматизованої системи розподілу та маршрутизації волонтерів під час катастроф NaTech наведено в Додатку Ш.

Висновок до розділу 8

Розроблена геопросторова система є комплексним інструментом для пошуку волонтерів із необхідними вміннями в заданому радіусі, який поєднує точність обчислень, ефективність обробки великих обсягів геоданих та інтеграцію з реальними транспортними маршрутами. Завдяки використанню

бібліотек geopy, folium, API OSRM та застосуванню гаверсинової формули з прив’язкою до центральної точки, система враховує кривизну Землі, що забезпечує коректність розрахунків відстаней. Впровадження трьох різних алгоритмів пошуку (Brute Force, KD-Tree та Ball-Tree) дає змогу підвищити надійність системи, забезпечуючи верифікацію результатів шляхом порівняння та обрати оптимальний варіант для подальшої роботи.

Завдяки поєднанню алгоритмічної оптимізації, географічної фільтрації та візуалізації результатів на інтерактивній мапі, рішення не тільки підвищує точність пошуку, а й забезпечує гнучкість у налаштуванні під різні сценарії використання. Крім того, інтеграція з базою даних SQLite для зберігання волонтерських профілів за вміннями дозволяє динамічно формувати запити відповідно до потреб ситуації. Таким чином, система функціонує як масштабована платформа підтримки прийняття рішень для ефективної мобілізації ресурсів у кризових умовах, поєднуючи технології просторового аналізу, оптимізації та зручного візуального інтерфейсу.

ВИСНОВКИ

У кваліфікаційній роботі здійснено дослідження розподілу та маршрутизації волонтерів у контексті техногенних катастроф типу NaTech. З огляду на характер таких катастроф обґрунтовано необхідність створення інструментів, здатних адаптуватися до швидкоплинної, непередбачуваної ситуації в умовах високої щільності міського середовища, зруйнованої інфраструктури та каскадного розвитку подій.

Досліджено ризики, що виникають внаслідок поєднання природних і техногенних катастроф у містах. Виявлено, що висока концентрація населення та інфраструктури посилює вразливість перед природними й техногенними загрозами, особливо коли вони взаємодіють між собою, утворюючи складні каскадні ланцюги подій із непередбачуваними наслідкам. Катастрофи NaTech вирізняються складністю реагування через залежності в інфраструктурі та її деградацію.

Проведено аналіз існуючих методів та моделей, що застосовуються для розподілу та маршрутизації волонтерів під час ліквідації наслідків катастроф. Аналіз показав, що сучасні моделі прагнуть враховувати широкий спектр факторів, включаючи компетентності та вподобання волонтерів, пріоритетність потреб постраждалих, специфіку завдань, часові та геопросторові обмеження, логістику, динамічність ситуації, невизначеність та ризики. Розглянуто методи залучення та призначення завдань спонтанним волонтерам, з акцентом на централізовану координацію, використання технологій (онлайн-платформи, мобільні додатки) та гнучких підходів, таких як рекомендаційні системи. Результати цього аналізу створили теоретичну основу для розробки алгоритмів та функціоналу автоматизованої системи розподілу та маршрутизації волонтерів.

Технічною основою розробленої системи став Python як гнучке середовище реалізації логіки пошуку, обробки просторових запитів та візуалізації. Розроблено та реалізовано алгоритми пошуку волонтерів у заданому

радіусі із застосуванням структур просторового розподілу: K-D Tree, Ball Tree, QuadTree та Brute-force. Реалізовано модулі аналізу геопросторових даних, динамічного пошуку доступних волонтерів і побудови маршрутів. Алгоритми були підібрані та протестовані з урахуванням різних типів завдань і особливостей міського середовища. Порівняльна оцінка KD-Tree, Ball Tree, Quadtree та прямого перебору дозволила встановити придатність кожного методу залежно від характеристик вхідних даних і вимог до швидкодії. Враховано також викривлення, пов’язані з кривизною поверхні Землі, що підвищило точність просторових обчислень.

Результати дослідження однозначно вказують на перевагу KD-Tree як базового методу для ефективного пошуку найближчих сусідів у великих множинах точок. Ball-Tree може слугувати альтернативою в умовах складного або багатовимірного розподілу даних. Натомість Brute-force і QuadTree не забезпечують достатнього рівня масштабованості й можуть розглядатися лише в якості контрольних або допоміжних методів.

Особливу увагу приділено побудові маршрутів у міському середовищі. У межах дослідження проведено порівняльний аналіз алгоритмів маршрутизації Дейкстри, A* і Флойда-Уоршелла. У результаті встановлено доцільність застосування A* або Дейкстри як найбільш збалансованих за швидкістю та точністю. Алгоритми Дейкстра та A* підходять для швидкого пошуку в малих і середніх графах, тоді як Флойд-Уоршелл зручний для знаходження всіх найкоротших шляхів у великих графах, але не підходить для задач з конкретною парою точок через високу обчислювальну складність.

Проаналізовано можливості зовнішніх сервісів маршрутизації для забезпечення швидкого доступу волонтерів до місць потреби, зокрема OSRM, GraphHopper, ORS. Виявлено, що OSRM продемонстрував найкращі результати в умовах обмеженого часу та високої щільності графів міської інфраструктури. У практичному застосуванні це означає, що OSRM – оптимальний вибір для

задач реального часу з кількома стартовими точками та вимогою до високої точності та швидкодії.

Розроблена автоматизована геопросторова система, яка демонструє здатність оперативно знаходити найближчих волонтерів, аналізувати їх відповідність критеріям задачі та забезпечувати швидке прокладання маршрутів до цілей. Завдяки інтеграції з базою даних, використанню географічної візуалізації та адаптивному вибору алгоритмів, забезпечено гнучке реагування на зміну умов. Система може ефективно масштабуватись, підтримуючи роботу як у локальних, так і в регіональних сценаріях реагування.

Виконано тестування ефективності розроблених підходів на модельних сценаріях із застосуванням симуляцій на прикладі міста Стамбул.

Таким чином, розроблена автоматизована система розподілу та маршрутизації волонтерів під час катастроф NaTech виступає як гнучкий інструмент для ухвалення рішень у надзвичайних ситуаціях, забезпечуючи оперативне залучення ресурсів завдяки інтеграції методів геоаналітики, алгоритмічної оптимізації та інтуїтивно зрозумілого візуального середовища. Система функціонує як масштабована платформа підтримки прийняття рішень для ефективної мобілізації ресурсів у кризових умовах, поєднуючи технології просторового аналізу, оптимізації та зручного візуального інтерфейсу.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. UN Office for Disaster Risk Reduction. URL: https://www.undrr.org/terminology/disaster (дата звернення: 10.08.2025).

2. An active year of U.S. billion-dollar weather and climate disasters. URL: https://www.climate.gov/news-features/blogs/beyond-data/2024-active-year-us-billion-dollar-weather-and-climate-disasters (дата звернення: 10.08.2025).

3. European Environment Agency. Technical accidents and natural disasters. URL: https://www.eea.europa.eu (дата звернення: 10.08.2025).

4. Управління кіберінцидентами Державна служба спеціального зв'язку та захисту інформації. URL: https://cip.gov.ua/api/attachmen (дата звернення: 10.08.2025).

5. Technological hazard. URL: https://www.preventionweb.net/knowledge-base/hazards/technical-disaster (дата звернення: 10.08.2025).

6. 2024 Global assessment report on disaster risk reduction. URL: https://digitallibrary.un.org/record/4063343?v=pdf (дата звернення: 10.08.2025).

7. Natural disasters are marking 2025 as a 'Scary Year'. URL: https://www.observerbd.com/news/507299 (дата звернення: 10.08.2025).

8. Cat DDO URL: https://www.worldbank.org/en/topic/disasterriskmanagement (дата звернення: 10.08.2025).

9. Японське агентство з управління катастрофами (FDMA). URL: http://www.fdma.go.jp (дата звернення: 10.08.2025).

10. Sperling M., Schryen G. Decision support for disaster relief: coordinating spontaneous volunteers. *Eur J Oper Res*. 2022. № 299 (2). P. 690-705.

11. Wang Q., Reed A., Nie X. Joint initial dispatching of official responders and registered volunteers during catastrophic mass-casualty incidents. *Transport Res Part E: Log Transport Rev*. 2022. № 159. URL: https://doi.org/10.1016/j.tre.2022.102648 (дата звернення: 10.08.2025).

12. Improving volunteer productivity and retention during humanitarian relief efforts. URL: https://www.researchgate.net/publication/261803944_Improving_Volunteer_Productivity_and_Retention_during_Humanitarian_Relief_Efforts (дата звернення: 17.08.2025).

13. Optimization of volunteer task assignments to improve volunteer retention and nonprofit organizational performance. URL: https://www.sciencedirect.com/science/article/abs/pii/S0038012122001872 (дата звернення: 17.08.2025).

14. A model for assignment of rescuers considering multiple disaster areas. URL: https://www.sciencedirect.com/science/article/abs/pii/ S2212420919300603?via%3Dihub (дата звернення: 17.08.2025).

15. Modeling uncertain task compliance in dispatch of volunteers to out-of-hospital cardiac arrest patients. URL: https://www.sciencedirect.com/science/article/pii/S0360835221004198?via%3Dihub (дата звернення: 17.08.2025).

16. An optimization model for volunteer assignments in humanitarian organizations. URL: https://www.sciencedirect.com/science/article/abs/pii/S0038012112000353?via%3Dihub (дата звернення: 17.08.2025).

17. Rabiei P., Arias-Aranda D., Stantchev V. Introducing a novel multi-objective optimization model for volunteer assignment in the post-disaster phase: combining fuzzy inference systems with nsga-ii and nrga. *Expert Syst Appl*. 2023. № 226. URL: https://doi.org/10.1016/j.eswa.2023.120142 (дата звернення: 10.08.2025).

18. A robust optimization approach to volunteer management in humanitarian crises. URL: https://www.sciencedirect.com/science/article/abs/pii/ S0925527315000523?via%3Dihub (дата звернення: 17.08.2025).

19. Fei L., Wang Y. An optimization model for rescuer assignments under an uncertain environment by using Dempster–Shafer theory. *KB Syst.* 2022. № 255. URL: https://doi.org/10.1016/j.knosys.2022.109680 (дата звернення: 10.08.2025).

20. The optimal assignment of spontaneous volunteers. URL: https://www.tandfonline.com/doi/full/10.1057/s41274-017-0219-2 (дата звернення: 17.08.2025).

21. Optimal control of parallel queues for managing volunteer convergence. URL: https://journals.sagepub.com/doi/10.1111/poms.13224 (дата звернення: 17.08.2025).

22. Assigning spontaneous volunteers to relief efforts under uncertainty in task demand and volunteer availability. URL: https://www.sciencedirect.com/science/article/abs/pii/S0305048319300611?via%3Dihub (дата звернення: 17.08.2025).

23. Managing volunteer convergence at disaster relief centers. URL: https://www.sciencedirect.com/science/article/abs/pii/S0925527319301999?via%3Dihub (дата звернення: 17.08.2025).

24. Paret K., Rodriguez S. A., Mayorga M. E., Velotti L., Lodree E. J. Agent-based simulation of spontaneous volunteer convergence to improve disaster planning. *Nat Hazards Rev*. 2023. Vol. 24, No. 2. URL: https://doi.org/10.1061/NHREFO.NHENG-1659 (дата звернення: 10.08.2025).

25. Schmidt A., Albert L. Task recommendations for self-assigning spontaneous volunteers. *Comput Ind Eng*. 2022. № 163. URL: https://doi.org/10.1016/j.cie.2021.107798 (дата звернення: 10.08.2025).

26. Modeling uncertain task compliance in dispatch of volunteers to out-of-hospital cardiac arrest patients. URL: https://www.sciencedirect.com/science/article/pii/S0360835221004198?via%3Dihub (дата звернення: 17.08.2025).

27. Two-sided matching model for assigning volunteer teams to relief tasks in the absence of sufficient information. URL: https://www.sciencedirect.com/science/article/abs/pii/S0950705121007577?via%3Dihub (дата звернення: 17.08.2025).

28. Volunteer multi-person multi-task optimization dispatch method considering two-sided matching. URL: https://link.springer.com/article/10.1007/s00500-022-06784-8 (дата звернення: 17.08.2025).

29. Лекція 3. Метод k найближчих сусідів. URL: https://om.knu.ua/users_upload/15/upload/file/pr_lecture_03.pdf (дата звернення: 17.08.2025).

30. Chen X, Güttel S. Fast and exact fixed-radius neighbor search based on sorting. *PeerJ Computer Science*. 2024. URL: https://doi.org/10.7717/peerj-cs.1929 (дата звернення: 10.08.2025).

31. KD-Tree and Ball Tree in KNN Algorithm. URL: https://medium.com/@narasimharaodevisetti14/kd-tree-and-ball-tree-in-knn-algorithm-09a86d1bc6e6 (дата звернення: 10.08.2025).

32. Define a ball tree and its applications in nearest neighbor searches. URL: https://www.beyond-tutors.com/resources/faq/define-a-ball-tree-and-its-applications-in-nearest-neighbor-searches/ (дата звернення: 10.08.2025).

33. Nearest Neighbors. URL: https://scikit-learn.org/stable/modules/neighbors.html (дата звернення: 10.08.2025).

34. Quad Trees: A Data Structure for Retrieval on Composite Keys. URL: https://www.researchgate.net/publication/220197855_Quad_Trees_A_Data_Structure_for_Retrieval_on_Composite_Keys (дата звернення: 10.08.2025).

35. Shortest path: Dijkstra’s algorithm, Bellman-Ford algorithm. URL: https://www.uvm.edu/~cbcafier/cs2240/content/13_graph_algos/01_shortest_path.html (дата звернення: 10.10.2025).

36. Dijkstra’s single-source shortest path algorithm. URL: https://www.cs.cornell.edu/courses/cs2112/2021fa/lectures/lecture.html?id=ssp (дата звернення: 10.10.2025).

37. The A* Algorithm: A Complete Guide. URL: https://www.datacamp.com/tutorial/a-star-algorithm (дата звернення: 10.10.2025).

38. Floyd-Warshall Algorithm. URL: https://www.tutorialspoint.com/data_structures_algorithms/floyd_warshall_algorithm.htm (дата звернення: 10.10.2025).

39. Routing Faster Than Dijkstra Thanks to Contraction Hierarchies. URL: https://jeansebastien-gonsette.medium.com/routing-faster-than-dijkstra-thanks-to-contraction-hierarchies-232302345921 (дата звернення: 10.10.2025).

40. A* Search with Landmarks (ATL) Simulator. URL: https://tmdesigned.com/post/a-search-with-landmarks-atl-simulator (дата звернення: 10.10.2025).

41. How the routing OSRM algorithm works? URL: https://help.openstreetmap.org/questions/30272/how-the-routing-osrm-algorithm-works (дата звернення: 10.10.2025).

42. Project OSRM. URL: https://project-osrm.org/ (дата звернення: 10.10.2025).

43. Open source routing engine for OpenStreetMap. URL: https://github.com/graphhopper/graphhopper (дата звернення: 10.10.2025).

44. Шишковський С. О., Гуменюк Л. О. Методи та моделі розподілу та маршрутизації волонтерів під час катастроф. *Перспективні технології та прилади. Збірник статей*. Луцьк: ЛНТУ, 2025. Випуск 27. С. 84-89.