ШИФР “МАРШРУТ”

Конкурсна робота

на тему:

“ІНТЕЛЕКТУАЛЬНИЙ МІСЬКИЙ ТРАФІК”

2019

Зміст

Вступ

Принциповим питанням міського трафіку є час проїзду транспортного засобу (ТЗ) по вибраному маршруту. Зрозуміло, що цей час треба зробити мінімальним для кожного ТЗ. Іншими словами, потрібно прокласти оптимальний по часу маршрут (назвемо його t-оптимальним) для кожного ТЗ, що приймає участь у трафіку. У великому місті таких ТЗ може бути понад мільйон. Тому важливо забезпечити оптимальні умови проїзду для всіх автомобілів, що приймають участь у міському трафіку. Проблема далеко не нова, але не вирішена та набуває все більшої гостроти, особливо у великих містах-мегаполісах. Саме на вирішення такої проблеми спрямована представлена робота.

Розділ I. Огляд літературних джерел

Принципи взаємодії між дорожньою інфраструктурою та автомобілем детально описані у роботах [1-4]. Вказана взаємодія здійснюється з допомогою так званих точок доступу, розташованих вздовж автомобільної дороги та на перехрестях. В роботі [5] автори розглядають проблему оптимального розташування датчиків. Для досліджень принципових питань дорожнього міського трафіку у дослідженні [6] застосовується модифікована модель стільникових автоматів. Автори праці [7] використовують «розумну» мережеву імітаційну модель; в якості об’єкта досліджень використана центральна та західна частина міста Сінгапур. Ідея покращення трафіка на платформі NetLogo базується на координованому регулюванні автомобільних потоків за допомогою бездротового зв'язку між автомобілями, як було запропоновано в роботі [8]. Винахід [9] відноситься до інтелектуальної системи керування режимом роботи світлофорів через двосторонній обмін інформацією з міським пунктом керування трафіком. Проблеми паркінгу як різновид проблеми трафіку розглянуті в дослідженні [10]. Технологія запропонована компанією Siemens. Робота [11] надає результати досліджень трафіку, які були проведенні в штаті Юта (США). В дослідженні розглянута система показників

ефективності регулювання руху потоків ТЗ на основі аналізу даних, які отриманні із спеціальних мікрохвильових датчиків. В працях [12-13] для вирішення проблем, пов'язаних із дорожнім рухом, була використана рідинна динаміка. Тему взаємодії приватних автомобілів та громадського транспорту розглянуто в дослідженні [14]. Щоб розв'язати проблему такої взаємодії автори вводять поняття двотипних (бімодальних) міських мереж (bi-modal urban networks). З метою уникнення перевантажень, забезпечення пріоритету для автомобілів екстрених служб і скорочення середнього часу очікування ТЗ на перехрестях створена система навігації, описана в [15]. Системне дослідження управління трафіком з використанням бездротових датчиків та режим інтелектуального управління світлофором описані в [16].

Розділ II. Результати досліджень та їх аналіз

Особливо зауважимо, що саме час проїзду ТЗ по вибраному маршруту є принципово важливим критерієм, а не геометрична відстань. До речі, сучасна GPS-навігація прокладає, як правило, оптимальні маршрути з точки зору геометричної відстані (назвемо їх g-оптимальними) між початковим та кінцевим положеннями ТЗ. На відміну від цього, в основі нашої технології лежить принцип оптимізації по часу, зокрема оптимізації проїзду через окреме перехрестя (рис.1). Дійсно, в першу чергу саме на перехрестях виникають затори. Організувавши ефективний рух через міські перехрестя, досягнемо кращої ефективності трафіку по всьому місту. Але це лише перша, і не головна, фаза регулювання міського трафіку в плані збільшення його ефективності. Тому розглядати цю фазу детально ми не будемо. Їй приділена детальна увага в роботах [1-4]. Представимо лише схему (рис. 1) організації міського трафіку на першій фазі.

Для переходу до другої фази регулювання трафіку потрібно спочатку організувати реєстрацію ТЗ на ділянках дороги між сусідніми перехрестями. З цією метою використовуються вхідні датчики перехрестя А та вихідні датчики перехрестя В. Саме ці дві групи датчиків працюють сумісно – як показано на рис.2. Отже, з однієї сторони, вхідні та вихідні датчики на одному перехресті

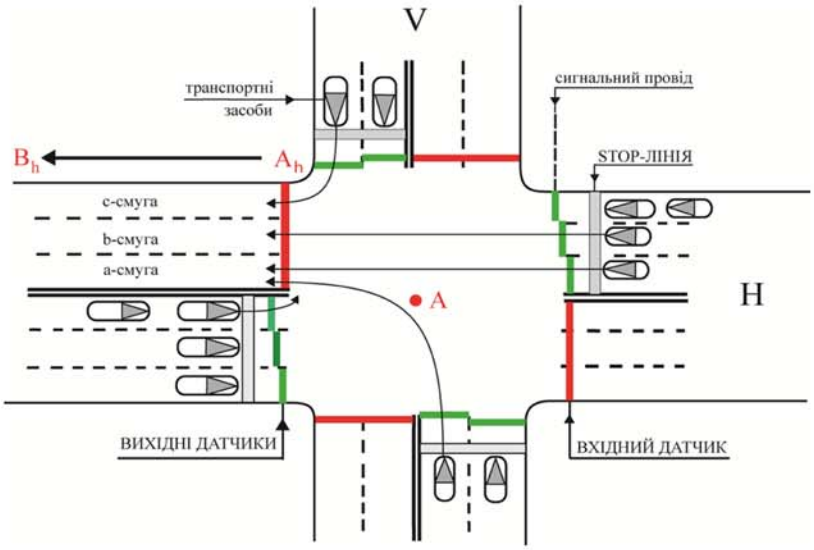

Рис.1. Окреме перехрестя А, оснащене датчиками – вхідними та вихідними (вхідні – червоні смужки, вихідні – три рядом розташовані зелені смужки). Скругленими прямокутниками зображені ТЗ, маршрути яких пролягають в напрямку $A_h B_h$. Вхідні та вихідні датчики з'єднані з дорожним контролером. Символ V символізує вертикальний напрямок, Н – горизонтальний.

працюють для регуляції ефективного проїзду через окреме перехрестя, а з іншої, – для реєстрації завантаженості доріг ТЗ між сусідніми перехрестями. В останньому випадку використовуються дані із вхідних датчиків перехрестя А та вихідних датчиків сусіднього перехрестя В.

На рис.2 для прикладу зображено траєкторію руху автомобіля 4. Цей ТЗ на ділянці руху $A_h B_h$ реєструється вхідним датчиком A_h та вихідним датчиком B_h^a. Спочатку даний автомобіль в'їжджає на смугу руху $(A_h B_h)_c$, потім перелаштовується на смугу руху $(A_h B_h)_b$ і остаточно займає смугу руху $(A_h B_h)_a$ і тому реєструється датчиком B_h^a. Далі вказаний автомобіль здійснює

поворот наліво на перехресті В, де реєструється вхідним датчиком вже цього перехрестя.

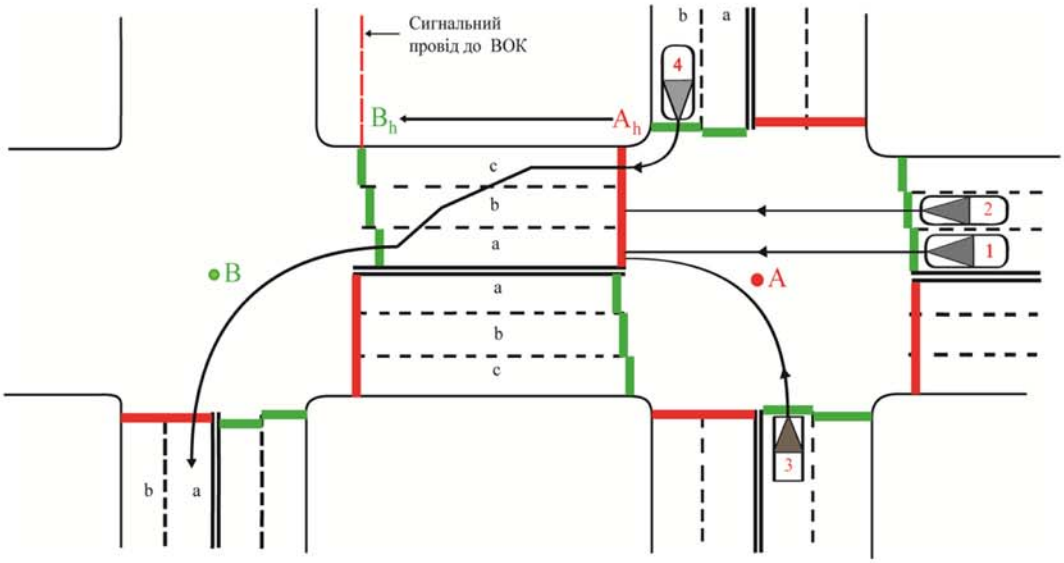

Рис.2. Два сусідніх перехрестя – А і В. Показано групи автомобілів 1,2,3 і 4, що під'їхали до перехрестя А, маршрут яких пролягає через ділянку дороги $A_h B_h$ в напрямку перехрестя В.

Отже, завантаженість ділянки дороги між сусідніми перехрестями $A_h B_h$ вимірюється вхідним датчиком перехрестя А та вихідними датчиками перехрестя В. Ці дві групи датчиків працюють сумісно.

Вхідний датчик – один на всю ширину проїзної частини дороги одного напрямку. Цей датчик реєструє всі автомобілі, що в'їжджають з різних напрямків на ділянку дороги одного напрямку $A_h B_h$. На цій ділянці дороги автомобілі розподіляються (перелаштовуються) по смугах руху у відповідності із своїм подальшим маршрутом. При виїзді із ділянки дороги одного напрямку $A_h B_h$ потрібно зареєструвати всі групи автомобілів, що зайняли свої смуги руху – $a,b,c,d,...$ (символом a позначається смуга руху, що примикає до осьової лінії дороги, сусідня до неї смуга позначається символом b і т.д.). Число вихідних датчиків залежить від числа смуг проїзної частини дороги одного

напрямку (на рис. 2 таких датчиків три у відповідності з числом смуг; позначимо число таких смуг через l). Отже, кожна смуга оснащається власним вихідним датчиком. Для чого? Справа в тому, що автомобілі розташовуються по смугах у відповідності із маршрутом руху кожного з них. Іншими словами, існує l груп автомобілів, маршрути яких різні, але всі їх треба реєструвати при прокладанні маршруту, узгодивши його з правилами дорожнього руху (ПДР). Наприклад, якщо автомобіль розташований у смузі (смуга а), що примикає до розділювальної лінії дороги, то водій цього ТЗ може рухатись наліво або здійснювати розворот на перехресті, але ні в якому разі не повертати направо – такий маневр заборонений ПДР. Таким чином, завдяки сумісній роботі одного вхідного та l вихідних датчиків, є можливість реєструвати l груп ТЗ, що в'їжджають на ділянку дороги між сусідніми перехрестями і відповідно реєструвати завантаженість трафіку на ділянці дороги $A_h B_h$ кожного із цих l маршрутів. Завдяки тому, що вхідний датчик A_h працює сумісно із вихідними датчиками $B_h^j (j = a,b,c...)$, кожна смуга ділянки дороги одного напрямку $A_h B_h$ знаходиться під контролем, що дає можливість визначити завантаженість кожної окремої смуги на цій ділянці.

Власне, будь-який маршрут складається із певної сукупності смуг руху, розташованих одна за одною між сусідніми перехрестями (мовою теорії графів маршрут – це простий ланцюг). Але кожна із таких ділянок дороги знаходиться під контролем – значить, і весь маршрут контролюється. Таким чином, на другій, і найголовнішій фазі ефективного регулювання трафіку є можливість прокладання оптимальних маршрутів для всіх ТЗ, що здійснюють рух по місту та координують свої маршрути з центральним пультом керування трафіком (ЦПКТ). Подібного роду координація відбувається або з допомогою GPS-навігаторів або з допомогою мобільних телефонів із спеціальним додатком.

На кількісному рівні динаміка руху вздовж ділянки дороги $A_h B_h$ обумовлена співвідношенням між величиною ТЗ, що в'їхали за певний час (наприклад, за час, рівний фазі горіння зеленого світла світлофору) та виїхали із

даної ділянки дороги за цей же час. Технічно процес організований наступним чином: датчик A_h (вхідний датчик) реєструє автомобілі (точніше число колісних пар), які в'їжджають на ділянку дороги $A_h B_h$ з усіх можливих напрямків перехрестя А (рис.2). В свою чергу, комплекс вихідних датчиків на перехресті В реєструє автомобілі, що виїхали за межі цієї ділянки. Відношення цих величин якраз і свідчить про динаміку руху на конкретній ділянці дороги.

З метою прокладання t-оптимального маршруту для кожного ТЗ, що приймає участь у трафіку, скористаємось теорією графів, а саме, співставимо міську транспортну мережу із зваженим орієнтованим графом (рис.3). Представлений граф моделює частину транспортної мережі міста. Найбільш прийнятним варіантом для прокладання оптимального маршруту у графі буде А*-алгоритм, оскільки цей алгоритм дозволяє прокласти такий маршрут у графі між двома заданими вершинами, іншими словами між заданими початковим та кінцевим положеннями маршруту.

Вхідні та вихідні датчики являють собою пристрої, що монтуються у полотно дороги (наприклад, це може бути п'єзокристалічний датчик типу RoadTrax BL [1-3]). Їх принцип роботи базується на фізичному явищі п'єзоефекту – при наїзді колісної пари автомобіля на зону розташування датчика генерується електричний імпульс, який реєструється вимірювально-обчислювальним комплексом, що передає дані на ЦПКТ. Таким чином, вхідні та вихідні датчики видають спектр величин $N_{A_h B_h}$ та $n_h^j (j = a,b,c...)$. Перша величина – це число автомобілів, що в'їхали на ділянку дороги між сусідніми перехрестями (на мал.2 це ділянка дороги $A_h B_h$) на протязі зеленої фази роботи світлофора, а друга – це число автомобілів, що виїхали із смуги j цієї ділянки дороги на сусідньому перехресті В за той самий час. Чим ближчим є відношення $N_{A_h B_h} / \sum_{j=1}^{l} n_h^j$ (у цій формулі і далі в подібних формулах з метою сумування по смугах руху зроблена заміна $1 = a, 2 = b, 3 = c...$) до одиниці, тим динаміка руху на вибраній ділянці дороги між перехрестями є кращою.

Навпаки, якщо відношення $N_{A_hB_h} / \sum_{j=1}^{l} n_h^j \gg 1$, то це свідчить або про заблокованість перехрестя В (рис.2) або про дуже низьку динаміку руху ТЗ. Відповідно ваги ребер (іншими словами, завантаженості смуг руху ділянки дороги A_hB_h) здобувають великих значень. Отже, програма, «шукаючи» в графі оптимальний маршрут, омине такі завантажені смуги руху чи ділянку дороги.

Самі ваги ребер графа обраховуються з допомогою обчислень наступного відношення

$$(N_{A_hB_h}^{j(h)} / n_{A_hB_h}^{j(h)})\, l_{A_hB_h}, \qquad (1)$$

де $N_{A_hB_h}^{j(h)}$ – число ТЗ, що в'їжджають на смугу j ділянки дороги h одного напрямку A_hB_h; $n_{A_hB_h}^{j(h)}$ – число ТЗ, що виїжджають із j-ої смуги ділянки дороги одного напрямку A_hB_h; h – індекс, що нумерує ділянки проїзної частини дороги між сусідніми перехрестями; $l_{A_hB_h}$ – геометрична довжина ділянки дороги одного напрямку A_hB_h.

Розрахунок ваги кожного ребра графа здійснюється якраз з допомогою обчислень виразу (1). Величини $n_{A_hB_h}^{j(h)}$ обраховуються вихідними датчиками (число таких датчиків дорівнює числу смуг руху l ділянки дороги h). Вхідні датчики обраховують лише сумарні значення виду

$$\sum_{j=1}^{l} N_{A_hB_h}^{j(h)} = N_{A_hB_h}. \qquad (2)$$

Тут l – число смуг руху на ділянці дороги з номером h.

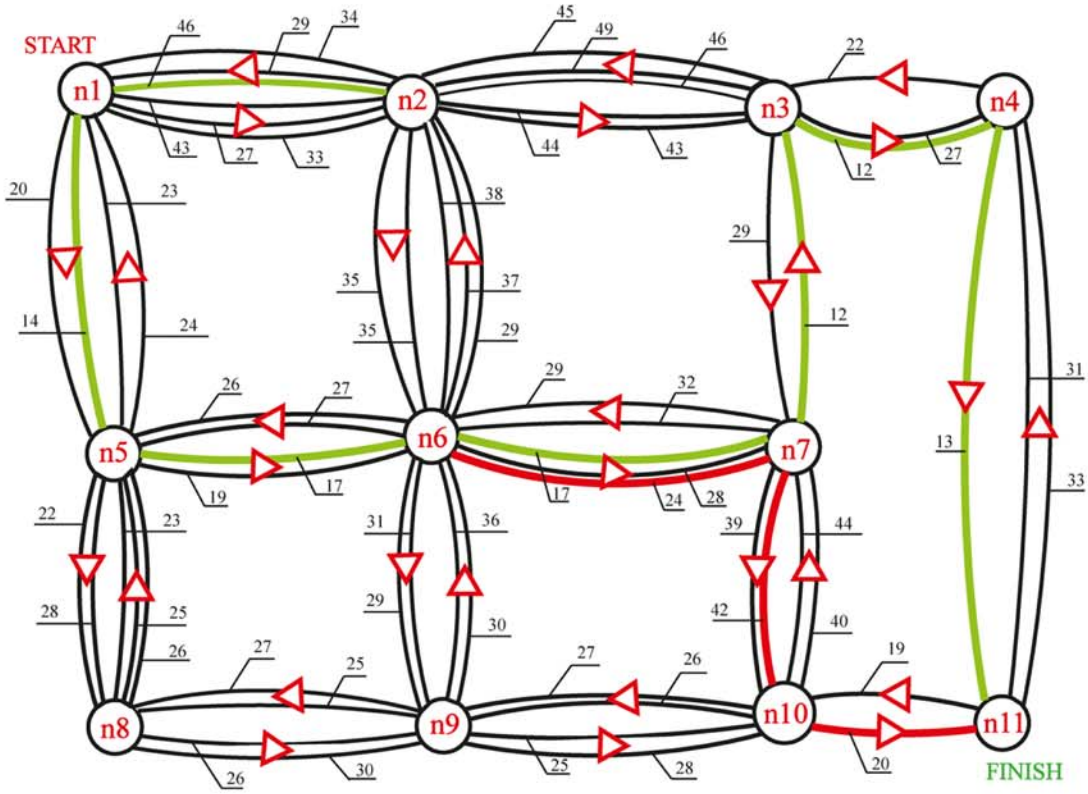

Рис.3. Зважений орієнтований мультиграф, що моделює частину транспортної мережі міста. Кожне ребро графа має вагу, що розраховується для кожного 5-секундного проміжку часу з допомогою виразу 1. Відповідні величини виставляються біля ребер графа. Для прикладу зеленою лінією показано оптимальний маршрут між вершиною n1a2 (START) та вершиною n11a4 (FINISH).

Значення величин $N_{A_h B_h}^{j(h)}$ обраховуються шляхом наступного розрахунку

$$(N_{A_h B_h} - \sum_{j=1}^{l} n_h^{j(h)}) / l . \qquad (3)$$

Тут перший доданок в дужках є сумарне число автомобілів, що в'їхали на ділянку дороги $A_h B_h$ за певний проміжок часу, а другий – число автомобілів, що виїхали із цієї ділянки дороги за цей самий проміжок часу.

У підпису до рис.3, як і далі в програмі `package IC`, введені специфічні позначення вершин графа. Дамо тлумачення таких позначень. Що, наприклад, означає запис n1a2? Фактично це ребро a, що підходить до вузла $n1$ зі сторони вузла $n2$. Іншими словами, смуга руху a, що примикає до перехрестя n1 і виходить із перехрестя n2. Таким чином, кожне перехрестя (або вузол графа n) складається із кількох складових (підвузлів). Наприклад, вузол n6 складається із 8 підвузлів – n6a5, n6b5, n6a2, n6b2, n6a7, n6b7, n6a9, n6b9. Потрібного роду розділення вузла на підвузли потрібно для прокладання маршрутів з урахуванням всіх смуг руху у транспортних мережах міста та для виконання вимог ПДР. Звернемо увагу, що, наприклад, число можливих маршрутів із вузла $n1a2$ (START) у вузол $n11a4$ (FINISH) може досягати сотні, і тому важливо із спектру можливих варіантів вибрати один-єдиний маршрут, проїзд по якому мінімальний по часу.

Сформулюємо наступну умову оптимізації маршруту

$$\sum_{h=1}^{f} (N_{A_hB_h}^{j(h)} \Big/ (n_{A_hB_h}^{j(h)}))l_{A_hB_h} -> min. \qquad (4)$$

Символ f означає число ділянок дороги A_hB_h, які формують маршрут із мультиплікат (1). З допомогою (4) обраховуються сумарна ваги різних маршрутів. A^*-алгоритм знаходить в спектрі величин (4) мінімальне значення. Після цього дані щодо такого оптимального маршруту передаються кожному водієві. У виразі (4) значення величин $n_{A_hB_h}^{j(h)}$ ($j = a,b,c,....$) вимірюються на протязі часу горіння зеленої фази світлофора. Реєстрація цих величин здійснюється вихідними датчиками B_h^j. Розрахунок спектру значень $N_{A_hB_h}^{j(h)}$ здійснюється з допомогою приведеного вище виразу (3). Зрозуміло, що розрахунок геометричних відстаней $l_{A_hB_h}$ між перехрестями не складає проблеми.

На мові теорії графів вираз (4) фактично представляє собою оптимізаційну умову, що визначає ланцюг у графі з мінімальною сумарною вагою.

Знаходження таких ланцюгів у графах можна реалізувати з допомогою спеціальних оптимізаційних алгоритмів, таких як алгоритм Дейкстри чи Флойда. Але для нашої задачі доцільно скористатись A^*-алгоритмом, який прокладає оптимальний маршрут між двома вибраними вершинами графа та володіє незначною алгоритмічною складністю виду $O(N)$, де N – об'єм вхідних даних.

Нехай водій i знаходиться у початкові позиції S_i, а позиція F_i є кінцевою координатою його маршруту. Тоді для кожного заявленого маршруту i програма, представлена далі, формує у відповідності із умовою (4) нерозривний ланцюг, що пов'язує початкову та кінцеву позиції кожного конкретного маршруту. Отже, технологія дозволяє супроводжувати ТЗ по будь-якому міському маршруту оптимальним чином, мінімізуючи ліву частину виразу (4).

Оскільки окремі перехрестя та ділянки між сусідніми перехрестями знаходяться під контролем, то це значить, що вся транспортна мережа міста знаходиться під контролем ЦПКТ. Вираз «під контролем» означає, що є можливість визначати динаміку руху ТЗ як на кожній смузі окремої ділянки дороги, так і на всіх можливих маршрутах міста.

Сформуємо спектр бінарних відношень наступного виду

$$R=\{(S_1,F_1),(S_2,F_2),(S_3,F_3),.....(S_i,F_i),...(S_m,F_m)\} . \qquad (5)$$

Кожна із пар (S_i,F_i) даного відношення задає стартову S_i та фінішну F_i позиції автомобіля i. Пари такого типу є ключовими для ЦПКТ, адже потрібно прокласти t-оптимальний маршрут кожному автомобілю i, а число таких об'єктів m у мегаполісі може перевищувати мільйон. При цьому важливо знайти саме t-оптимальний маршрут руху для кожного ТЗ на основі використання отримуваних з кожного перехрестя даних, що дуже швидко змінюються у відповідності із змінами дорожнього трафіку. Останнє означає, що потрібно використовувати динамічну базу даних, що оновлюється кожні 5 с. Іншими словами, канали «Перехрестя ↔ ЦПКТ» працюють в режимі

реального часу, постійно оновлюючи дані про завантаженість ділянок дороги між перехрестями.

Аналізуючи дані, що поступають із вхідних та вихідних датчиків усіх перехресть міста, ЦПКТ на основі роботи програми видає керуючі сигнали на світлофори кожного перехрестя (рис.1), з однієї сторони, а з іншої, прокладає оптимальний трафік кожному ТЗ, розраховуючи спектр величин на основі виразу (4).

Технічно ситуація виглядає наступним чином: програма на ЦПКТ працює з кожним ТЗ і для кожного такого об'єкта розраховує оптимальний маршрут руху і передає дані водієві на GPS-навігатор чи на мобільний телефон із спеціальним додатком. «Знаючи» розрахований оптимальніший маршрут – на даний момент часу – навігатор «веде» водія по цьому маршруту з використанням постійно поновлюваної, «свіжої» інформації. При цьому водій отримує інформацію про те, де він має повернути і в який бік, розвернутись чи перелаштуватись на іншу смугу руху і т.п. Але дорожня ситуація у сучасному місті змінюється кожну мить. І тому по мірі руху кожного ТЗ програма контролює маршрут та знаходиться в постійному пошуку найефективнішого t-оптимального маршруту.

Java-програма, яка на основі реалізації A^*-алгоритму прокладає t-оптимальний маршрут у графі, представленому на рис.3, виглядає наступним чином:

```
package IC;
import java.util.PriorityQueue;
import java.util.HashSet;
import java.util.Set;
import java.util.List;
import java.util.Comparator;
import java.util.ArrayList;
import java.util.Collections;
public class AstarSearchAlgo {
```

```
//h scores is the stright-line distance from n1 to n11
public static void main(String[] args) {
//initialize the graph map
Node n1a2 = new Node("a-Freedom str", 100);
Node n1b2 = new Node("n1b2", 100);
Node n1c2 = new Node("n1c2", 100);
Node n1a5 = new Node("n1a5", 100);
Node n1b5 = new Node("n1b5", 100);
Node n2a1 = new Node("n2a1", 100);
Node n2b1 = new Node("n2b1", 100);
Node n2c1 = new Node("n2c1", 100);
Node n2a6 = new Node("n2a6", 100);
Node n2b6 = new Node("n2b6", 100);
Node n2c6 = new Node("n2c6", 100);
Node n2a3 = new Node("n2a3", 100);
Node n2b3 = new Node("n2b3", 100);
Node n2c3 = new Node("n2c3", 100);
Node n3a2 = new Node("n3a2", 100);
Node n3b2 = new Node("n3b2", 100);
Node n3a4 = new Node("n3a4", 100);
Node n3a7 = new Node("a-Apple str", 100);
Node n4a3 = new Node("n4a3", 100);
Node n4b3 = new Node("b-Brosdmay str ", 100);
Node n4a11 = new Node("n4a11", 100);
Node n4b11 = new Node("n4b11", 100);
Node n5a1 = new Node("a-Broad av", 100);
Node n5b1 = new Node("n5b1", 100);
Node n5a6 = new Node("n5a6", 100);
Node n5b6 = new Node("n5b6", 100);
Node n5a8 = new Node("n5a8", 100);
```

```
Node n5b8 = new Node("n5b8", 100);
Node n5c8 = new Node("n5c8", 100);
Node n6a5 = new Node("a-Valley str", 100);
Node n6b5 = new Node("n6b5", 100);
Node n6a2 = new Node("n6a2", 100);
Node n6b2 = new Node("n6b2", 100);
Node n6a7 = new Node("n6a7", 100);
Node n6b7 = new Node("n6b7", 100);
Node n6a9 = new Node("n6a9", 100);
Node n6b9 = new Node("n6b9", 100);
 Node n7a6 = new Node("a-Heroes str", 100);
 Node n7b6 = new Node("n7b6", 100);
 Node n7c6 = new Node("n7c6", 100);
 Node n7a3 = new Node("n7a3", 100);
 Node n7a10 = new Node("n7a10", 100);
 Node n7b10 = new Node("n7b10", 100);
 Node n8a5 = new Node("n8a5", 100);
 Node n8b5 = new Node("n8b5", 100);
 Node n8a9 = new Node("n8a9", 100);
 Node n8b9 = new Node("n8b9", 100);
 Node n9a6 = new Node("n9a6", 100);
 Node n9b6 = new Node("n9b6", 100);
 Node n9b8 = new Node("n9b8", 100);
 Node n9a8 = new Node("n9a8", 100);
 Node n9a10 = new Node("n9a10", 100);
 Node n9b10 = new Node("n9b10", 100);
 Node n10a7 = new Node("n10a7", 100);
 Node n10b7 = new Node("n10b7", 100);
 Node n10a9 = new Node("n10a9", 100);
 Node n10b9 = new Node("n10b9", 100);
```

```
 Node n10a11 = new Node("n10a11", 100);
 Node n11a10 = new Node("n11a10", 0);
 Node n11a4 = new Node("a-Long str", 0);
 n1a2.adjacencies = new Edge[]{
 new Edge(n2a1, 43),
 new Edge(n2b1, 27),
 new Edge(n2c1, 33),
 new Edge(n5a1, 14),
 new Edge(n5b1, 20),};
n1b2.adjacencies = new Edge[]{};
n1c2.adjacencies = new Edge[]{};
n1a5.adjacencies = new Edge[]{
new Edge(n5a1, 14),
new Edge(n5b1, 20),};
n1b5.adjacencies = new Edge[]{
new Edge(n2a1, 43),
new Edge(n2b1, 27),
new Edge(n2c1, 33), };
n2a1.adjacencies = new Edge[]{
new Edge(n1a2, 46),
new Edge(n1b2, 29),
new Edge(n1c2, 34),};
n2b1.adjacencies = new Edge[]{
new Edge(n3a2, 44),
new Edge(n3b2, 43),};
n2c1.adjacencies = new Edge[]{
new Edge(n6a2, 35),
new Edge(n6b2, 35), };
n2a3.adjacencies = new Edge[]{
new Edge(n3a2, 44),
```

```
new Edge(n3b2, 43),
new Edge(n6a2, 35),
new Edge(n6b2, 35), };
n2b3.adjacencies = new Edge[]{
new Edge(n1a2, 46),
new Edge(n1b2, 29),
new Edge(n1c2, 34),};
n2c3.adjacencies = new Edge[]{};
n2a6.adjacencies = new Edge[]{
new Edge(n6a2, 35),
new Edge(n6b2, 35),
new Edge(n1a2, 46),
new Edge(n1b2, 29),
new Edge(n1c2, 34), };
n2b6.adjacencies = new Edge[]{};
n2c6.adjacencies = new Edge[]{
new Edge(n3a2, 44),
new Edge(n3b2, 43),};
n3a2.adjacencies = new Edge[]{
new Edge(n2a3, 46),
new Edge(n2b3, 49),
new Edge(n2c3, 45),
new Edge(n4a3, 27),
new Edge(n4b3, 12),  };
n3b2.adjacencies = new Edge[]{
new Edge(n7a3, 29), };
n3a4.adjacencies = new Edge[]{
new Edge(n4a3, 27),
new Edge(n4b3, 12),
new Edge(n2a3, 46),
```

```
new Edge(n2b3, 49),
new Edge(n2c3, 45),
new Edge(n7a3, 29),};
n3a7.adjacencies = new Edge[]{
new Edge(n7a3, 29),
new Edge(n4a3, 27),
new Edge(n4b3, 12),
new Edge(n2a3, 46),
new Edge(n2b3, 49),
new Edge(n2c3, 45), };
n4a3.adjacencies = new Edge[]{
new Edge(n3a4, 22), };
n4b3.adjacencies = new Edge[]{
new Edge(n11a4, 13),};
n4a11.adjacencies = new Edge[]{
new Edge(n11a4, 13),
new Edge(n3a4, 22),};
n4b11.adjacencies = new Edge[]{  };
n5a1.adjacencies = new Edge[]{
new Edge(n1a5, 23),
new Edge(n1b5, 24),
new Edge(n6a5, 17),
new Edge(n6b5, 19),
new Edge(n8a5, 28),
new Edge(n8b5, 22),  };
n5b1.adjacencies = new Edge[]{ };
n5a6.adjacencies = new Edge[]{
new Edge(n6a5, 17),
new Edge(n6b5, 19),
new Edge(n8a5, 28),
```

```
new Edge(n8b5, 22),};
n5b6.adjacencies = new Edge[]{
new Edge(n1a5, 28),
new Edge(n1b5, 22), };
n5a8.adjacencies = new Edge[]{
new Edge(n8a5, 28),
new Edge(n8b5, 22),  };
n5b8.adjacencies = new Edge[]{
new Edge(n1a5, 23),
new Edge(n1b5, 24), };
n5c8.adjacencies = new Edge[]{
new Edge(n6a5, 17),
new Edge(n6b5, 19),  };
n6a5.adjacencies = new Edge[]{
new Edge(n5a6, 27),
new Edge(n5b6, 26),
new Edge(n7a6, 17),
new Edge(n7b6, 28),
new Edge(n7c6, 24),
new Edge(n2a6, 38),
new Edge(n2b6, 37),
new Edge(n2c6, 29), };
n6b5.adjacencies = new Edge[]{
new Edge(n9a6, 131),
new Edge(n9b6, 29), };
n6a2.adjacencies = new Edge[]{
new Edge(n2a6, 38),
new Edge(n2b6, 37),
new Edge(n2c6, 29),
new Edge(n7a6, 17),
```

```
new Edge(n7b6, 28),
new Edge(n7c6, 24),
new Edge(n9a6, 131),
new Edge(n9b6, 29),   };
n6b2.adjacencies = new Edge[]{
new Edge(n5a6, 27),
new Edge(n5b6, 26),  };
n6a7.adjacencies = new Edge[]{
new Edge(n7a6, 17),
new Edge(n7b6, 28),
new Edge(n7c6, 24),
new Edge(n5a6, 27),
new Edge(n5b6, 26),
new Edge(n9a6, 131),
new Edge(n9b6, 29), };
n6b7.adjacencies = new Edge[]{
new Edge(n2a6, 38),
new Edge(n2b6, 37),
new Edge(n2c6, 29),  };
n6a9.adjacencies = new Edge[]{
new Edge(n9a6, 131),
new Edge(n9b6, 29),
new Edge(n2a6, 38),
new Edge(n2b6, 37),
new Edge(n2c6, 29),
new Edge(n5a6, 27),
new Edge(n5b6, 26),  };
n6b9.adjacencies = new Edge[]{
new Edge(n7a6, 17),
new Edge(n7b6, 28),
```

```
new Edge(n7c6, 24),  };
n7a6.adjacencies = new Edge[]{
new Edge(n6a7, 32),
new Edge(n6b7, 29),
new Edge(n3a7, 12), };
n7b6.adjacencies = new Edge[]{  };
n7c6.adjacencies = new Edge[]{
new Edge(n10a7, 42),
new Edge(n10b7, 39),  };
n7a10.adjacencies = new Edge[]{
new Edge(n10a7, 42),
new Edge(n10b7, 39),
new Edge(n6a7, 32),
new Edge(n6b7, 29),
new Edge(n3a7, 12),   };
n7b10.adjacencies = new Edge[]{
new Edge(n3a7, 12),   };
n7a3.adjacencies = new Edge[]{
new Edge(n3a7, 12),
new Edge(n10a7, 42),
new Edge(n10b7, 39),
new Edge(n6a7, 32),
new Edge(n6b7, 29),  };
n8a5.adjacencies = new Edge[]{
new Edge(n5a8, 23),
new Edge(n5b8, 25),
new Edge(n5c8, 26),
new Edge(n9a8, 26),
new Edge(n9b8, 30), };
n8b5.adjacencies = new Edge[]{ };
```

```
n8a9.adjacencies = new Edge[]{
new Edge(n9a8, 26),
new Edge(n9b8, 30), };
n8b9.adjacencies = new Edge[]{
new Edge(n5a8, 23),
new Edge(n5b8, 25),
new Edge(n5c8, 26), };
n9a8.adjacencies = new Edge[]{
new Edge(n8a9, 25),
new Edge(n8b9, 27),
new Edge(n6a9, 36),
new Edge(n6b9, 30),
new Edge(n10a9, 25),
new Edge(n10b9, 28),  };
n9b8.adjacencies = new Edge[]{ };
n9a6.adjacencies = new Edge[]{
new Edge(n6a9, 36),
new Edge(n6b9, 30),
new Edge(n10a9, 25),
new Edge(n10b9, 28), };
n9b6.adjacencies = new Edge[]{
new Edge(n8a9, 25),
new Edge(n8b9, 27),  };
n9a10.adjacencies = new Edge[]{
new Edge(n10a9, 25),
new Edge(n10b9, 28),
new Edge(n8a9, 25),
new Edge(n8b9, 27), };
n9b10.adjacencies = new Edge[]{
new Edge(n6a9, 36),
```

```
new Edge(n6b9, 30), };
n10a9.adjacencies = new Edge[]{
new Edge(n9a10, 26),
new Edge(n9b10, 27),
new Edge(n7a10, 44),
new Edge(n7b10, 40),
new Edge(n11a10, 20), };
n10b9.adjacencies = new Edge[]{  };
n10a7.adjacencies = new Edge[]{
new Edge(n7a10, 44),
new Edge(n7b10, 40),
new Edge(n11a10, 20),  };
n10b7.adjacencies = new Edge[]{
new Edge(n9a10, 26),
new Edge(n9b10, 27), };
n10a11.adjacencies = new Edge[]{
new Edge(n11a10, 20),
new Edge(n7a10, 44),
new Edge(n7b10, 40),
new Edge(n9a10, 26),
new Edge(n9b10, 27),};
n11a10.adjacencies = new Edge[]{
new Edge(n10a11, 19),
new Edge(n4a11, 31),
new Edge(n4b11, 33),  };
n11a4.adjacencies = new Edge[]{
new Edge(n4a11, 31),
new Edge(n4b11, 33),
new Edge(n10a11, 19), };
AstarSearch(n1a2, n11a4);
```

```
List<Node> path = printPath(n11a4);
System.out.println("Path: " + path);}
public static List<Node> printPath(Node target) {
List<Node> path = new ArrayList<Node>();
for (Node node = target; node != null; node =
node.parent) {
path.add(node);}
Collections.reverse(path);
return path;}
public static void AstarSearch(Node source, Node goal)
{
Set<Node> explored = new HashSet<Node>();
PriorityQueue<Node> queue = new
PriorityQueue<Node>(134,new Comparator<Node>() {
//override compare method
public int compare(Node i, Node j) {
if (i.f_scores > j.f_scores) {
return 1;
} else if (i.f_scores < j.f_scores) {
return -1;
} else {
return 0;
}}});
//cost from start
source.g_scores = 0;
queue.add(source);
boolean found = false;
while ((!queue.isEmpty()) && (!found)) {
//the node in having the lowest f_score value
Node current = queue.poll();
```

```
explored.add(current);
//goal found
if (current.value.equals(goal.value)) {
found = true;}
//check every child of current node
for (Edge e : current.adjacencies) {
Node child = e.target;
double cost = e.cost;
double temp_g_scores = current.g_scores + cost;
double temp_f_scores = temp_g_scores + child.h_scores;
/*if child node has been evaluated and the newer
f_score is higher, skip*/
if ((explored.contains(child)) &&
(temp_f_scores >= child.f_scores)) {continue;}
else if ((!queue.contains(child)) ||
temp_f_scores < child.f_scores)) {
child.parent = current;
child.g_scores = temp_g_scores;
child.f_scores = temp_f_scores;
if (queue.contains(child)) {
queue.remove(child);}
queue.add(child);  }}}}}
class Node {
public final String value;
public double g_scores;
public final double h_scores;
public double f_scores = 0;
public Edge[] adjacencies;
public Node parent;
public Node(String val, double hVal) {
```

```
value = val;
h_scores = hVal;}
public String toString() {
return value; }}
class Edge {
public final double cost;
public final Node target;
public Edge(Node targetNode, double costVal) {
target = targetNode;
cost = costVal;    }}
```

Суть A*-алгоритму зводиться до визначення мінімальної величини із спектру значень функції $f(n)=g(n)+h(n)$. Тут $g(n)$ – відстань від стартової вершини (в нашій задачі це вершина $n1a2$) до поточної вершини n; $h(n)$– так звана евристична відстань. Взагалі значення функції $h(n)$ розраховується для кожного вузла графа і представляє собою відстань по прямій від стартової вершини до кінцевої (в нашому випадку це вершина $n11a4$). Проте в даній задачі маємо справу з особливою ситуацією, адже величини $g(n)$ розраховуються по-особливому, а саме на основі співвідношення (1). І величини цих значень зовсім не корелюють з геометричною відстанню між вузлами графа. Тому в цьому сенсі доцільно використати одне-єдине значення евристичної функції. Запустимо програму та отримаємо наступний результат:

Path: [a-Freedom str, a-Broad av, a-Valley str, a-Heroes str, a-Apple str, b-Brosdmay str , a-Long str]. (6)

Ми свідомо замінили у програмі `package IC` деякі назви вузлів на назви вулиць – щоб прокладений маршрут виглядав більш наочно і тому

отримали запис (6). Власне це і є t-оптимальний маршрут, виділений на рис.3 зеленим кольором. На противагу даному маршруту показано інший, виділений червоним. Між ділянках дороги $n1-n5$ та $n5-n6$ обидва згадані маршрути співпадають, а після вершини (перехрестя) $n6$ – розходяться. Маршрут, виділений червоним, буде видавати сучасний GPS-навігатор. Але, і це дуже важливо, вага такого маршруту буде складати величину 117 умовних одиниць (у.о.). В той самий час геометрично довший «зелений» маршрут має вагу лише 88 у.о. Саме значення у.о. якраз і є величинами, пропорційними часу проїзду ТЗ вздовж маршруту. Тобто доцільніше вибрати хоча і геометрично довший, проте по часу більш кращий маршрут, зображений на рис. 3 зеленим кольором. Дані щодо такого маршруту будуть передаватись водієві із ЦПКТ, поки ситуація на трасі руху не зміниться. Водій, отримавши алгоритм проїзду типу (6) без проблем дістанеться в любий міський пункт за мінімально короткий час. Проте, як тільки ситуація на трасі зміниться і наш «зелений» маршрут стане не оптимальним, то в такому разі програма обрахує новий маршрут та миттєво передасть його водієві. І так для всіх ТЗ по всьому місту. В результаті відбудеться цілковита синхронізація транспортних потоків, що приведе до повного зникнення заторів у транспортній мережі та дозволить кожному водієві прибувати до місця призначення за мінімально короткий проміжок часу. Таким чином, міський трафік перейде на якісно новий рівень.

Дуже важливо зауважити, що програма `package IC`, приведена вище, враховує всі вимоги ПДР. Адже згідно ПДР, автомобілі на проїзній частині дороги при проїзді перехресть повинні займати чітко визначені смуги руху – у відповідності із своїм маршрутом. Скажімо, із крайньої лівої смуги можна виконувати розворот та поворот наліво, із крайньої правої смуги – тільки поворот направо. Всі ці нюанси саме і враховує приведена програма. Тому у виразі (6) біля назв вулиць проставлені маркери смуг руху, по яких

має рухатись водій. Наприклад, запис «$b - Brosdmay\ str$» означає, що водій на вказаній вулиці має зайняти смугу b для того, щоб виконати маневр повороту направо (рис. 3). Це є «ноу-хау» і тому на ньому, зокрема, акцентується увага. Якщо звернутись до запису (6), то він цілком точно дозволяє супроводжувати водія по його маршруту, тому що програма заздалегідь передбачає можливість своєчасного перелаштування автомобіля (заздалегідь) на відповідну смугу руху, з якої маневр проїзду через перехрестя дозволений ПДР!

Тестування приведеної програми проводилось на графах з числом вершин до 150 та числом ребер 430. При цьому час виконання складав складав 3 с 354 мс. Взагалі, обчислювальна складність A^*- алгоритму лінійна $O(N)$, що робить його досить ефективним для задач з великим об'ємом вхідних даних (на відміну від згаданих алгоритмів Дейкстри та Флойда).

Висновки

Дане дослідження ставить за мету реальне вирішення проблеми трафіку у великих містах. Створена технологія, що реально дозволяє покращити трафік ТЗ у містах-мегаполісах.

Перед технічним впровадженням пропонованого проекту проведено імітаційне моделювання даної технології з допомогою програми візуального імітаційного моделювання AnyLogic 8.4.[17]. Результати моделювання показують цілковиту прийнятність пропонованого проекту для цілей технічного впровадження. Алгоритм допоможе позбутись проблеми кожного великого міста – заторів – та зробить для водіїв більш комфортне слідування по заявленому маршруту. Дана технологія, при втіленні її у життя, значно покращить трафік-клімат на вулицях міст та зіграє суттєву роль в організації руху міського транспорту в усьому світі.

ЛІТЕРАТУРА

1. Богуто Д.Г., Волинець В.І., Сомов О., Ярощук М. Автоматизована система керування рухом транспортних засобів в межах міста / Д.Г. Богуто, В.І. Волинець, О. Сомов, М. Ярощук // Вісник Харківського університету, серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління». – 2017.– вип.35. – С. 5-12.

2. Богуто Д.Г., Комаров В.Ф., Сомов О., Ярощук М. Інтелектуальний алгоритм управління міським трафіком транспортних засобів / Д.Г. Богуто, В.Ф. Комаров, О. Сомов, М. Ярощук // Вісник Харківського університету, серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління».– 2018.– вип.38. – С. 4-13.

3. Комаров В.Ф., Сомов О., Ярощук М. Інтелектуальне перехрестя/ В.Ф.Комаров, О. Сомов, М. Ярощук //Міжвузівський збірник «Наукові нотатки», Луцьк. – 2018.–Вип.63.- С 139-147.

4. Pat.US20150153176 Korea, Vehicle, vehicle cloud system, and data dissemination system / Hyuk Lim, In Shick Kim, Ryangsoo Kim; applicant Gwangju Institute Of Science and Technology; publ. 04.06.2015.

5. Rinaldi Marco, Viti Francesco. Exact and approximate route set generation for resilient partial observability in sensor location problems/Marco Rinaldi, Francesco Viti//Transportation Research Part B: Methodological. – 2017. – V.105. – № 11. – P. 86 – 119.

6. Gaurav Pandey, K. Ramachandra Rao, Dinesh Mohan. Modelling vehicular interactions for heterogeneous traffic flown using cellular automata with position preference/ Gaurav Pandey, K. Ramachandra Rao, Dinesh Mohan // J. Mod. Transport. – 2017. – V. 25. – №3. – P.163–177.

7. Memon A. A., Meng M., Wong Y. D., Lam S. H. Calibration of a rule-based intelligent network simulation model/ A. A.Memon, M. Meng, Y. D Wong., S. H. Lam // J. Mod. Transport. – 2016. – V. 24. – №1. – P.48–61.

8. Wu Wei, Liu Yang, Xu Yue, Wei Quanlun, Zhang Yi. Traffic Control Models

Based on Cellular Automata for At-Grade Intersections in Autonomous Vehicle Environment/ Wei Wu, Yang Liu, Yue Xu, Quanlun Wei, Yi Zhang // Journal of Sensors. –Volume 2017. – Article ID 9436054.– 6 pages.

9. Pat. CN104575066 China, Intelligent terminal based intelligent traffic light system and method/ Xu Chunmao Xu Jin; applicant Shanghai Bolter Digital Technology Co., LTD.; publ.29.04.2015.

10. Case studies for traffic solutions - Siemens [Електронний ресурс]. – Режим доступу: www.mobility.siemens.com/ solutions / case-studies-for-tr.

11. David K. Chang, Mitsuru Saito, Grant G. Schultz, Dennis L. Eggett. Use of Hi-resolution data for evaluating accuracy of traffic volume counts collected by microwave sensors/ K. David, Chang, Saito Mitsuru, Schultz G.Grant, Eggett L. Dennis // Journal of traffic and transportation engineering. – 2017. – V.4. – Is. 5. – P. 423-435.

12. Dazhi Sun,Jinpeng Lv,S. Travis Waller. In-depth analysis of traffic congestion using computational fluid dynamics (CFD) modeling meth/ Sun Dazhi, Lv Jinpeng, Waller S. Travis // Journal of Modern Transportation. – 2011.–V.1. – № 1. – P. 58-67.

13. Nair T.R. Gopalakrishnan, Sooda Kavitha. Comparison of Genetic Algorithm and Simulated Annealing Technique for Optimal Path Selection in Network Routing/ Gopalakrishnan Nair T.R., Kavitha Sooda // 2010. –arXiv: 1001.3920. Available at: http://arxiv.org/abs/1001.3920.

14. Ampountolas Konstantinos, Zheng Nan, Geroliminis Nikolas. Macroscopic modelling and robust control of bi-modal multi-region urban road networks/ Konstantinos Ampountolas, Nan Zheng, Nikolas Geroliminis // Transportation Research Part B: Methodological. – 2017. –V.104. – P.616–637.

15. Intelligent traffic light control [Електронний ресурс]. Режим доступу: https://www.ercim.eu/publication/Ercim_News/enw53/wiering.html

16.П'єзокристалічний датчик [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Piezoelectric_sensor.

17.Альфа-версия Any-Logic 8.4 [Електронний ресурс]. Режим доступу: https://www.anylogic.ru/blog/alfa-versiya-anylogic-8-4/.