Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Створення засобів аналізу дорожнього руху на основі імітаційного моделювання

Текстова частина до кваліфікаційної роботи
за спеціальністю «Комп'ютерні науки» - 122

Керівник кваліфікаційної роботи
Старший викладач
Кундік К.В.

_____
(Підпис)
"___" _____ 2025 року

Виконала студентка
КН-4 Луцюк І. Р.
"___" _____ 2025 року

Київ 2025

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри інформатики, доцент, кандидат наук

_____ Гороховський С.С

(підпис)

“___”_____ 2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студентці Луцюк Іванні Русланівні 4-го курсу факультету інформатики

ТЕМА: Створення засобів аналізу дорожнього руху на основі імітаційного моделювання

Зміст ТЧ до кваліфікаційної роботи:

Індивідуальне завдання

Вступ

1. Аналіз предметної області
2. Огляд наявних інструментів симуляції
3. Розробка механізму для генерації сценаріїв та аналізу дорожнього руху

Висновок

Список використаних джерел

Додатки

Дата видачі “___” _____ 2025 р.

Керівник _____ Завдання отримано _____

Календарний План Виконання Кваліфікаційної Роботи

Тема : Створення засобів аналізу дорожнього руху на основі імітаційного моделювання

№ з/п	ПЕРЕЛІК РОБІТ	Термін Виконання	Примітка
1	Вибір теми	Вересень 2024	
2	Затвердження теми з керівником	Кінець вересня – Початок жовтня 2024	
3	Вивчення процесу симуляції дорожнього руху	Середина жовтня – Початок грудня 2024	
4	Дослідження симуляторів дорожнього руху	Початок грудня 2024 – Середина січня 2025	
5	Розробка підходу для генерації сценаріїв	Середина січня – початок березня 2025	
6	Створення методу для аналізу вихідних файлів	Початок квітня-кінець травня	
7	Створення пояснювальної роботи	Березень 2025	
8	Остаточне оформлення пояснювальної роботи та слайдів	Травень 2025	
9	Коригування роботи за результатами попереднього захисту	27.05.2025	

Студентка Луцюк І.Р _____

Керівник Кундік К.В _____

АНОТАЦІЯ

Мета кваліфікаційної роботи проаналізувати особливості процесу симуляції дорожнього руху, дослідити використання інструментів для моделювання транспортної системи та реалізувати підхід для автоматичної генерації та аналізу тестових сценаріїв на основі обраного симулятора, щоб знайти проблемні точки в транспортній інфраструктурі. Дана робота розкриває особливості роботи з моделюванням дорожнього руху та висвітлює метод створення сценаріїв поведінки транспортних засобів з використанням інструментів симулятора.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

UTM - Universal Transverse Mercator

ВСТУП

На сьогоднішній день мобільність стала вагомим фактором в оцінці якості життя. Кількість населення у сучасних прогресуючих містах невпинно зростає, наслідком чого є збільшення кількості транспортних засобів, що має не абиякий тиск на дорожній трафік, що є важливою частиною глобальної інфраструктури. Стрімкий зріст популяції та автотранспорту призводить до потреби оптимізації та розширення транспортної системи та грамотного управління потоками транспорту. До початку реалізації інфраструктурних проєктів: будівництва нових доріг, додавання смуг руху, світлофорів та розв’язок важливо мати реалістичну модель транспортних потоків, щоб пропоновані проєкти мали найкращі шанси на успіх у зниженні інтенсивності руху. У цьому контексті важливо використовувати інструменти оптимізації та аналізу транспортних систем, для прийняття оптимальних рішень в управлінні трафіком.

Здавалося б, що простим вирішенням поточних питань в інфраструктурі може бути розширення смуг або збільшення кількості доріг, але підвищення пропускної здатності за рахунок смуг або довжини доріг може не спрацювати так добре, як очікується. Саме цей факт і висвітлює «Парадокс Бреса» - який стверджує, що збільшення пропускної потужності мережі за умови, що суб'єкти руху самі обирають свій маршрут, може знизити загальну продуктивність [2]. Один шлях може бути більш популярний серед водіїв не лише завдяки якості покриття, але і завдяки меншій щільності потоку інших автомобілів. Якщо кожен водій обиратиме маршрут, який виглядає найсприятливішим для нього, то отриманий час перебування в дорозі не обов'язково буде мінімальним.

Згадані вище факти і створюють необхідність у застосування таких засобів як моделювання дорожнього руху. Імітаційні моделі допомагають зрозуміти, як різні заходи впливають на інтенсивність руху та транспортні потоки за різних обставин. Таким чином, моделювання дорожнього руху створює міцну основу для прийняття правильних і економічно ефективних рішень, роблячи дорожній рух і мобільність безпечними та стабільними.

Метою кваліфікаційної роботи є аналіз особливостей процесу симуляції дорожнього руху, дослідження використання інструментів для моделювання транспортної системи та реалізація підходу для автоматичної генерації та аналізу тестових сценаріїв на основі обраного симулятора.

Об'єктом роботи є процес симуляції дорожнього руху з використанням комп'ютерних моделей, а саме : запуск симуляції на основі згенерованих сценаріїв та аналіз вихідних файлів симуляції**.**

Предмет дослідження являє собою методи та засоби генерації сценаріїв дорожнього руху для використання в симуляційному середовищі та аналізу вихідних файлів з використанням інструментів та бібліотек обраного програмного пакету.

Дана робота розкриває особливості роботи з моделюванням дорожнього руху та висвітлює метод створення сценаріїв поведінки транспортних засобів з використанням інструментів симулятора та показує метод аналізу вихідних файлів симуляції.

Перший розділ присвячений ознайомленню з предметною областю : окреслення поняття імітаційне моделювання, які існують рівні деталізації моделювання та які існують методи моделювання.

Другий розділ присвячений дослідженню існуючих програмних продуктів призначених для симуляції дорожнього руху. Буде описано можливості найпопулярніших симуляторів дорожнього руху, їх можливості та інструменти. Також буде обґрунтовано вибір одного з них для реалізації генерації сценаріїв та аналізу результатів симуляції

Третій розділ включає ознайомлення з основними компонентами обраного симулятора та пояснення того, як вони задіяні в симуляції. Буде розроблено підхід для генерації сценаріїв для різної кількості транспортних засобів і буде проведено аналіз вихідних файлів симуляції для того аби виявити проблемні ділянки в транспортній інфраструктурі.

1. Аналіз предметної області

Попит на мобільність і легкодоступні засоби пересування постійно зростає. Кожен очікує безпечного, доступного, швидкого і комфортного транспорту. Тому перед інженерами стоїть завдання запропонувати надійні транспортні рішення, які є доступними, ефективними та справедливими. У транспортному плануванні та розробці сучасних систем мобільності вирішальну роль відіграє прогнозування транспортної поведінки та попиту на поїздки. Тільки якщо можна буде оцінити, як і де люди будуть подорожувати в найближчі роки, можна ухвалити правильні рішення для майбутньої системи мобільності. Моделювання та імітація транспортних потоків дозволяє інженерами зрозуміти поточні проблеми транспортної системи, визначити можливості, спрогнозувати та виміряти наслідки розвитку планування. Це слугує основою для прийняття обґрунтованих рішень і встановлення правильних рамок для майбутнього транспортної системи.

1.1 Імітаційне моделювання

Імітаційне моделювання – це метод дослідження, при якому досліджувана система замінюється моделлю, яка з достатньою точністю описує реальну систему, і з нею проводяться експерименти з метою отримання інформації про цю систему. Експериментування з моделлю називають імітацією [25]. Моделювання дорожнього руху це процес імітації руху транспортних засобів в транспортній мережі за допомогою спеціального програмного забезпечення. У транспортних дослідженнях моделювання дорожнього руху здатне представляти складні транспортні мережі, які зазвичай складаються з різноманітних сутностей, наприклад, різних типів транспортних засобів, пішоходів, велосипедистів, світлофорів, тощо, що взаємодіють за певними правилами та впливають на міські транспортні системи [26].

Моделювання транспортних систем базується на сценаріях у яких описуються умови мережі, кількість та поведінка транспортних засобів, тощо. Сценарій для

симуляції дорожнього руху – це повне налаштування для моделювання транспортного потоку руху в певному середовищі. Основними складовими сценаріїв є : налаштування дорожньої мережі, пішоходи та їхні точки призначення, транспортні засоби та заздалегідь визначені маршрути, налаштування світлофорів та додаткові параметри для моделювання, що не є обов'язковими для базових сценаріїв. Формально, сценарії визначають обставини для проведення експериментів з моделювання дорожнього руху. Більш детально складові сценаріїв будуть розглянуті на прикладі конкретного симулятора в розділі 3.

1.2 Рівні деталізації імітаційної моделі дорожнього руху

За весь час розвитку теорії транспортних потоків з'явилося безліч способів математичного опису процесів поведінки як потоків загалом, так і окремих автомобілів, їх взаємодії між собою і транспортною інфраструктурою. Багато методів набули широкого поширення на практиці для моделювання поведінки транспортних потоків з метою прогнозування й оцінки їх параметрів. Наявні методи моделювання використовують різний математичний апарат, ґрунтуються на різних припущеннях, мають різний ступінь деталізації і характеризуються різними функціональними властивостями, недоліками і перевагами. Транспортна модель залежно від масштабу завдань може охоплювати територію від цілого регіону до окремого перехрестя. Рівень деталізації та клас точності моделі повинні відповідати меті, для якої модель призначена [10]. Залежно від потрібного ступеня деталізації опису потоків і точності одержуваних параметрів вибирається відповідний клас моделі. Імітаційні моделі дорожнього руху поділяються на наступні рівні деталізації :

- Макроскопічні моделі. Основне положення макроскопічної моделі транспортного потоку полягає в тому, що рух транспорту може бути описаний як суцільний потік, подібно до рідини або газу. Макроскопічні моделі зосереджені на моделюванні транспортних потоків на основі

математичних моделей високого рівня. В цих моделях критеріями оцінки трафіку виступають густина транспортного потоку, його швидкість та інтенсивність, тому у цих умовах модель поведінки всіх водіїв однакова [11]. Даний тип моделювання може бути використаний для аналізу систем з великою площею, в яких не потрібне детальне моделювання, наприклад, для моделювання руху на автомагістралях. Макроскопічні моделі описують найважливіші властивості транспортних потоків, а саме : утворення та розсіювання черг, часи пік, тощо. Макроскопічні моделі менш вимогливі до обчислювальних ресурсів. Крім того, обчислювальна потреба не зростає зі збільшенням щільності трафіку, тобто не залежить від кількості транспортних засобів у мережі. Вони дозволяють визначати середній час подорожі, середню витрату палива та викиди залежно від дій транспортних потоків [12].

- Мікроскопічні моделі. Мікроскопічне моделювання зосереджується на моделюванні окремих сутностей на основі високого рівня деталізації здебільшого припускаючи, що поведінка транспортного засобу залежить як від фізичних можливостей транспортного засобу рухатися, так і від керуючої поведінки водія. Даний тип деталізації можна визначити як опис рухів окремих транспортних засобів, котрі розглядаються як результат впливу певних факторів, а саме : характеристики водіїв та транспортних засобів, взаємодії між водієм і автомобілем, якість дорожнього покриття, умови навколишнього середовища та засоби регулювання руху [13]. Цей підхід розглядає рух автомобілів як рух окремих об’єктів і є найбільш ефективним в описі руху на невеликих відстанях (наприклад, на перехрестях). Перевагою макроскопічних моделей зазвичай є їхня висока швидкість виконання. Однак детальне моделювання мікроскопічних моделей є більш точним, особливо коли потрібно моделювати окремі маршрути. На відміну від макроскопічних моделей, у мікроскопічних моделях кожен транспортний засіб характеризується власною швидкістю та індивідуальними обмеженнями [11]. Прикладами імітації дорожнього

руху такого типу деталізації можуть бути моделювання заторів на перехрестях, поводження водіїв на швидкісних дорогах (обгін, дотримання дистанції, тощо), симуляції поведінки водіїв при ДТП, рух транспортних засобів при поганих погодних умовах. Крім того, мікроскопічні симуляції мають так званий підтип – субмікроскопічний рівень деталізації. Ці типи розширюють мікроскопічні моделі транспортних засобів, наприклад, за значенням повороту (також званим курсом) і, таким чином, за поперечним положенням транспортного засобу. На противагу цьому, мікроскопічна симуляція дорожнього руху в основному моделює поздовжню поведінку транспортних засобів вздовж смуги руху в поєднанні з рішеннями про зміну смуги руху. Субмікроскопічні моделі розглядають окремі транспортні засоби, так само як і мікроскопічні моделі, але розширюють їх розділяючи на додаткові підструктури, які описують швидкість обертання двигуна залежно від швидкості транспортного засобу або бажані дії водія щодо перемикання передач. Це дозволяє проводити більш детальні обчислення порівняно з простими мікроскопічними симуляціями. Однак субмікроскопічні моделі вимагають більшого часу обчислення, що може обмежувати розмір мереж, що моделюються.

- Мезоскопічні моделі. Мезоскопічні симуляції являють собою поєднання рис макроскопічних та мікроскопічних симуляційних моделей, як показано на рисунку 1. Вони заповнюють прогалину між підходом макроскопічних моделей на агрегованому рівні та індивідуальними взаємодіями мікроскопічних моделей. Мезоскопічні моделі зазвичай описують дорожні об'єкти з високим рівнем деталізації, як у мікромоделювання. Однак, їхня поведінка та взаємодія описуються з нижчим рівнем деталізації, як у макромоделюванні [11]. Транспортний потік розглядається як набір декількох груп сутностей, дії яких і взаємодія між якими розглядається з високим ступенем деталізації. Наразі дослідниками розроблено різноманітні форми мезоскопічніх моделей [14]. Транспортні засоби можуть об’єднувати в групи, що маршрутизуються за транспортною

мережею. Група автомобілів діє як єдине ціле і швидкість групи на кожній дорожній ділянці встановлюється функцією швидкості, визначеної для цієї групи. Густина цієї групи задається як відношення кількості автомобілів у групі до кількості кілометрів на одну смугу руху. Якщо густина групи на лінії висока, то швидкість автомобілів буде низька. Якщо густина групи низька, то швидкість групи висока. У цій формі мезоскопічної моделі не підтримуються зміна смуг руху, прискорення і гальмування. Інша форма мезоскопічної моделі: індивідуальні транспортні засоби об'єднані в осередки, які встановлюють поведінку цих транспортних засобів. Автомобілі можуть входити в осередки і полишати осередки, але не можуть їх обганяти. Швидкість автомобілів установлюється осередком, а не за індивідуальним рішенням водія. Мезомоделювання дає змогу моделювати дорожню мережу і рух автомобілів майже з таким же рівнем деталізації, як і мікромоделювання. Мезоскопічне моделювання застосовується там, де бажано використати мікроскопічні моделі, але не можна реалізувати через великий розмір транспортної мережі.

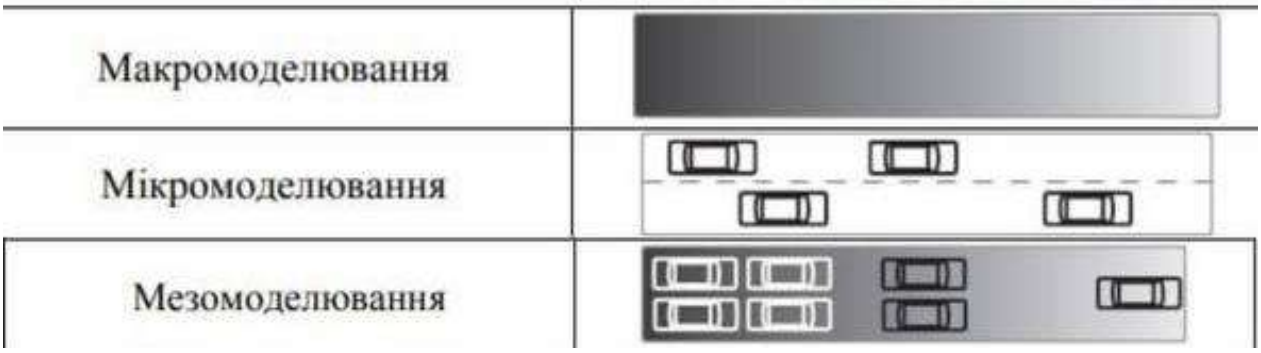


Рис. 1 Мезоскопічні моделі як поєднання рис макроскопічних та мікроскопічних моделей

Рівень деталізації визначає, які аспекти транспортної системи охоплюються. Такі відмінності також відображаються на даних, необхідних для моделювання. Використання реальних даних має підвищити реалістичність і точність моделювання. Однак дослідники повинні усвідомлювати мету свого моделювання і вибирати імітаційну модель, яка підтримує необхідний рівень

деталізації для вирішення поставлених дослідницьких завдань. Заглиблення в деталізацію може зробити імітаційну модель складнішою, а також вимагатиме більше вхідних даних.

1.3 Методи транспортного моделювання

Залежно від необхідного рівня деталізації та точності моделі, прогнозованого періоду, доступних вхідних даних та ресурсів можна використовувати різні математичні підходи. Так склалось історично, що найчастіше використовувалася агрегована методологія, яка називається чотирикроковою моделлю або моделлю на основі поїздок . Останнім часом у багатьох місцях впроваджено більш детальні дезагреговані підходи, які називаються моделями на основі діяльності та моделями на основі агентів. Ці моделі носять назви activity-based models та - agent-based models – ABM.

1.3.1 Four-step travel model

В рамках раціонального планування транспортні прогнози традиційно складаються за послідовною чотириступеневою моделлю або процедурою планування міського транспорту. Процес починається з прогнозування землекористування. Як правило, прогнози робляться для регіону в цілому, наприклад, щодо зростання населення. Такі прогнози надають контрольні підсумки для аналізу місцевого землекористування. Зазвичай, регіон поділяється на зони і за допомогою аналізу тенденцій або регресійного аналізу визначається чисельність населення та зайнятість для кожної з них . Чотири кроки класичної моделі планування міської транспортної системи: [16]:

- Перший крок - генерація поїздок. Генерація поїздок визначає частоту місць відправлення або призначення поїздок у певній зоні аналізу подорожей. Транспортно-аналізовані зони поїздок кодуються даними про землекористування, такими як кількість домогосподарств та зайнятість, для розуміння попиту на подорожі. Згенеровані поїздки пов'язані з різними

цілями поїздок, наприклад, робота, покупки або відпочинок. Виробництво та привабливість поїздок визначаються так званими показниками поїздок, середніми показниками, що базуються на кількості людей у домогосподарствах або кількості доступних транспортних засобів [9].

- Другий крок – розподіл поїздок. Крок розподілу поїздок зіставляє пункти відправлення з пунктом призначення. Це робиться шляхом зважування популярності потенційного пункту призначення та зусиль, необхідних для його досягнення, таких як відстань дороги, час подорожі та плата за проїзд/вартість. В результаті початковий попит на транспортно-аналізовані зонами розподіляється між кількома зонами призначення. Залежно від сегментації моделі, можуть бути згенеровані кілька матриць розподілу, наприклад, за метою поїздки або доходом домогосподарства [9]. Другий крок дає розуміння того, як переміщується населення протягом дня, як поїздки розподіляються по території, які напрямки мають найбільший попит та дозволяє побачити повну картину потоків між районами міста – це є основа для прийняття рішень у транспортному плануванні
- Третій крок – вибір виду транспорту. На третьому кроці поїздки між транспортно-аналізованими зонами розподіляються між різними видами транспорту. Вибір конкретного типу транспортного засобу залежить від індивідуальних характеристик, умов життя домогосподарства, наприклад, наявність та кількість одиниць транспортного засобу, так і від особистих вподобань у виборі способу пересування. Інші фактори, такі як час подорожі, вартість, наявність паркування та кількість пересадок на громадський транспорт, мають додатковий вплив на розподіл попиту за видами транспорту. Ці змінні та параметри зазвичай включаються в дискретні вибіркові моделі, зокрема в логістичну модель, для розрахунку та оцінки розподілу попиту між видами транспорту. У результаті, матриці які були отримані на етапі розподілу поїздок додатково уточнюються в матриці поїздок для кожного виду транспорту [9].

- Четвертий крок – призначення поїздки. Етап призначення маршруту розподіляє поїздки між пунктом відправлення та пунктом призначення, розподіл здійснюється окремо для кожного виду транспорту. Матриці поїздок з попередніх кроків використовуються як вхідні дані для призначення маршрутних потоків фактичній транспортній мережі. Для дорожнього руху легкових автомобілів та великовантажних транспортних засобів, який обмежений пропускною спроможністю доріг, застосовуються ітераційні рівноважні процедури розподілу мережі. Розподіл руху на різні маршрути в цих процедурах базується на спостереженні, що фактична швидкість руху на дорогах зменшується зі збільшенням інтенсивності руху по відношенню до пропускної здатності, тобто рівня насиченості. Зі збільшенням транспортного навантаження та зменшенням швидкості руху на основних дорогах, учасники дорожнього руху переходять на другорядні, швидші маршрути. Процедури розподілу ітеративно перерозподіляють частки попиту на перевезення між різними маршрутами, поки всі маршрути, призначені для кожної пари зон відправлення та призначення, не матимуть однаковий або дуже схожий час у дорозі. Таке балансування відбувається для всіх пар одночасно, що призводить до стану рівноваги в масштабах всієї мережі, який називається рівновагою Уордропа. Процес призначення маршрутів громадського транспорту працює інакше, ніж на автомобільних дорогах. Мережа громадського транспорту складається з окремих маршрутів з певною частотою руху та можливим часом очікування на зупинках. Вхід до мережі громадського транспорту - це момент, коли пасажир починає користуватися громадським транспортом, потрапляючи на одну з офіційних зупинок, де можна сісти на трамвай, автобус, метро, тощо. Вхід до мережі громадського транспорту можливий лише на певних зупинках, тому вхід і вихід - зазвичай пішки - є обов'язковими. Для певної пари пунктів відправлення та призначення може не існувати прямого сполучення, тому може знадобитися одна або кілька пересадок. Крім того, при виборі маршруту також враховуються транзитні

тарифи. Існують різні фактори, які впливають на досвід подорожі, такі як час у дорозі, кількість пересадок, час очікування або час входу та виходу. В рамках завдання громадського транспорту ці фактори враховуються в моделі вибору. На основі мережі громадського транспорту та розкладу руху визначаються різні варіанти пересадок. Потім поїздки розподіляються між цими альтернативами на основі вподобань пасажирів та отриманої мережі громадського транспорту, ліній, зупинок, кількості посадок та обсягів перевезень, доступних для аналізу. Хоча активні види транспорту, такі як велосипед, часто користуються тією ж дорожньою інфраструктурою, що й автомобілі, на вибір маршруту велосипедистами впливають інші фактори, ніж водіями. Швидкість руху велосипедів не залежить від ефекту пропускної здатності, як швидкість руху автомобілів. Натомість, велосипедисти більш чутливі до таких характеристик, як нахили, покриття, світлофори, смуги тощо. Тому моделі вибору, що відображають ці аспекти, застосовуються для розподілу активних режимів руху на різні альтернативи маршруту [9].

Після виконання чотирьох кроків класичної моделі відбувається оцінка відповідно до узгодженого набору критеріїв і параметрів рішення. Типовим критерієм є аналіз витрат і вигод. Такий аналіз може бути застосований після того, як модель мережевого розподілу визначить необхідну пропускну спроможність, тобто чи варта така пропускна спроможність того, щоб її створювати. На додаток до визначення етапів прогнозування та прийняття рішень як додаткових етапів процесу, важливо зазначити, що прогнозування та прийняття рішень відбуваються протягом кожного етапу процесу планування міського транспорту.

1.3.2 Activity-based model and agent-based model

Терміни «агентно-орієнтований» (agent-based) та «діяльнісний» (activity-based) часто використовуються як взаємозамінні, проте вони охоплюють різні концепції. Агентно-орієнтоване моделювання використовується в широкому спектрі галузей, однією з яких є транспортне моделювання. Моделювання попиту на основі діяльності використовується виключно в моделюванні поведінки подорожуючих. Це різні методи моделювання, які іноді використовуються разом [17].

Методи на основі агентів моделюють рішення, пов'язані з подорожами, на рівні окремої особи. Агент - основна одиниця в агентній моделі, може бути представленням будь-якого типу автономної сутності, яка автономно працює в середовищі, переслідуючи свою мету або цілі. Агентом може бути людина, транспортний засіб або будь-який інший об'єкт, який активно взаємодіє з іншими агентами, якщо це необхідно [18]. Цей метод має на меті відтворити поведінку окремих осіб , коли вони взаємодіють з іншими учасниками дорожнього руху та навколишнім середовищем, а також те, як це проявляється в ширшій транспортній системі. Агент-орієнтовані методи широко використовуються в інших сферах для розуміння складних системних взаємодій, наприклад, в економіці, фізиці та біології. Через свою більш детальну природу вони можуть вимагати більш складних досліджень або даних. Агентні методи вже широко використовуються в транспортному моделюванні. Найпоширенішими прикладами агент-орієнтованих методів у транспортному моделюванні є :

- пішохідне моделювання
- мікросимуляції дорожнього руху
- моделювання динамічних призначень, враховуючи умови на дорогах

Методи на основі агентів дають можливість зрозуміти, як може змінюватися сукупна поведінка системи, коли поведінка користувачів змінюється у відповідь на впливи, такі як зміна графіку роботи, плата за проїзд або нові технології [17].

Моделі на основі діяльності (activity-based model) - це клас моделей, які передбачають, де і коли люди здійснюють певні види діяльності та більшою мірою зосереджуються на діяльності індивідів, а не на поїздках. І вже на основі активності можна змоделювати транспортну поведінку. Основною передумовою моделей, заснованих на діяльності, є те, що попит на подорожі є похідним від діяльності, яку люди повинні або бажають здійснювати, а рішення щодо подорожей є частиною рішень щодо планування. Таким чином, подорожі розглядаються лише як один з атрибутів системи [16]. Підходи до моделювання попиту на основі активності вважають, що попит на транспортні послуги випливає з потреб і бажань людей брати участь у певних видах занять. Вони використовують поведінкові теорії про те, як люди приймають рішення про участь у діяльності за наявності обмежень. Це включає рішення про те, коли і де брати участь у заходах, а також як дістатися до цих заходів. Моделі попиту на основі видів діяльності можуть реалістично відображати часові та просторові обмеження, показувати зв'язки між видами діяльності та подорожами для кожної особи, а також для кількох осіб у домогосподарстві (сім'ї). Це повинно дозволити моделям краще відображати вплив умов пересування на діяльність людей, а отже, і на вибір подорожей [9].

Гнучкість агентно-орієнтованих методів та моделювання попиту на основі діяльності дозволяє використовувати широкий спектр реалізацій та адаптувати моделі до конкретних випадків використання.

2. Інструменти імітаційного моделювання дорожнього руху

2.1 SUMO (Simulation of Urban MObility)

SUMO (Simulation of Urban MObility) – це безкоштовний програмний пакет з відкритим вихідним кодом. SUMO дозволяє моделювати інтермодальну мікроскопічну симуляцію, включаючи дорожні транспортні засоби, громадський

транспорт і пішоходів. До складу програмного пакету входить безліч допоміжних інструментів, а саме: sumo(command line), sumo-gui, netconvert, netedit, netgenerate, od2trips, duarouter, jtrrouter, dfrouter, marouter, polyconvert, activitygen, emissionsMap, emissionsDrivingCycle, osmWebWizard . Вони допомагають автоматизувати основні завдання зі створення, виконання та оцінки симуляцій дорожнього руху, такі як імпорт мереж, розрахунки маршрутів, візуалізація та розрахунок викидів. SUMO може бути розширений за допомогою користувацьких моделей і надає API для дистанційного керування симуляцією. Даний симулятор можна використовувати на різних операційних системах, таких як Windows, Linux або macOs, оскільки він реалізований на C++ та Python з застосуванням портативних бібліотек. API Traffic Control Interface (TraCI), що надає доступ до запущеної симуляції дорожнього руху дозволяє отримувати значення змодельованих об'єктів і керувати їхньою поведінкою у режимі реального часу. SUMO не встановлює жорстких обмежень на розмір мережі, тривалість симуляції та кількість транспортних засобів, які можна використовувати в моделюванні сценаріїв [1]. Сценарій в симуляторі SUMO являє собою конфігураційний файл, який складається з основних компонентів симуляції, а саме : файли з налаштуванням мережі та побудовані маршрути для транспортних засобів, що формуються на основі файлів XML формату.

2.2 MATSim (Multi-Agent Transport Simulation)

MATSim (Multi-Agent Transport Simulation) – це інтермодальна агент-орієнтована мезоскопічна система з можливістю розширення, що заснована на основі діяльності агентів. Симулятор реалізований на Java та має відкритий вихідний код що доступний для завантаження на веб-сайті MATSim або на GitHub. Розроблений для великомасштабних сценаріїв, фреймворк надає пріоритет обчислювальній ефективності шляхом спрощення функцій моделі, де це можливо, з підтримкою паралельних обчислень. Симулятор працює за

коеволюційним принципом, де агенти ітеративно оптимізують свої щоденні графіки, конкуруючи за часово-просторові ресурси в транспортній мережі. Також існує поняття, так званий цикл MATSim [4] . Процес моделювання складається з багатьох ітерацій, як показано на рисунку 2, починаючи з початкового попиту, отриманого з ланцюгів активності населення. Ці ланцюги, як правило, генеруються на основі емпіричних даних за допомогою вибірки або моделей дискретного вибору. Кожен агент зберігає в пам'яті дані про можливі щоденні плани , кожен з яких має відповідну оцінку (інтерпретується як корисність). Перед кожним етапом симуляції агенти обирають план на основі цих оцінок і частина агентів, зазвичай це приблизно 10% може змінювати обраний план шляхом коригування та клонування. Після кожної ітерації агенти оцінюють свої плани, тобто підраховують бали, і плани з низькими балами відкидаються. Процес повторюється до тих пір, поки середня оцінка популяції не стабілізується, нагадуючи еволюційну оптимізацію.

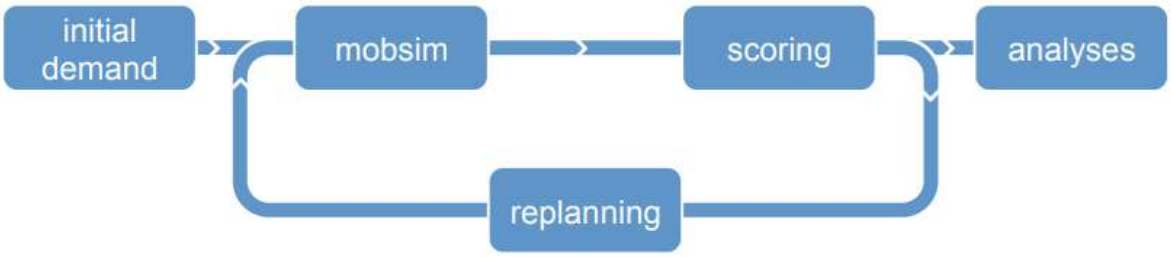


Рис 2. Схема роботи циклу MATSim

Для створення та обробки вхідних або вихідних файлі існує сторонній інструмент - MATSim Python API, який призначений для прямого використання бібліотеки даних pandas або для сумісної роботи з нею. Наразі ця бібліотека перебуває на ранніх стадіях розробки і підтримуються лише файли мережі, подій та планів[3].

2.3 PTV Group

PTV Planung Transport Verkehr GmbH - німецька компанія котра спеціалізується на розробці програмного забезпечення для планування, моделювання і управління мобільністю та працює в галузі консалтингових послуг у сфері дорожнього руху, транспорту та мобільності [5]. До основних продуктів компанії PTV Group входять:

- PTV Visum – програмне забезпечення призначене для макроскопічних симуляцій і макроскопічного моделювання транспортних мереж.
- PTV Vissim – програмне забезпечення розроблене для моделювання руху у мікроскопічному масштабі
- PTV Vissim Automotive – програмне забезпечення надає можливість створювати реалістичний, реактивний і заснований на моделях поведінки навколишній рух, що дозволяє проводити реалістичні та високоточні віртуальні випробування в замкнутому циклі в рамках великих дорожніх сценаріїв.
- PTV Viswalk – програмне забезпечення для мікроскопічного моделювання моделювання пішохідного руху, зображує пішоходів та їхню взаємодію один з одним.
- PTV Vistro – універсальний інструмент для аналізу дорожнього руху, оптимізації роботи світлофорів і оцінки впливу нових забудов на транспортну інфраструктуру міста.
- PTV Optima – програмне забезпечення для управління трафіком в режимі реального часу для мультимодальних мереж. Надає детальні прогнози трафіку та моніторингові панелі, що дозволяє операторам визначати найкращі сценарії для управління заторами, перекриттям доріг і будівельними майданчиками, а також точно моделювати результати роботи мережі.

Компанія PTV Group орієнтована на комерційну співпрацю, але деякі програмні продукти доступні в академічній версії. PTV Academia пропонує студентам, викладачам та дослідникам широкий спектр можливостей для підтримки їхніх досліджень у сфері дорожнього руху. PTV Academia надає доступ до декількох типів ліцензій, що мають гнучкі умови використання, наприклад, шестимісячний безкоштовний період з можливістю експлуатації продуктів для симуляції дорожнього руху, мережевого моделювання та інструментів оцінки трафіку.

2.4 CityFlow

CityFlow – високопродуктивний симулятор дорожнього руху, що має відкритий вихідний код, націлений на мікроскопічне моделювання міського руху. Він має ретельно розроблену структуру даних та використовує алгоритми моделювання з багатопотоковістю, що дозволяє ефективно моделювати міський трафік [7]. Щоб підтримувати високу продуктивність ядро було розроблене на мові програмування C++ та має Python API для взаємодії з користувачем. Мультиагентність CityFlow має перевагу над Sumo. Як показано на Рисунку 3, CityFlow перевершує SUMO у всіх сценаріях від малого до більш масштабного трафіку з одним потоком, а особливо помітним стає різниця при використанні багатопоточності. CityFlow досягає приблизно 25-кратного прискорення на великомасштабних дорожніх мережах 30×30 з десятками тисяч транспортних засобів, використовуючи 8 потоків, що становить 72 кроки моделювання в секунду. Крім того, CityFlow показує кращу ефективність при отриманні інформації про симуляцію через інтерфейс python. Це пов'язано з тим, що SUMO використовує сокет для взаємодії, в той час як CityFlow використовує pybind11 для безперебійної інтеграції C++ та python [8].

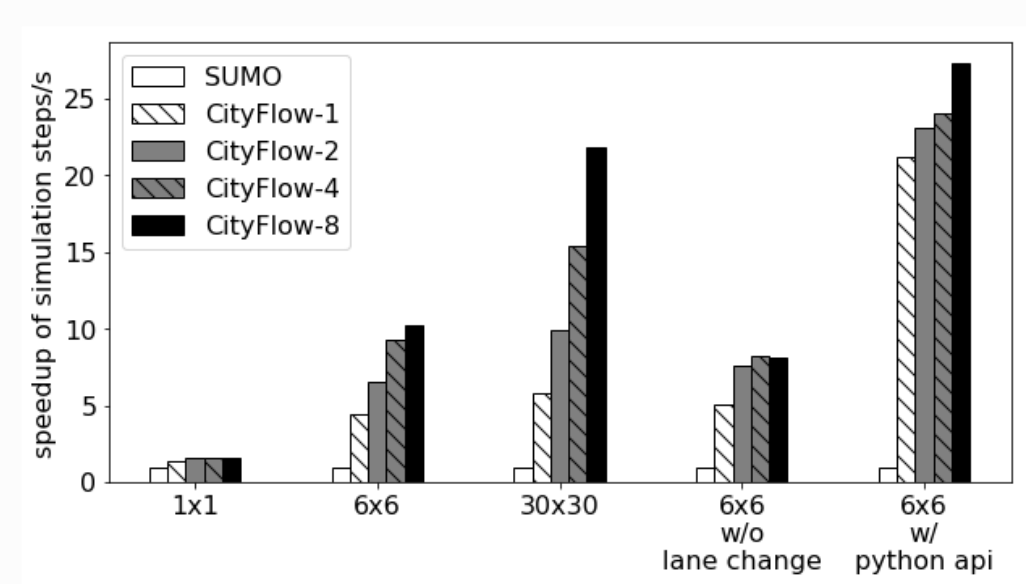


Рис 3. Порівняння продуктивності CityFlow з різною кількістю потоків (1, 2, 4, 8) та SUMO

2.5 Обґрунтування вибору симулятора для автоматизованої генерації сценаріїв

У якості симулятора для реалізації динамічного створення сценаріїв та аналізу вихідних файлів було обрано симулятор SUMO. Даний симулятор є повністю безкоштовним і не має ніяких ліцензійних обмежень, що являється великою перевагою для академічної дослідницької роботи. З SUMO можна взаємодіяти як з консолі, використовуючи різні команди, так і через графічний інтерфейс, спостерігаючи за процесом симуляції в режимі реального часу, sumo-gui є доволі простим та інтуїтивно зрозумілім. У порівнянні з іншими розглянутими симуляторами в SUMO доволі просто будувати сценарії, оскільки вони побудовані на основі файлів XML і використовуючи інструменти симулятора можна сформувати максимально реалістичні сценарії починаючи від загального потоку, закінчуючи манерою водіння водія. Що не менш важливо для подальшої роботи з симулятором це підтримка інструментів для реального середовища. SUMO підтримує реальні мапи які можуть бути імпортовані з OpenStreetMap. Використання SUMO в кваліфікаційній роботі дозволить реалізувати динамічне

створення сценаріїв для різної кількості авто, збирати та аналізувати вихідні дані симуляції для отримання інформації про проблемні ділянки та дасть змогу показати як збільшення кількості транспортних засобів впливає на транспортну мережу.

3. Розробка підходу для автоматичної генерації сценаріїв

Перед тим як перейти до реалізації автоматичної генерації сценаріїв потрібно оглянути з яких компонентів вони складаються. Як було зазначено в попередньому розділі, обраний симулятор для реалізації – SUMO, тому далі у підрозділах буде проаналізовано основні компоненти сценаріїв для даного симулятора, зокрема файли мережі, маршрутів, додаткових налаштувань, файл конфігурації та вихідні файли.

3.1 Основні компоненти системи

Побудова базового сценарію SUMO – це багатоетапна діяльність, що включає наступні етапи : підготовка транспортної мережі, визначення трафіку, встановлення алгоритму маршрутизації та запуск симуляції.

3.1.1 Мережа

Дорожні мережі в SUMO закодовані у вигляді XML-файлів, а їхній вміст згруповано за типами у наступному подяку [19]:

- Картографічна проекція, дійсна для даної мережі
- Ребра, котрі поділені на внутрішні та прості ребра і представлені відповідно. Кожне ребро також складається зі смуг руху
- Логіка світлофорів
- Перехрестя
- Зв’язки між смугами руху

- Кільцеві перехрестя (опціонально)

Далі буде детально розглянуто кожен компонент мережі.

- **Координати та вирівнювання.** Мережі SUMO завжди кодуються в декартових координатах (метрах) і можуть містити інформацію про географічну прив'язку, що дозволяє конвертувати їх у довготу та широту. За замовчуванням для декартових координат використовується UTM-проекція зі зміщенням початку координат так, щоб нижній лівий кут мережі знаходився на рівні 0,0 [20]. Інформація про проекцію закодована в елементі <location> у верхній частині файлу *.net.xml та описує просторову інформацію про дорожню мережу, тобто, як координати з географічної системи реального світу представлені у координатах в системі координат SUMO. *NetOffset* описує зсув, застосований для переміщення мережі до (0,0). *ConvBoundary* описує границю, що проходить через вузли перетвореної мережі, тобто координати меж мережі після перетворення, представлені в системі координат SUMO. *OrigBoundary* описує початкові межі мережі перед проектуванням. А ProjParameter представляє інформацію про те, як була спроектована мережа, що включає тип картографічної проекції, номер зони в системі координат, модель форми землі, систему координат та одиниці виміру [19]. Приклад елементу <location> можна побачити на лістингу 1, що представляє просторову інформацію про мережу : тип картографічної проекції – UTM, 36 зона, що охоплює Київ, Україну (30°–36° сх. довготи), систему WGS84 та одиниці виміру – метри.

```
<location netOffset="-318940.34,-5582537.55"
convBoundary="0.00,0.00,5417.00,5828.85"
origBoundary="30.454007,50.367019,30.528964,50.420894"
projParameter="+proj=utm +zone=36 +ellps=WGS84 +datum=WGS84 +units=m +no_defs"/>
```

Лістинг 1 Приклад елементу <location> в файлі *.net.xml

- **Ребра та смуги.** Кожне ребро edge, що представляє дорогу, містить визначення смуг руху lanes, з яких воно складається. У лістингу 2 показано одне ребро з двома смугами руху, одна з яких двовимірна, а інша тривимірна. Смуга руху включає в себе такі атрибути: ідентифікатор смуги *id*, порядковий номер *index*, швидкість *speed*, довжину смуги *length* та геометрію смуги *shape*. Ідентифікатор смуги є комплексним і складається з ідентифікатора ребра, до якого належить смуга, і порядкового номера, розділених символом підкреслення ('_') [20]. Порядковий номер починається з нуля для крайньої правої смуги. Цей же номер також вказано в атрибуті index. Максимально дозволена швидкість зазвичай вимірюється в метрах за секунду, відповідно довжина дороги визначається в метрах . Атрибут shape визначає траєкторію смуги руху в просторі і являє собою послідовність точок координат, котрі описують як саме смуга проходить на карті.

```
<edge id="<ID>" from="<FROM_NODE_ID>" to="<TO_NODE_ID>" priority="<PRIORITY>">
    <lane id="<ID>_0" index="0" speed="<SPEED>" length="<LENGTH>" shape="0.00,495.05 248.50,495.05"/>
    <lane id="<ID>_1" index="1" speed="<SPEED>" length="<LENGTH>" shape="0.00,498.35,2.00 248.50,498.35,3.00"/>
</edge>
```

Лістинг 2 Приклад одного ребра з двома смугами руху

У SUMO є два типи ребер – normal та internal edges, котрі майже однакові в кодуванні за винятком деяких атрибутів. Звичайні ребра являють собою з’єднання між двома вузлами та представляють типові ділянки дороги з реального світу. Ідентифікатор *ID* ребра зчитується під час імпорту мережі. Ідентифікатори початкового та кінцевого вузлів вказані в атрибутах *from* та *to* відповідно. Пріоритет *priority* – це абстрактне порядкове число, яке визначає правила смуги відведення. Атрибут *function* у наведеному лістингу 3 було пропущено, оскільки за замовчуванням він має значення *"normal"*, що є значенням функції прикладу ребра. Для симуляції цікавий лише атрибут *function*. Він описує, як використовується ребро, і чи це ребро, яке можна знайти в реальному світі, чи це лише допоміжна конструкція, що використовується для призначення. Атрибут *function* може

мати такі типи : *"normal"* – визначає просту частину дорожньої мережі, *"connector"* – визначає з'єднувальні ділянки, які не є частиною реальної мережі, а використовуються для технічних цілей, *"internal"* – допоміжні ребра в середині перехресть, *"crossing"* – спеціальне ребро яке представляє пішохідний перехід через смуги руху автомобілів, а *"walkingarea"* – смуга для вільного руху пішоходів.

```
<edge id="<ID>" from="<FROM_NODE_ID>" to="<TO_NODE_ID>" priority="<PRIORITY>">
    ... one or more lanes ...
</edge>
```

Лістинг 3 Приклад звичайного ребра

На лістингу 4 наведено приклад застосування атрибуту *function*, що визначено як *"internal",* що означає воно є внутрішнім. Внутрішні ребра – це допоміжні ребра, які існують лише в межах перехресть і моделюють рух транспортних засобів тільки всередині них. Внутрішнє ребро лежить всередині перехрестя і з'єднує вхідне нормальне ребро з вихідним нормальним ребром.

```
<edge id="<ID>" function="internal">
    ... one lane ...
</edge>
```

Лістинг 4 Приклад внутрішнього ребра

На лістингу номер 5 можна побачити як формуються внутрішнє ребро з кількома смугами руху. Ідентифікатор внутрішнього ребра завжди починається з символу ':' та складається з <NODE_ID>_<EDGE_INDEX>, де <NODE_ID> - це ідентифікатор вузла, в якому знаходиться ребро, а <EDGE_INDEX> - це порядковий номер ребра за годинниковою стрілкою навколо вузла. Якщо вхідні та вихідні ребра, з'єднані внутрішнім ребром, мають декілька смуг, то внутрішнє ребро також має декілька смуг. Коли мережа побудована з внутрішніми ребрами, кожному з'єднанню зазвичай відповідає рівно одна внутрішня смуга. Якщо є кілька з'єднань, які мають однакову пару ребер «від» і «до», то внутрішні смуги для цих ребер будуть частиною одного внутрішнього ребра [19].

```
<edge id=":<NODE_ID>_<EDGE_INDEX>" function="internal">
    <lane id=":<EDGE_ID>_<INDEX>" index="INDEX" speed=".." length=".." shape=".." />
    <lane id=":<EDGE_ID>_<INDEX>" index="INDEX" speed=".." length=".." shape=".." />
</edge>
```

Лістинг 5 Приклад внутрішнього ребра з кількома ребрами

- **Світлофорні програми.** У SUMO є окремий компонент <*tlLogic*> який визначає порядок і тривалість сигналів на перехресті. Він може входити до як до файлу мережі *.net.xml так і до окремого файлу з налаштуваннями additional-file, який додається до конфігураційного файлу. Світлофорна програма складається з набору фаз і за замовчування програми генеруються з чотирма зеленими фазами : пряма фаза, фаза поперечного напрямку, фаза лівого повороту, фаза лівого повороту для поперечного напрямку, за умови що є виділена смуга для лівого повороту [21]. Компоненти світлофорної програми можуть бути з фіксованою тривалість фаз *static*, подовження фази на основі проміжків часу між транспортними засобами *actuated* або на основі накопиченої втрати часу транспортними засобами в черзі *delay_based*. Фази світлофорів мають свою тривалість *duration* та стан *state.* Стан являє собою комбінацію символів, де кожен символ відповідає окремим сигналом для певної смуги руху. G – дозвіл руху з пріоритетом, g – дозвіл руху без пріоритету, y – попередження про зміну кольору світлофору, r – проїзд заборонено. На лістингу 6 показано приклад світлофорної програми, яка складається з трьох фаз, а у кожній фазі по чотири смуги руху. Фаза зеленого кольору, що дозволяє рух триває 79 секунд, жовта фаза триває 6 секунд і червона фаза триває 5 секунд.

```
<tlLogic id="0101" type="static" programID="0" offset="0">
    <phase duration="79" state="GGG"/>
    <phase duration="6"  state="yyy"/>
    <phase duration="5"  state="rrr"/>
</tlLogic>
```

Лістинг 6 Приклад компоненту світлофорної програми

- **Перехрестя.** Перехрестя представляють собою зону перетину різних потоків, включаючи правила руху транспортних засобів, яких вони повинні дотримуватися під час перетину перехрестя. Приклад компоненту перехрестя <*junction*> представлено на лістингу 7. За замовчуванням , ідентифікатор перехрестя *id* матиме той самий ідентифікатор що і світлофор, що контролює це перехрестя. Кожне перехрестя має свій атрибут типу *type,* котрий визначає тип пріоритету : priority (звичайний тип пріоритету), right_before_left (поступитися автомобілям праворуч), dead_end (глухий кут), traffic_light (керується світлофором) і тому подібні Атрибути *x, y, z* відповідають за координати перехрестя, а *incLanes* та *intLanes* визначають, відповідно, ідентифікатори смуг руху, які закінчуються на перехресті, та смуг, розташованих в межах перехрестя.

```
<junction id="<ID>" type="<JUNCTION_TYPE>" x="<X-POSITION>" y="<Y-POSITION>"
          incLanes="<INCOMING_LANES>" intLanes="<INTERNAL_LANES>"
          shape="<SHAPE>">
    ... requests ...
</junction>
```

Лістинг 7 Приклад компоненту перехрестя

Компонент перехрестя включає в себе пріоритетні запити, які окреслюють правила пріоритетності руху для кожного з’єднання і описуються компонентом <*request*>. Даний елемент відповідає за визначення того, який саме потік має вищий пріоритет, з якими потоками утворюється конфлікт та чи може транспортний засіб заїхати в зону перехрестя і очікувати проїзду там. Основними атрибутами елементу request є : індекс зв’язку *index* для якого визначаються правила, бітовий рядок *response,* у якому 1 показує чи повинен транспортний засіб гальмувати чи може проїхати без зупинки якщо 0, розмір рядка відповідає кількості смуг на дорозі, аналогічно у бітовому рядок *foes* 1 вказує на те, що зв1язок є конфліктним, а 0, якщо конфлікту немає і булеве значення *const* позначає чи можна транспортному засобу виїжджати на перехрестя і очікувати всередині нього [19].

```
<request index="<INDEX>" response="<RELATIVE_MAJOR_LINKS>" foes="<FOE_LINKS>" cont="<MAY_ENTER>"/>
```

Лістинг 8 Структура компоненту запитів пріоритетності

- Зв’язки. Елемент <*connectin*> описує як автомобілі пересуваються між смугами на перехресті, тобто на які виїзні смуги можна потрапити з в'їзної смуги. В зв’язках обов’язково вказуються основні атрибути : ідентифікатор *from* вхідного ребра, на якому перебуває автомобіль, ідентифікатор *to* вихідного ребра, на якому закінчується з’єднання, *fromLane* та *toLane* — відповідають за ідентифікатори смуг вхідного та вихідного ребер, де рух на перехресті починається і закінчується відповідно, напрямок руху *dir* - *«s»* = прямо, *«t»* = поворот, *«l»* = ліворуч, *«r»* = праворуч і так далі. Ідентифікатор *via* вказується, якщо при проходженні перехрестя вказується внутрішнє ребро, за інших обставин значення опускається. Так само оптимально вказується і ідентифікатор світлофора *tL, linkIndex* – номер сигналу світлофора, що відповідає фазі та *state* – додатковий сигнал для світлофора, що додається до конкретного з’єднання і відповідає за пріоритет, якщо перехрестя регулюється сигналами світлофора, а не правилами пріоритету [19].

```
<connection from="<FROM_EDGE_ID>" to="<TO_EDGE_ID>" fromLane="<FROM_LANE_INDEX>" toLane="<TO_LANE_INDEX>"
            via="<VIA_LANE_ID>" tl="<TRAFFIC_LIGHT_ID>" linkIndex="12" dir="r" state="o"/>
```

Лістинг 9 Структура компоненту зв’язків

3.1.2 Транспортні засоби та файл маршрутів

Однією з основних частин симуляції є транспортні засоби та їхні маршрути. Транспортний засіб в SUMO складається з трьох частин : сам транспортний засіб, тип транспортного засобу, який описує його фізичні властивості та маршрут котрим він буде рухатися [22]. Маршрути і тип автотранспорту можуть бути спільними для кількох одиниць. Зазвичай, якщо явно не вказувати при створенні тип транспортного засобу, використовується тип за замовчуванням.

У симуляторі SUMO транспортні засоби та маршрути описуються в XML-файлах з розширенням *.rou.xml. Ці файли можна створювати як вручну так і за допомогою вбудованих інструментів, наприклад, Python скрипт *randomTrips.py* , що генерує випадкові поїздки на основі заданих параметрів. У SUMO основна одиниця транспортної симуляції є транспортний засіб і його характеризує компонент <*vehicle*>. Кожна одиниця транспортного засобу повинна мати свій ідентифікатор *id*, тип *type*, ідентифікатор маршруту яким буде слідувати *route* та крок часу, на якому транспортний засіб повинен в'їхати в мережу *depart* [22].

```
<vehicle id="0" type="type1" route="route0" depart="0" color="1,0,0"/>
```

Лістинг 10 Приклад компоненту транспортного засобу

Компонент тип транспортного засобу <*vType*> слугує в якості універсального компонента для опису групи автотранспорту та описує основні характеристики [22]. Цей компонент налічує велику кількість атрибутів, що дозволяє максимально реалістично відтворити рух транспортних засобів на дорозі, сформувавши індивідуальний набір характеристик. До фізичних властивостей автомобілів належать категорія транспорту *vClass*, колір *color*, довжина *length*, ширина *width*, висота *height,* маса *mass* та мінімальний проміжок між іншими учасниками дорожнього руху *minGap* – всі ці атрибути мають одиниць вимірювання метри. До технічних характеристик належать швидкісні параметри – технічно можлива максимальна швидкість *maxSpeed*, бажана максимальна швидкість транспортного засобу *desiredMaxSpeed*, множник до обмежень швидкості *speedFactor,* варіативні відхилення *speedDev*, динамічні показники - максимальне прискорення *accel*, штатне гальмування *decel*, екстренне гальмування *emergencyDecel*, сприймане гальмування *apparentDecel*, енергетичні характеристики – клас екологічності *emissionClass*, витрата палива *fuel*, потужність двигуна *power* [22].

```
<vType id="type1" accel="0.8" decel="4.5" sigma="0.5" length="5" maxSpeed="70"/>
```

Лістинг 11 Приклад компоненту типу транспортного засобу

Також невід'ємною частиною транспортної симуляції є маршрути для автомобілів, за які відповідає компонент <*route*>. Вони визначають шляхи пересування транспортних засобів дорожньою мережею, описуючи покроково інструкцію з пересування. Описується послідовність ділянок дороги *edge*, якими повинен проїхати транспортний засіб, щоб потрапити в зазначену точку, маршрут повинен містити в собі хоча б одне ребро. Кожен маршрут складається з набору ідентифікаторів ребер, що відповідають конкретним ділянкам дороги в мережі. Маршрути можуть бути налаштовані як вручну, так і згенеровані автоматично на основі точок початку та призначення. Також, обов'язковим критерієм є те, що маршрути мають бути пов'язаними між собою. На даний момент симуляція видає помилку, якщо наступне ребро поточного маршруту не є наступником поточного ребра або якщо транспортному засобу не дозволено їхати по жодній зі смуг [22]. Маршрути можна визначати як індивідуально для кожного транспортного засобу, так і для громадського транспорту з можливістю використовувати один і той самий маршрут різними транспортними засобами. Таким чином можна реалізувати симуляцію регулярного перевезення громадським транспортом : автобусами, трамваями, поїздами і тому подібне.

```
<route id="route1" edges="edgeA edgeB edgeC"/>
```

Лістинг 12 Приклад структури компоненту маршруту

Маршрути в SUMO генеруються на основі точок призначення які описуються в компоненті <*trips*>. Поїздка має ідентифікатор *id*, який відповідає ідентифікатору транспортного засобу для якого призначені точки відправлення та прибуття, крок часу *depart*, на якому транспортний засіб повинен в'їхати в мережу, характеристику смугу з якої почнеться рух автомобіля *departLane* та ідентифікатори ребер *from* і *to*, початкових та кінцевих ребер відповідно.

```
<trip id="tripId"
depart="departTime" departLane="departLaneValue"
 from="fromEdgeId" to="toEdgeId"/>
```

Лістинг 13 Приклад структури компоненту поїздки

3.1.3 Конфігураційний файл

Симуляція налічує величезну кількість даних, тим паче якщо виконувати моделювання реального міста з автотранспортом, громадським транспортом, пішоходами та тому подібне. За таких умов вхідних даних буде безліч. Тому аби правильно керувати симуляцією існує конфігураційний файл, який включає в себе усі основні компоненти симуляції та додаткові файли. Більше того, програма має запускатися виключно через цей конфігураційний файл. Файл конфігурації – це XML-файл котрий має розширення **.sumo.cfg* який має кореневий елемент з назвою *configuration.* Параметри записуються як імена елементів, а потрібне значення зберігається в атрибуті *value.* Налаштування мережі зберігаються в атрибуті *<net-file>,* файл маршрутів визначається в атрибуті *<route-files>* ці два файли є обов'язковими, без них симуляції не відбудеться. В додаткових параметрах *<additional-files>*, що є оптимальними в конфігураційному файлі, можуть міститися налаштування зупинок громадського транспорту, паркінги, магазини, лікарні, тощо [23].

```
<configuration>
    <input>
        <net-file value="test.net.xml"/>
        <route-files value="test.rou.xml"/>
        <additional-files value="test.add.xml"/>
    </input>
</configuration>
```

Лістинг 14 Приклад структури конфігураційного файлу

Для запуску симуляції з консолі, потрібно виконати команду з викликом конфігураційного файлу, наприклад : sumo -c test.sumo.cfg. Параметр -c, тобто --

configuration-file завантажує названий конфігураційний файл. Якщо потрібно побачити процес симуляції з візуальним інтерфейсом , то необхідно відкрити конфігураційний файл в додатку sumo-gui і відбудеться такий самий запуск як з консолі тільки з графічним інтерфейсом і можна буде побачити процес симуляції наживо.

3.2 Методика формування сценаріїв для імітаційного моделювання

Транспортна симуляція допомагає ефективно проаналізувати наявну транспортну інфраструктуру, для цього потрібно створювати численні сценарії, які будуть охоплювати різні навантаження на транспортну мережу. Налаштування таких сценаріїв вручну є трудомістким процесом, вимагає багато часу та ускладнює процес проведення масштабних тестових сценаріїв. У межах цієї роботи було реалізовано підхід для автоматичного створення сценаріїв симуляції, котрий дозволяє динамічно змінювати кількість транспортних засобів в симуляції, формувати для них точки призначення та маршрути та генерувати файли конфігурацій без додаткового втручання користувача. При розробці використовувались інструменти, що входять в програмний пакет SUMO, мова програмування Python та його бібліотеки. Нижче буде наведено код та пояснення запропонованого способу автоматичної генерації сценаріїв.

```
SUMO_TOOLS_PATH = "C:/Program Files (x86)/Eclipse/Sumo/tools"
CONFIG = {
    'simulation': {
        'begin': 0,
        'end': 10000
    },
    'output' : {
        'fcd_period': 100,
        'tripinfo_period': 100
    }
}
```

Лістинг 15 Визначення глобальних параметрів

Перед початком самої реалізації потрібно визначити глобальні параметри. SUMO_TOOLS_PATH визначає локальний шлях до інструментів SUMO. Даний симулятор має велику кількість вбудованих допоміжних інструментів, які завантажуються разом з симулятором. У словнику CONFIG зберігаються важливі параметри які входять до складу конфігураційного файлу *simulation* : час початку симуляції *begin,* час кінця симуляції *end* та налаштування вихідних файлів симуляції *output* : інтервал часу, що показує як часто з автомобілів зчитується інформація *fcd_period* та інтервал часу, що показує як часто записувати інформацію про поїздки *trip_info.*

```
class TrafficSimulation:  1 usage
    def __init__(self):...

    def run_command(self, description, command):...

    def generate_trips(self, vehicles):...

    def run_routing(self, trips_file, vehicles):...

    def create_config(self, routes_file, vehicles):...

    def run_simulation(self, config_file, vehicles):...

    def run(self):...
```

Лістинг 16 Визначення класу симуляції

Був створений клас *TrafficSimulation* який реалізовує основну логіку циклу створення сценарію для симуляції. Під час ініціалізації створюється змінна *self.vehicles_range,* яка зберігає в собі діапазон кількості транспортних засобів. На кожній ітерації кількість автомобілів збільшується на 500, поки не дійде до 5000. Таке число було обрано шляхом експериментальних запусків, оскільки після 5000 виділеної мапи не вистачає для такої кількості авто і симуляція стає в мертву точку.

```
def __init__(self):
    self.vehicles_range = range(500, 5001, 500)
```

Лістинг 17 Ініціалізація класу *TrafficSimulation*

Функція *run_command* призначена для запуску зовнішніх команд та виводить опис операції перед її виконанням.

```
def run_command(self, description, command):  3 usages
    """Function to execute commands"""
    print(f"\n{description}...")
    try:
        result = subprocess.run(command, check=True, text=True, capture_output=True)
        print(f"Successful! Output:\n{result.stdout[:200]}...")
        return True
    except subprocess.CalledProcessError as e:
        print(f"Error:\nCode: {e.returncode}\n{e.stderr}")
        return False
```

Лістинг 18 Функція для запуску команд

Одна з основних функцій – *generate_trips,* що генерує випадкові поїздки для транспортних засобів за допомогою вбудованого скрипта в Sumo *randomTrips.py*. Було вирішено генерувати саме випадкові поїздки, оскільки так можна буде більш об’єктивно визначити проблемні точки в мережі. При виконанні скрипту потрібно вказати файл мережі *-n,* як повинен називатися вихідний файл *-o,* час початку та кінця симуляції, інтервал між відправленнями авто *-p,* що обраховується як час кінця мінус час початку, що потім ділиться на кількість авто, додаткові параметри *--trip-attributes,* налаштування смуги відправлення *departLane* та для перевірки коректності створених поїздок потрібно вказати параметр *--validate.*

```
def generate_trips(self, vehicles):  1 usage
    """Random trip generation (trips)"""
    trips_file = f"trips_{vehicles}.xml"
    cmd = [
        "python",
        str(Path(SUMO_TOOLS_PATH) / "randomTrips.py"),
        "-n", "vdnhMap.net.xml",
        "-o", trips_file,
        "--begin", str(CONFIG['simulation']['begin']),
        "--end", str(CONFIG['simulation']['end']),
        "-p", str((CONFIG['simulation']['end'] - CONFIG['simulation']['begin']) / vehicles),
        "--trip-attributes", "departLane='best'",
        "--validate"
    ]
    return self.run_command( description: f"Create {vehicles} trips", cmd), trips_file
```

Лістинг 19 Функція для генерації поїздок

Функція *run_routing* створює маршрути для згенерованих поїздок у минулій функції. *jtrrouter* обчислює маршрути, які можуть бути використані сумо на основі об'ємів трафіку та коефіцієнтів розвороту на перехрестях [24]. На цьому етапі створюється файл маршрутів з детально описаними ребрами яким повинен слідувати транспортний засіб. *Jtrrouter* також є частиною програмного пакету SUMO і для його виконання потрібно вказати параметри : файл поїздок, файл мережі, файл для збереження згенерованих маршрутів, час початку та кінця, а параметр *--accept-all-destinations* допомагає уникнути помилок маршрутизації, якщо точка зазначена в пунктах призначення виявилася недосяжною.

```
def run_routing(self, trips_file, vehicles):  1 usage
    """Route optimization"""
    routes_file = f"vdnhRoutes_{vehicles}.rou.xml"
    cmd = [
        "jtrrouter",
        f"--trip-files={trips_file}",
        f"--net-file=vdnhMap.net.xml",
        f"--output-file={routes_file}",
        "--begin", str(CONFIG['simulation']['begin']),
        "--end", str(CONFIG['simulation']['end']),
        "--accept-all-destinations"
    ]
    return self.run_command( description: f"Route optimization for {vehicles} vehicles", cmd), routes_file
```

Лістинг 20 Функція з генерації маршрутів

Функція *create_config* об’єднує раніше створені файли маршрутів та мережі та додає налаштування для вихідних файлів. Файл конфігурації генерується за допомогою *xml.etree.ElementTree,* що формує файл XML.

```python
def create_config(self, routes_file, vehicles):  1 usage
    """Creating a configuration file for simulation"""
    config_file = f"vdnh_{vehicles}.sumo.cfg"
    config = ET.Element('configuration')

    input_elem = ET.SubElement(config, tag: 'input')
    ET.SubElement(input_elem, tag: 'net-file', attrib: {'value': 'vdnhMap.net.xml'})
    ET.SubElement(input_elem, tag: 'route-files', attrib: {'value': routes_file})

    time_elem = ET.SubElement(config, tag: 'time')
    ET.SubElement(time_elem, tag: 'begin', attrib: {'value': str(CONFIG['simulation']['begin'])})
    ET.SubElement(time_elem, tag: 'end', attrib: {'value': str(CONFIG['simulation']['end'])})

    output_elem = ET.SubElement(config, tag: 'output')
    ET.SubElement(output_elem, tag: 'fcd-output', attrib: {'value': f"fcd_output_{vehicles}.xml"})
    ET.SubElement(output_elem, tag: 'tripinfo-output', attrib: {'value': f"tripinfo_{vehicles}.xml"})

    tree = ET.ElementTree(config)
    tree.write(config_file, encoding='utf-8', xml_declaration=True)

    return config_file
```

Лістинг 21 Функція створення конфігураційного файлу

Функція *run_simulation* передбачена для запуску симуляції. Використовує конфігураційний файл, налаштовує період запису fcd файлу та записує інформацію про незакінчені поїздки.

```python
def run_simulation(self, config_file, vehicles):  1 usage
    """Execute the simulation"""
    cmd = [
        "sumo",
        "-c", config_file,
        "--device.fcd.period", str(CONFIG['output']['fcd_period']),
        "--tripinfo-output.write-unfinished"
    ]
    return self.run_command( description: f"Simulation for {vehicles} vehicles", cmd)
```

Лістинг 22 Функція для запуску симуляції

Фінальний метод *run* поєднує в собі всі раніше зазначені методи та координує процес симуляції. У ній послідовно виконуються всі процеси необхідні для симуляції для кожної нової кількості транспортних засобів : створює поїздки, файл маршрутів, конфігураційний файл та запускає симуляцію.

```python
def run(self):  1 usage
    """Main execution function"""
    for vehicles in self.vehicles_range:
        print(f"\n{'=' * 50}\nProcessing {vehicles} vehicles\n{'=' * 50}")

        success, trips_file = self.generate_trips(vehicles)
        if not success:
            print(f"Missed {vehicles} vehicles due to trips creation error" )
            continue

        success, routes_file = self.run_routing(trips_file, vehicles)
        if not success:
            print(f"Missed {vehicles} vehicles due to trip generation error")
            continue

        config_file = self.create_config(routes_file, vehicles)

        if not self.run_simulation(config_file, vehicles):
            print(f"Missed {vehicles} vehicles due to simalution error")
            continue

    print("\nSimulations completed!")
```

Лістинг 23 Функція для запуску програми

Після проходження всіх кроків для згенерованих 500 авто програма створить 5 файлів : файл поїздок *trips_500.xml*, файл маршрутів vdnhRoutes_500.xml, конфігураційний файл *vdnh_500.sumo.cfg* та два вихідний файли *fcd_outout_500.xml, tripinfo_500.xml.*

Рисунок 4 Приклад створених файлів

3.3 Аналіз вихідних файлів та знаходження проблемних ділянок в мережі

Однією з головних проблем транспортної інфраструктури є затори. Зазвичай, затори утворюються коли попит перевищує пропускну здатність доріг, але є такі ділянки дороги, де згущення транспорту створюються не залежно від

інтенсивності руху. Такі проблемні зони часто є наслідком недоліків у початковому проектуванні транспортної інфраструктури — наприклад, нераціональне розташування перехресть, вузькі смуги або непродумане зонування. Аналіз у цій роботі орієнтований саме виявлення таких ділянок, де затори зумовлені помилками на етапі проектування, а не тимчасовим збільшенням навантаження. Проблемними ділянками вважаться смуги та перехрестя, де накопичуються автомобілі і мають великий час очікування в одній точці.

Для виявлення проблемних ділянок під час аналізу будуть використовуватися вихідні файли симуляції *fcd.xml* та *tripinfo.xml*. З файлу *tripinfo.xml* можна отримати дані про час очікування в дорозі для кожного авто, а *fcd.xml* зберігає в собі інформацію про знаходження транспортного засобу на смузі у конкретний момент часу.

```
def find_high_waiting_vehicles(tripinfo_path, waiting_time_threshold=50.0):  1 usage
    tree = ET.parse(tripinfo_path)
    root = tree.getroot()

    high_waiting_vehicles = []
    for trip in root.findall('tripinfo'):
        waiting_time = float(trip.get( key: 'waitingTime',  default: 0))
        if waiting_time >= waiting_time_threshold:
            vehicle_id = trip.get('id')
            high_waiting_vehicles.append(vehicle_id)

    print(f"Found {len(high_waiting_vehicles)} vehicles with waitingTime >= {waiting_time_threshold}")
    return high_waiting_vehicles
```

Лістинг 24 Функція для знаходження авто з великим часом очікування

Функція *find_high_waiting_vehicles* отримує файл *tripinfo.xml* і для кожного авто шукає *waitingTime,* що більше 50 і додає їх ідентифікатори у список.

```python
def find_congested_lanes(fcd_path, high_waiting_vehicles):  1 usage
    tree_fcd = ET.parse(fcd_path)
    root_fcd = tree_fcd.getroot()

    lane_congestion = defaultdict(int)

    for timestep in root_fcd.findall('timestep'):
        for vehicle in timestep.findall('vehicle'):
            vehicle_id = vehicle.get('id')
            if vehicle_id in high_waiting_vehicles:
                lane = vehicle.get('lane')
                if lane:
                    lane_congestion[lane] += 1

    return sorted(lane_congestion.items(), key=lambda x: x[1], reverse=True)
```

Лістинг 25 Функція для знаходження найбільш завантажених смуг

Функція *find_congested_lanes* отримує файл *fcd.xml* та список ідентифікаторів транспортних засобів з великим часом очікування. Для кожного кроку часу функція знаходить усі транспортні засоби, що знаходяться в симуляції і якщо транспортний засіб належить до списку авто з високим часом очікування записується смуга на якій він зараз перебуває. І чим частіше трапляється смуга на кожному кроці циклу тим більше буде *lane_congestion.*

3.4 Виділення проблемних ділянок на карті

Для того аби відобразити проблемні ділянки на реальній мапі потрібно підготувати дані. Функція *extract_lane_shapes* призначена для того, аби отримати координати смуг за заданими ідентифікаторами, які були отримані на попередніх кроках аналізу.

```python
def extract_lane_shapes(net_file, target_lanes):  1 usage
    tree = ET.parse(net_file)
    root = tree.getroot()

    lane_data = {}
    for edge in root.findall('edge'):
        for lane in edge.findall('lane'):
            lane_id = lane.get('id')
            if lane_id in target_lanes:
                shape_str = lane.get('shape')
                points = [tuple(map(float, point.split(','))) for point in shape_str.split()]
                lane_data[lane_id] = points
    return lane_data
```

Лістинг 26 Функція для отримання координат смуг

Наступним кроком відбувається конвертація координат SUMO в реальні координати у широту й довготу WGS84. Маючи значення *netOffset x* та *netOffset y* можна зробити зсув та відбувається перетворення з UTM в географічні координати.

```python
def convert_to_latlon(lane_shapes, x_offset, y_offset, from_crs):  1 usage
    transformer = Transformer.from_crs(from_crs, crs_to: "epsg:4326", always_xy=True)
    lanes_latlon = {}

    for lane_id, points in lane_shapes.items():
        lane_points = []
        for x_sumo, y_sumo in points:
            x_utm = x_sumo + x_offset
            y_utm = y_sumo + y_offset
            lon, lat = transformer.transform(x_utm, y_utm)
            lane_points.append((lat, lon))
        lanes_latlon[lane_id] = lane_points
    return lanes_latlon
```

Лістинг 27 Функція конвертації координат

Використовуючи бібліотеку Folium, яка дозволяє візуалізувати дані на інтерактивній карті виділяються проблемні ділянки за допомогою функції *highlight_traffic_jam*. Спочатку обчислюється середнє значення широти і довготи з усіх точок і це середнє значення використовується як центр карти. І на основі словника, що містить набір точок малюється лінія на інтерактивній карті. Після виконання всіх вище описаних функцій локально створюється HTML-файл, який можна відкрити в будь-якому зручному браузері і побачити результат.

```python
def highlight_traffic_jam(lanes_latlon, output_file):  1 usage
    all_points = [pt for lane in lanes_latlon.values() for pt in lane]
    lat_center = sum(lat for lat, _ in all_points) / len(all_points)
    lon_center = sum(lon for _, lon in all_points) / len(all_points)

    m = folium.Map(location=[lat_center, lon_center], zoom_start=15)
    for lane_id, points in lanes_latlon.items():
        folium.PolyLine(
            locations=points,
            color='blue',
            weight=4,
            tooltip=f"Lane: {lane_id}"
        ).add_to(m)

    m.save(output_file)
    print(f"Net saved to file: {output_file}")
```

Лістинг 28 Виділення проблемних ділянок на мапі

Після проведення аналізу вихідних файлів симуляції були знайдені та висвітлені проблемні точки і в Додатку А, Додатку Б та Додатку В представлені результати. На мапах чітко видно виділені синім кольором смуги на яким найчастіше утворюються затори як для 1000, 2500 так і для 5000 транспортних засобів в симуляції. Отримані результати доводять те, що не залежно він попиту на маршрути проблемні точки залишаються однаковими. Отримані результати можна в подальшому використовувати для покращення транспортної інфраструктури і усунення проблемних ділянок.

Висновки

У ході роботи було досліджено імітаційні моделі та проаналізовано особливості процесу транспортної симуляції. Зокрема, було розглянуто рівні деталізації імітаційних моделей та методи транспортного моделювання. Досліджено сучасні програмні продукти для симуляції дорожнього руху та було сформовано їхні особливості та інструменти. Також, аргументовано вибір симулятора SUMO як інструменту для роботи з симуляцією та було описано його основні компоненти для формування сценарію. Було розроблено підхід генерації сценаріїв з використанням інструментів SUMO для запуску симуляцій на основі різної кількості авто. У роботі було запропоновано метод аналізу вихідних файлів симуляції для знаходження проблемних ділянок в транспортній інфраструктурі. Було доведено за допомогою аналізу, що проблемні ділянки в транспортній інфраструктурі можуть формуватися не через тимчасове збільшення навантаження на дорожню мережу, а зумовлені помилками на етапі проектування. Та було знайдено та виділено ці ділянки на реальній мапі, де найчастіше формуються затори – ці дані можуть допомогти інженерам з транспортного планування коректно налагодити дорожню мережу.

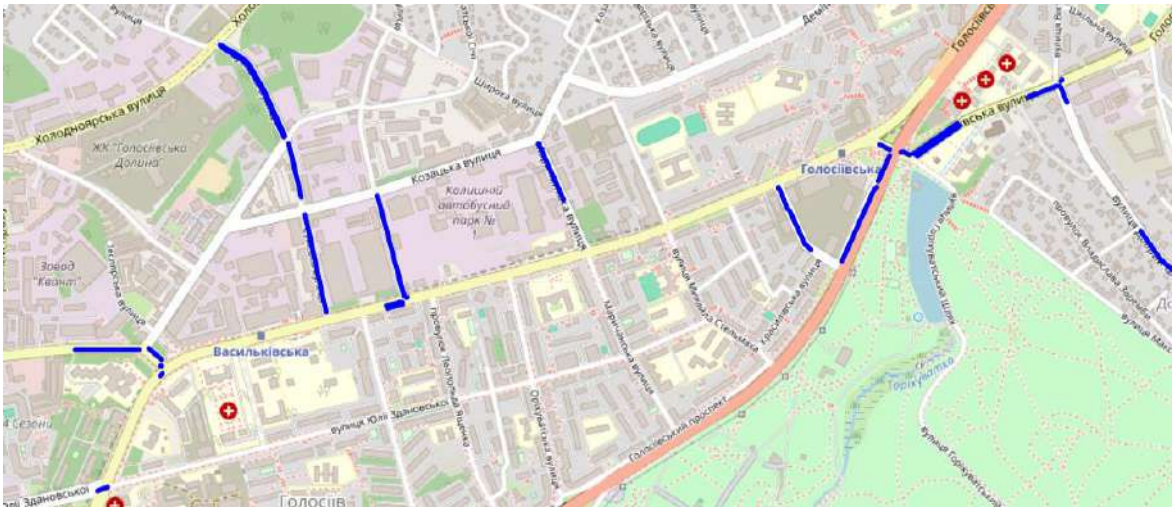
Список використаних джерел

1. https://eclipse.dev/sumo/
2. https://uk.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%BE%D0%BA%D1%81_%D0%91%D1%80%D0%B5%D1%81%D0%B0
3. https://pypi.org/project/matsim-tools/
4. https://matsim.org/files/book/partOne-latest.pdf
5. https://en.wikipedia.org/wiki/PTV_Group
6. https://www.ptvgroup.com/en/about/academia
7. https://cityflow.readthedocs.io/en/latest/introduction.html
8. https://arxiv.org/pdf/1905.05217
9. https://www.ptvgroup.com/en/application-areas/transportation-modeling#recommendedproduct
10. https://library.knuba.edu.ua/books/8_1_23.pdf
11. http://biblio.umsf.dp.ua/jspui/bitstream/123456789/4793/1/%D0%A1%D0%BE%D1%85%D0%B0%D1%86%D1%8C%D0%BA%D0%B8%D0%B9%20%D0%BC%D0%BE%D0%BD%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D1%96%D1%8F.pdf
12. https://ocw.tudelft.nl/wp-content/uploads/Chapter-9.-Macroscopic-traffic-flow-models.pdf
13. https://ocw.tudelft.nl/wp-content/uploads/Chapter-14.-Microscopic-simulation-models.pdf
14. https://www.researchgate.net/publication/255654806_Mesoscopic_Simulation_Models_for_Short-Term_Prediction
15. https://www.transitwiki.org/TransitWiki/index.php/Four-step_travel_model#cite_note-1
16. https://en.wikipedia.org/wiki/Transportation_forecasting#Four-step_models
17. https://assets.publishing.service.gov.uk/media/666af3bf50dca4553304f334/tag-unit-m5-4-agent-based-methods-activity-based-demand-modelling.pdf

18. https://onlinelibrary.wiley.com/doi/10.1155/2022/1252534
19. https://sumo.dlr.de/docs/Networks/SUMO_Road_Networks.html
20. https://sumo.dlr.de/docs/Geo-Coordinates.html
21. https://sumo.dlr.de/docs/Simulation/Traffic_Lights.html
22. https://sumo.dlr.de/docs/Definition_of_Vehicles%2C_Vehicle_Types%2C_and_Routes.html
23. https://sumo.dlr.de/docs/Basics/Using_the_Command_Line_Applications.html
24. https://sumo.dlr.de/docs/jtrrouter.html
25. https://ela.kpi.ua/server/api/core/bitstreams/3fe27852-776f-4130-bc62-c9c2be44ed6e/content
26. https://www.sciencedirect.com/science/article/pii/S2192437624000050

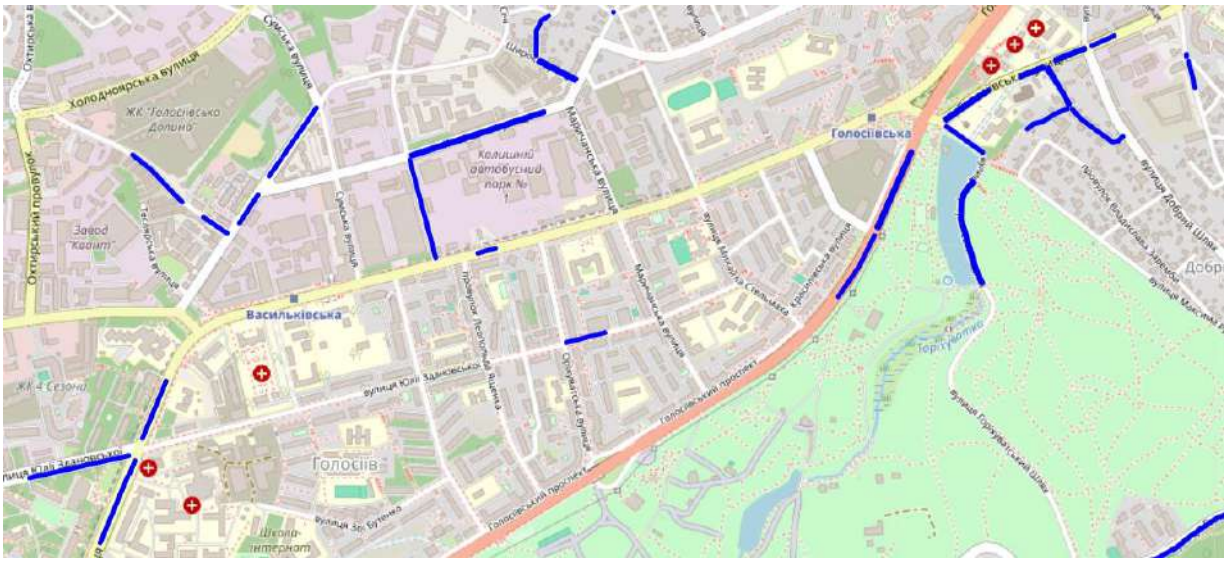
Додатки

Додаток А. Результат аналізу, виділення проблемних ділянок на мапі. Симуляція для 1000 автомобілів



Додаток Б. Результат аналізу, виділення проблемних ділянок на мапі. Симуляція для 2500 автомобілів

Додаток В. Результат аналізу, виділення проблемних ділянок на мапі. Симуляція для 5000 автомобілів


Холодногірська вулиця
Сумська вулиця
ЖК "Голосіївська Долина"
Комунальний автобусний парк №
Маршрутська вулиця
Голосіївська
Завод "Кварт"
Васильківська
вулиця Юлії Здановської
Голосіїв
Голосіївський проспект
вулиця Горлівська
Школа-інтернат