Шифр «Трафік»

Конкурсна робота

«Як уникнути заторів у великих містах?»

Зміст

ВСТУП

Транспортний колапс – затори – є, як добре відомо, болючою проблемою міст-мегаполісів. Дане дослідження дозволяє в деякій мірі вирішувати сформульовану проблему завдяки застосування спеціального алгоритму у поєднанні із певними технічними засобами. Фактично, робота представляє собою автоматизовану інтелектуальну систему (АІС) регуляції дорожнього руху транспортних засобів (ТЗ) у великому місті. АІС здійснює прокладання оптимальних маршрутів для кожного ТЗ, що приймає участь у міському трафіку та взаємодіє із центром керування трафіком (ЦКТ).

Моделювання транспортної мережі міста здійснюється з допомогою зваженого орієнтованого планарного мультиграфа. У такому графі кожному ребру співставляється у відповідність вага, що змінюється синхронно із динамікою змін завантаженості автомобілями транспортних артерій міста. З допомогою використання A^*-алгоритму здійснюється прокладання оптимальних по часу (t-оптимальних) маршрутів для кожного автомобіля, що замовив такий маршрут.

Оскільки всі автомобілі (або переважна їх кількість) рухатимуться по t-оптимальних маршрутах, то відбудеться повна синхронізація транспортних потоків, що приведе до принципово нової якості і, як наслідок, до зникнення заторів (або до різкого їх зменшення) у транспортній мережі. Це дозволить кожному водієві прибувати до місця призначення за мінімально короткий час. Отже, міський трафік трансформується у принципово новий стан.

Розділ І. Аналіз найсучасніших літературних джерел

Робота [1] представляє собою оглядове фундаментальне дослідження, присвячене аналізу сучасних видів “розумних” транспортних систем (intelligent transportation systems). Розглядається широкий спектр питань, пов’язаних з

візуалізацією даних міського трафіку. Одним із найважливіших методів візуалізації таких даних є використання графів, що дають можливість наочно представляти топологію і геометрію розташування міських районів (в нашому дослідженні використання графів та пов'язаних з ними алгоритмів є якраз головним методом досліджень). Автори проводять також узагальнений аналіз мобільності динамічних міських потоків, що зводиться до наступного:

- Міські транспортні потоки та їх моніторинг;
- Людська динаміка в міському середовищі;
- Дорожні транспортні інциденти;
- Забруднення повітря.

Яскравим прикладом візуалізації є рис.1, де показані рівні завантаженості міських доріг – червоний колір представляє найбільшу завантаженість міської мережі [1].

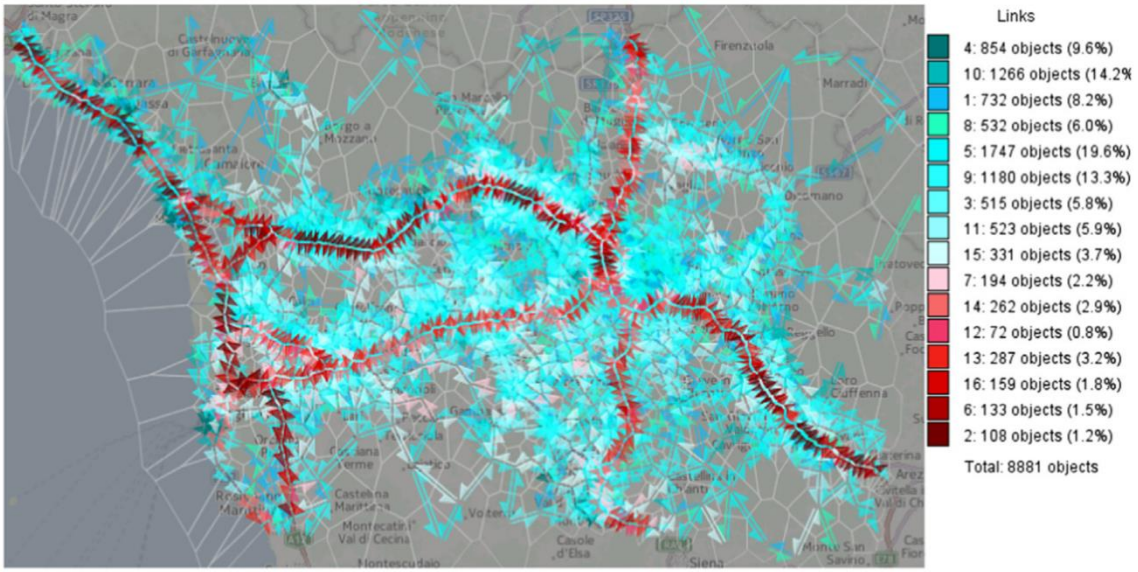

Рис.1. Візуалізація транспортних потоків в термінах значення швидкості ТЗ на міських дорогах (червоні кольори відповідають транспортним магістралям, де швидкість найменша [1]).

Характерною прикметою досліджень міських транспортних мереж є використання технології Vehicular Ad-hoc Networks (VANETs), детально описаної в [2]. Ця технологія використовується для збору даних про

завантаженість транспортних мереж та передачі їх до ЦКТ. Важливим застосуванням VANETs-технології являється навігація із зворотнім зв'язком (feedback-based navigation). Вона використовує спеціальні автомобілі (sensors vehicles) для збору інформації щодо ситуації на дорогах міста та передає цю інформацію до ЦКТ.

Спектр робіт присвячено використанню нейронних мереж для аналізу та прогнозу трафік-ситуацій на транспортних артеріях міст. В цьому відношенні заслуговує уваги робота [3], в якій розроблена модель, що дозволяє здійснювати короткотермінові прогнози (глибина прогнозу 5-10 хвилин) щодо поведінки транспортних потоків на автомагістралях міст. В дослідженні використовується сортувальний генетичний алгоритм типу II (NSGA-II), що використовує дві важливі обставини – оцінку трафіку та параметри його оптимізації. Подібне попередньому дослідження проведено у [4]. Тут використовується фільтр Калмана для передбачення завантаженості транспортних міських артерій. Застосовується технологія «зв'язаних автомобілів» – таких сучасних ТЗ, на яких встановлюються спеціальні пристрої, що дозволяють їм взаємодіяти з міською придорожньою інфраструктурою. В результаті такої «співпраці» можна здійснювати оцінку різноманітних параметрів транспортних потоків – середню швидкість руху, час простою на завантажених перехрестях, час поїздки і т.п. Алгоритм не використовує дані, отримувані іншими подібного роду методиками із петлевих детекторів та відеокамер, а лише дані із «зв'язаних автомобілів», що свідчить про дешевизну такого підходу.

Ефективне використання даних, зібраних із різного роду дорожніх сенсорів, є суттєвою проблемою у зв'язку із процесами втрати деяких даних при передачі їх на ПКТ. Цій проблемі присвячена стаття [5], що пропонує метод передачі даних з високим ступенем надійності і ефективності. У порівнянні з іншими подібного роду технологіями якість передачі таких даних зростає на 87%. Автори вводять у розгляд спеціальну матрицю декомпозиції, що використовує принцип стиснення потоку даних для підрахунку показників трафіку.

Прогнозування трафіку є важливим аспектом у багатьох сферах, пов’язаних із дорожнім рухом. Такій темі приділяють увагу автори роботи [6], де проаналізовані три моделі, відібрані шляхом аналізу величезної кількості методів, які здійснюють короткотермінові прогнози щодо трафік-ситуації. Досліджуються італійські транспортні мережі з використанням непараметричної К-нейронної регресивної моделі. Вивчаються щотижневі та щомісячні варіації трафік-параметрів на основі усереднених даних такого типу. В статті [7] досліджується проблема інтегрованого трафік-контролю за транспортними мережами шляхом використання макроскопічної моделі, що являє собою математичну модель трафік-потоку ТЗ. Аналізуються щільність, інтенсивність та швидкість потоку автомобілів. Запропоновано інтегрований контрольний алгоритм для розв’язування поставленої проблеми. Суть алгоритму зводиться до вибору оптимальних маршрутів у міській транспортній мережі.

У проблемі вирішення принципових питань дорожнього трафіку вагому роль відіграють міжнародні винаходи, яким автор даної роботи приділяє велику увагу. У цьому відношенні прототипом нашого дослідження є винахід [8], що представляє собою технологію, пов’язану із застосуванням мобільних телефонів водіями ТЗ. На перехрестях міста встановлюються спеціальні пристрої, що реєструють транспортні потоки та передають зібрану інформацію до ЦКТ. Далі інформація через інтернет поступає до кожного водія з метою вибору ним оптимального маршруту.

Вирішенню проблеми оптимізації трафіку шляхом оцінки потоку ТЗ через кожне міське перехрестя присвячений винахід [9]. Система працює в режимі реального часу і тому динамічно контролює та регулює рух на міських трасах шляхом створення режиму зеленої хвилі. У [10] застосовується метод аналізу часової послідовності скритого ланцюга Маркова. Як головні оптимізаційні критерії використовуються час поїздки та швидкість ТЗ, що в результаті приводить до знаходження найкращого варіанту поїздки.

Взаємодія між автомобілями на міських трасах (і не тільки) відіграє важливу роль в організації координованої поведінки величезної кількості динамічних

об’єктів, особливо при їх скупченості, як це має місце у містах-мегаполісах. Саме проблема такої взаємодії на основі стандарту IEEE 802.11p розглядається у роботі [11]. Взаємодія типу «Автомобіль – Автомобіль» повинна забезпечуватись надійною системою комунікації, яку представляє собою згаданий вище стандарт.

Транспортні мережі працюють в різних режимах [12]. Знання цих режимів та переходи між ними є важливим фактором, що визначає сутність роботи транспортної мережі. Перевантаження мережі веде до заторів. Щоб уникнути таких проблем потрібно детально дослідити принципи функціонування транспортних мереж. В зазначеній роботі досліджується стохастичний та динамічний перехід між різними режимами трафіку. Робота [13] спрямована на вирішення конкретної проблеми покращення трафіку у міському середовищі, де часто виникають затори завдяки невмілому режиму керування трафіком. Аналогічній проблемі присвячено дослідження [14], що представляє собою елементи технології, придатної до практичного використання. Акцент зроблено на чотири напрямки – теорію графів, інтелектуальний A^*-алгоритм, інформаційні технології та технічні засоби реєстрації ТЗ (сенсори). Поєднання цих компонентів дозволяє зробити реальний крок у вирішенні найголовнішої проблеми великих міст – проблеми заторів. Реальну технологію вирішення згаданої проблеми пропонують також автори винаходу [15]. Дана інновація відноситься до інтелектуальної системи керування режимом роботи світлофорів шляхом двостороннього обміну інформацією з ЦКТ.

Розділ 2. Алгоритм оптимального міського маршруту

Організація роботи інтелектуальної системи регулювання трафіку складається, так би мовити, із двох компонент. З однієї сторони – це технічне забезпечення роботи АІС. З другого боку – це програмний алгоритм.

Технічно робота АІС організована наступним чином. В районі перехрестя у полотно проїзної частини перпендикулярно до поздовжньої вісі дороги

монтуються п’єзокристалічні сенсори [14], що реагують на тиск, спричинюваний колісною парою автомобіля, який перетинає смугу, де змонтовані ці пристрої. Електричний сигнал, спричинений стиском п’єзосенсора, поступає на вимірювально-обчислювальний комплекс (ВОК). ВОК, таким чином, реєструє число сигналів, що поступають з різних напрямків перехрестя. Спектр таких чисел пропорційний числу автомобілів, які перетнули перехрестя. ВОК передає отримані сигнали на ЦКТ.

Програмне забезпечення представляє собою алгоритм, що реалізує прокладання t-оптимальних маршрутів у графі, який моделює транспортну мережу міста. При цьому самим принциповим питанням є вибір ваг ребер графа. Ці ваги повинні бути змінними у часі з таким розрахунком щоб відповідати змінам у завантаженості ТЗ транспортної мережі міста. Для прокладання маршрутів використовується A^*-алгоритм, який по своїй сутності є оптимізаційним. Наше завдання полягає у тому, щоб здійснити оптимізацію по часу проїзду кожним автомобілем свого вибраного маршруту, адже принциповим питанням логістики міського трафіку є саме час проїзду ТЗ , а не відстань, пройдена ТЗ. 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. Але реально корисним для кожного автомобіля (водія) є саме t-оптимальний маршрут. У великому місті число ТЗ, що одночасно курсують вулицями, може бути більше мільйона. Тому важливо забезпечити оптимальні умови проїзду для всіх автомобілів мегаполіса, що приймають участь у трафіку. Проблема далеко не нова, але не вирішена та набуває все більшої гостроти, особливо у великих містах. Саме на вирішення такої проблеми спрямована представлена робота.

Для ефективного регулювання трафіку потрібно спочатку реєструвати ТЗ на кожному окремому перехресті. З цією метою використовуються вхідні та вихідні сенсори. Дані з цих сенсорів поступають для реєстрації завантаженості смуг руху між сусідніми перехрестями.

Розглянемо частину транспортної мережі міста Толедо (Канада), представлену на рис.2. Потрібно створити модель транспортної мережі цього мікрорайону міста з урахуванням числа смуг руху на кожній ділянці дороги. Така модель представляє собою граф, представлений на рис.3. Вказаний граф

Рис.2. Мікрорайон міста Толедо (Канада). Пунктиром виділена частина району, що моделюється графом, представленим на рис. 3.

моделює не просто дорогу між перехрестями, а відтворює окремі смуги руху, що є принципово важливим моментом, оскільки завантаженості різних смуг, як правило, суттєво різняться. Тому необхідно реєструвати ці компоненти трафіку окремо. Мультиграф, представлений на рис.3., зважений – кожне ребро отримує вагу, що постійно змінюється (про це мова буде йти далі). Вузли графа імітують перехрестя доріг, а направлені дуги (ребра) символізують собою смуги руху. Надписи «START» та «FINISH» представляють собою відповідно початкову та кінцеву позиції автомобіля, що замовив маршрут між вказаними пунктами.

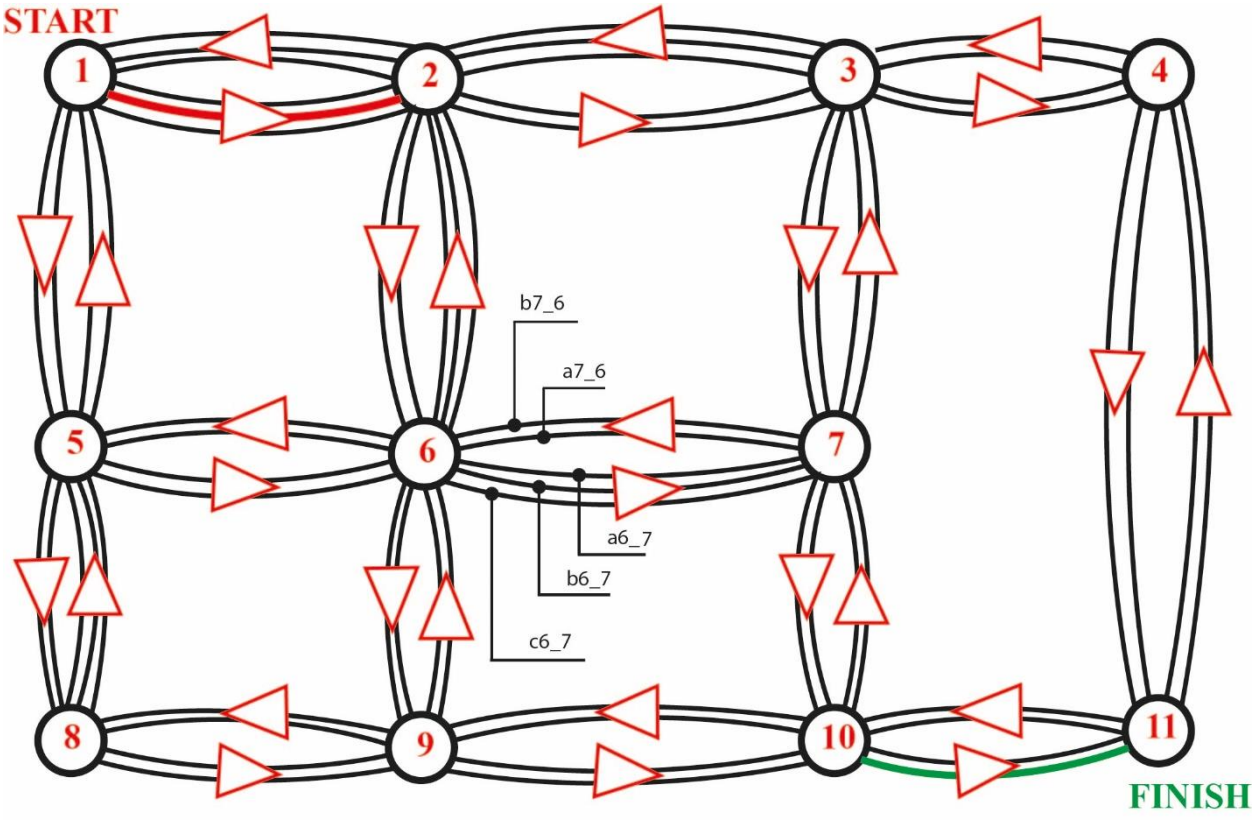

Рис.3. Зважений орієнтований планарний мультиграф, що моделює частину транспортної мережі міста Толедо. Кожному ребру графа приписується змінна у часі (динамічна) вага, що розраховується для прокладання t-оптимального маршруту. В центрі приведені позначення ребер, зміст яких описаний у тексті. Стартова та фінішна позиції ТЗ виділені.

На рис.4 зображено два сусідніх перехрестя A і B. Розглянемо автомобілі, що рухаються від перехрестя A до перехрестя B вздовж ділянки дороги, яку позначимо $A_h B_h$. Ця ділянка дороги має три смуги руху. Найменування смуг руху починається від осьової розділювальної лінії – крайня ліва смуга позначається a, середня – b і крайня права – c (і так далі по алфавіту, якщо смуг більше). Показано траєкторію руху двох автомобілів (№1 і №4), що під'їхали до перехрестя A. Обидва ці ТЗ на ділянці руху $A_h B_h$ реєструється вхідним сенсором A_h та вихідними сенсорами B_h^a та B_h^c відповідно. Взагалі на ділянці дороги $A_h B_h$ траєкторії руху автомобілів, як правило, пересікаються.

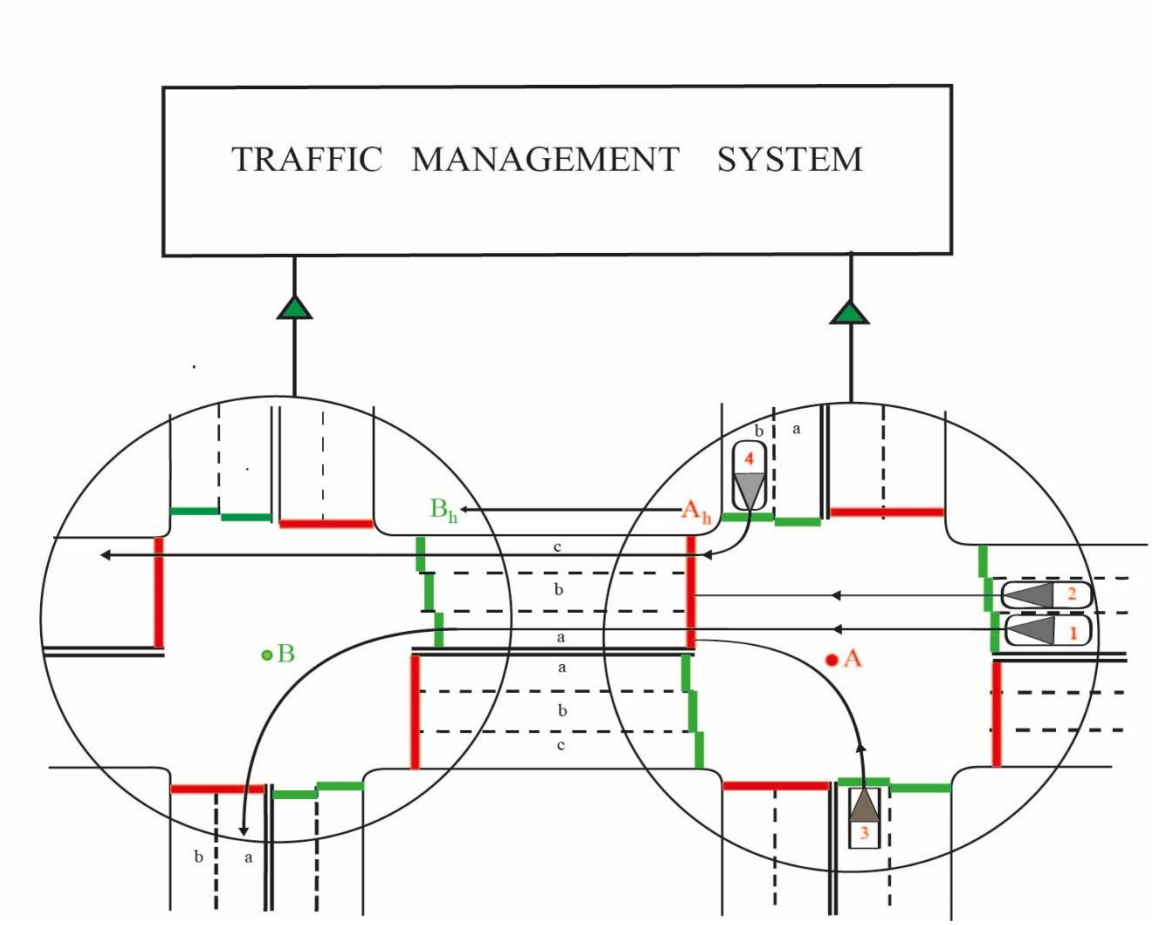

Рис.4. Зображено два сусідніх перехрестя А і В та ділянку дороги між ними. Автомобілі, що під'їхали до перехрестя А, позначені 1,2,3 і 4. Маршрути цих ТЗ пролягають вздовж ділянки дороги $A_h B_h$ в напрямку перехрестя В. Показано маршрути автомобілів 1 і 4[14].Дані від сенсорів обох видів поступають до ЦКТ (Transportation Management System).

Отже, завантаженість ділянки дороги $A_h B_h$ між сусідніми перехрестями вимірюється вхідним сенсором перехрестя А та вихідними сенсорами перехрестя В. Потім ці дані співставляються, що робить можливим говорити про динаміку руху ТЗ на ділянці дороги $A_h B_h$. Взагалі, вхідні та вихідні сенсори являють собою пристрої, що монтуються у полотно дороги (наприклад, це може бути п'єзокристалічний сенсор типу RoadTrax BL [14]). Їх принцип роботи базується на фізичному явищі п'єзоефекту – при наїзді колісної пари автомобіля на зону

розташування сенсора генерується електричний імпульс, який реєструється на ЦКТ. Таким чином, вхідні та вихідні сенсори видають спектр величин $N_{A_h B_h}$ та $n^{i}_{A_h B_h}$. Перша величина – це загальне число автомобілів, що в'їхали на ділянку дороги між сусідніми перехрестями (на рис.4 це ділянка дороги $A_h B_h$) на протязі часу горіння зеленої фази світлофора для автомобілів перехрестя A, маршрут яких пролягає у напрямку $A_h B_h$, а друга – це число автомобілів, що виїхали із смуги i цієї ділянки дороги на сусідньому перехресті B також за час горіння зеленої фази світлофора.

На ділянці дороги $A_h B_h$ автомобілі розподіляються (перелаштовуються) по смугах руху у відповідності із своїм подальшим маршрутом, що розраховується програмно (лістинг 1). При виїзді із ділянки дороги одного напрямку $A_h B_h$ потрібно зареєструвати всі групи автомобілів, що зайняли відповідні смуги руху. Очевидно, що число вихідних сенсорів дорівнює числу смуг проїзної частини дороги одного напрямку (на рис. 4 число таких сенсорів дорівнює трьом у відповідності з числом смуг; позначимо число таких смуг через l). Отже, кожна смуга оснащається власним вихідним сенсором.

Будь-який маршрут складається із певної послідовності 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 сенсорів – а значить і весь маршрут контролюється. Таким чином, є можливість прокладання оптимальних маршрутів для всіх ТЗ, що здійснюють рух по місту та координують свої маршрути з ЦКТ. Подібного роду координація відбувається з допомогою GPS-навігаторів та GPS-трекерів [11].

Динаміка руху вздовж ділянки дороги $A_h B_h$ обумовлена співвідношенням між величиною ТЗ, що в'їхали за час, рівний фазі горіння зеленого світла світлофору та виїхали із даної ділянки дороги за аналогічний час. Технічно процес організований наступним чином: сенсор A_h (вхідний сенсор) реєструє автомобілі (точніше число колісних пар), які в'їжджають на ділянку дороги

A_hB_h зі сторони перехрестя A (рис.4). В свою чергу, комплекс вихідних сенсорів на перехресті B реєструє автомобілі, що виїхали за межі цієї ділянки. Відношення цих величин якраз і свідчить про динаміку руху на конкретній ділянці дороги. Запишемо відношення

$$D_h = \sum_{i=1}^{l} N^i_{A_hB_h} / \sum_{i=1}^{l} n^i_{A_hB_h} = N_{A_hB_h} / \sum_{i=1}^{l} n^i_{A_hB_h} . \qquad (1)$$

Тут $N^i_{A_hB_h}$ – число автомобілів, що в'їхали на смугу i ділянки дороги A_hB_h, яка складається із l смуг, на протязі часу горіння зеленої фази світлофора; $n^i_{A_hB_h}$ – число автомобілів, що виїхали із вказаної смуги за аналогічний час.

Зрозуміло, що чим ближчим є D_h до одиниці, тим вищою буде динаміка руху автомобілів по ділянці дороги A_hB_h. Наше завдання полягає в тому, щоб знайти у павутинні міської транспортної мережі смуги руху, які складають простий ланцюг та задовольняють умові оптимальності маршруту по часу, тобто формують t-оптимальний маршрут:

$$\sum_{h=1}^{f} (N^i_{A_hB_h} / (n^i_{A_hB_h}))L_{A_hB_h} \to min. \qquad (2)$$

Тут f означає кількість смуг руху, які послідовно формують t-оптимальний маршрут; $L_{A_hB_h}$ – довжина смуги руху h.

У вираз (2) час t у явному вигляді не входить! Тоді виникає запитання: чому ця умова відповідає умові оптимальності маршруту по часу? Проаналізуємо відношення

$$N^i_{A_hB_h} / n^i_{A_hB_h} , \qquad (3)$$

що фігурує у мультиплікаті виразу (2). Значення величини (3) дуже сильно впливає на тривалість поїздки по маршруту, оскільки реально може коливатись

у межах $1 \div 10$ або навіть у більшому діапазоні. Припустимо, дане відношення в середньому дорівнює трьом. Це фактично значить, що на кожному перехресті автомобіль затримується на три періоди (цикли) переключення світлофора. Якщо один цикл переключення світлофора складає орієнтовно 70 с, то затримка на кожному перехресті буде 210 с, а тоді по всьому маршруту отримаємо дуже значну величину $210 \times f$ (с) . В ідеалі величина (3) повинна складати значення близьке до одиниці: при такому розкладі час поїздки скоротиться втричі! І саме представлений далі алгоритм (лістінг 1) знаходить маршрут з мінімальною вагою; іншими словами час, затрачений на поїздку по такому маршруту, буде найменшим із всіх можливих. Ось чому умова (2) є критерієм оптимальності по часу.

З метою створення умов для виконання оптимальності виду (2) застосуємо A*-алгоритм, який дозволяє прокласти маршрут у графі між двома заданими вершинами (іншими словами між заданим початковим та кінцевим положеннями маршруту).

Відношення (1) дає, так би мовити, інтегральну характеристику щодо динаміки руху на ділянці дороги одного напрямку $A_h B_h$. Проте, цілком можливою є ситуація, коли на вказаній ділянці дороги динаміка руху ТЗ буду різною – деякі смуги руху будуть задовольняти умові високої динамічності, тобто величина (3) буде близькою до одиниці:

$$N^{i}_{A_h B_h} \Big/ n^{i}_{A_h B_h} \approx 1 \ , \qquad (4)$$

а інші, навпаки, – для цих смуг руху тієї ж ділянки дороги може виконуватиметься умова

$$N^{i}_{A_h B_h} / n^{i}_{A_h B_h} \ll 1. \qquad (5)$$

Останнє свідчить або про заблокованість відповідної смуги руху або про дуже низьку динаміку руху ТЗ на цій смузі руху. Відповідно вага ребра графа (іншими словами, завантаженость смуги руху i ділянки дороги $A_h B_h$) здобуває великих

значень. Таким чином, потрібно сепаративно визначати пропускну здатність для кожної окремої смуги руху. Лише при умові

$$N_{A_hB_h} \approx \sum_{i=1}^{l} n^{i}_{A_hB_h} \qquad (6)$$

всі смуги i вільні для трафіку.

Але, як правило, умова (6) виконується не часто. Ті смуги руху, для яких будуть виконуватись умови типу (4), будуть включатись програмою, що реалізує A*-алгоритм, у трафік (лістінг 1); навпаки, у випадку виконання умови (5), відповідні смуги руху не будуть формувати маршрут руху ТЗ. Простіше кажучи, якщо на певній смузі руху утворився затор, то програма, представлена далі, омине таку ділянку дороги.

Вхідні сенсори обраховують лише сумарні величини виду

$$\sum_{i=1}^{l} N^{i}_{A_hB_h} = N_{A_hB_h} \ . \qquad (7)$$

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

Проте вагома роль вхідних сенсорів полягає також у тому, щоб реєструвати переїзд ТЗ на нову смугу руху та, відповідно, обраховувати маршрут по-новому (про це пізніше).

Програма, представлена на лістингу 1, складена так, що дозволяє прокладати маршрути при обов'язковому виконанні вимог ПДР. Скажімо, для водія, що знаходиться на смузі руху $c6\_7$ (рис.3) згадана програма може прокласти маршрут лише у напрямках смуг $a7\_10$ та $b7\_10$, адже автомобіль, згідно ПДР, із крайньої правої смуги може здійснювати поворот лише направо (якщо інше не передбачено дорожньою розміткою).

Звернемо далі увагу, що, наприклад, число можливих маршрутів із смуги $b1\_2$ (START) до смуги $b10\_11$ (FINISH) може досягати сотні, і тому важливо із спектру можливих варіантів вибрати один-єдиний маршрут оптимальний маршрут.

На мові теорії графів вираз (2) 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 Дейкстри чи Флойда[13]. Але для нашої задачі доцільно скористатись A^*-алгоритмом, який прокладає оптимальний маршрут між двома вибраними вершинами графа та володіє незначною алгоритмічною складністю типу $O(N)$, де N – об'єм вхідних даних. Принцип дії цього алгоритму зводиться до визначення мінімальної величини із спектру значень функції $f(n) = g(n) + h(n)$, де $g(n)$ являє собою відстань від стартової позиції до поточної вершини n; $h(n)$ – так звана евристична відстань. В нашій задачі значення функції $h(n)$ розраховується для кожного вузла графа і представляє собою відстань по прямій від поточної вершини n до кінцевої.

Канал «ЦКТ ↔ Автомобіль» працює в режимі реального часу, постійно оновлюючи дані про завантаженість ділянок дороги між перехрестями, базуючись на даних, отримуваних із перехресть. Далі ЦКТ на основі роботи програми (лістінг 1) прокладає оптимальний трафік кожному ТЗ. Звичайно ж, представлена ділі програма реалізує прокладання маршрутів лише для групи ТЗ, що знаходяться між перехрестями 1та 2, оскільки евристика (функція $h(n)$) для всіх цих автомобілів однакова). Але місто є стаціонарним об'єктом і тому скласти програми типу приведеної знизу з урахуванням евристик для кожної пари вузлів не складає ніякої проблеми.

Java-програма, яка прокладає t-оптимальний маршрут у графі, представленому на рис.3, виглядає наступним чином (автор приносить вибачення за надто протяжний код програми у зв'язку із значним числом вершин та ребер у графі рис.3):

Лістинг 1. Оптимальний маршрут

```
package TRAFFIC;
import java.util.*;
public class AstarSearchAlgo {


        static Node a1_2 = new Node("a1_2-Liberty avenue", 962);
        static Node b1_2 = new Node("b1_2-Liberty avenue", 962);
        static Node c1_2 = new Node("c1_2-Liberty avenue", 962);
```

```
static Node a1_5 = new Node("a1_5-Freedom street", 1002);
static Node b1_5 = new Node("b1_5-Freedom street", 1002);
static Node a2_1 = new Node("a2_1-Liberty avenue", 962);
static Node b2_1 = new Node("b2_1-Liberty avenue", 962);
static Node c2_1 = new Node("c2_1-Liberty avenue", 962);
static Node a2_6 = new Node("a2_6-Heroes street", 733);
static Node b2_6 = new Node("b2_6-Heroes street", 733);
static Node a2_3 = new Node("a2_3-Broad street", 753);
static Node b2_3 = new Node("b2_3-Broad street", 753);
static Node a3_2 = new Node("a3_2-Broad street", 753);
static Node b3_2 = new Node("b3_2-Broad street", 753);
static Node c3_2 = new Node("c3_2-Broad street", 753);
static Node a3_4 = new Node("a3_4-Short street", 658);
static Node b3_4 = new Node("b3_4-Short street", 658);
static Node a3_7 = new Node("a3_7-Valley street", 504);
static Node b3_7 = new Node("b3_7-Valley street", 504);
static Node a4_3 = new Node("a4_3-Short street", 658);
static Node b4_3 = new Node("b4_3-Short street", 658);
static Node a4_11 = new Node("a4_11-Long street", 325);
static Node b4_11 = new Node("b4_11-Long street", 325);
static Node a5_1 = new Node("a5_1-Freedom street", 1002);
static Node b5_1 = new Node("b5_1-Freedom street", 1002);
static Node a5_6 = new Node("a5_6-Middle street", 777);
static Node b5_6 = new Node("b5_6-Middle street", 777);
static Node a5_8 = new Node("a5_8-Victory street", 887);
static Node b5_8 = new Node("b5_8-Victory street", 887);
static Node a6_5 = new Node("a6_5-Middle street", 887);
static Node b6_5 = new Node("b6_5-Middle street", 887);
static Node a6_2 = new Node("a6_2-Heroes street", 733);
static Node b6_2 = new Node("b6_2-Heroes street", 733);
static Node c6_2 = new Node("c6_2-Heroes street", 733);
static Node a6_7 = new Node("a6_7-Happy street", 523);
static Node b6_7 = new Node("b6_7-Happy street", 523);
static Node c6_7 = new Node("c6_7-Happy street", 523);
static Node a6_9 = new Node("a6_9-Joy street", 569);
static Node b6_9 = new Node("b6_9-Joy street", 569);
static Node a7_6 = new Node("a7_6-Happy street", 523);
static Node b7_6 = new Node("b7_6-Happy street", 523);
static Node a7_3 = new Node("a7_3-Valley street", 504);
static Node b7_3 = new Node("b7_3-Valley street", 504);
static Node a7_10 = new Node("a7_10-Trump street", 289);
static Node b7_10 = new Node("b7_10-Trump street", 289);
static Node a8_5 = new Node("a8_5-Victory street", 887);
static Node b8_5 = new Node("b8_5-Victory street", 887);
static Node c8_5 = new Node("c8_5-Victory street", 887);
static Node a8_9 = new Node("a8_9-Lincoln street", 713);
static Node b8_9 = new Node("b8_9-Lincoln street", 713);
static Node a9_6 = new Node("a9_6-Joy street", 569);
static Node b9_6 = new Node("b9_6-Joy street", 569);
static Node b9_8 = new Node("b9_8-Lincoln street", 713);
static Node a9_8 = new Node("a9_8-Lincoln street", 713);
static Node a9_10 = new Node("a9_10-Kennedy street", 400);
static Node b9_10 = new Node("b9_10-Kennedy street", 400);
static Node a10_7 = new Node("a10_7-Trump street", 289);
static Node b10_7 = new Node("b10_7-Trump street", 289);
static Node a10_9 = new Node("a10_9-Kennedy street", 400);
static Node b10_9 = new Node("b10_9-Kennedy street", 400);
static Node a10_11 = new Node("a10_11-Apple street", 125);
static Node b10_11 = new Node("b10_11-Apple street", 125);
static Node a11_10 = new Node("a11_10-Apple street", 125);
static Node b11_10 = new Node("b11_10-Apple street", 125);
static Node a11_4 = new Node("a11_4-Long street", 325);
static Node b11_4 = new Node("b11_4-Long street", 325);
```

```
        static Node nodes[] = new Node[]{
a1_2, b1_2,c1_2, a1_5, b1_5, a2_1, b2_1,c2_1,a2_6,b2_6, a2_3, b2_3,
a3_2,b3_2,c3_2,a3_4,b3_4,a3_7,b3_7,a4_3,b4_3,a4_11,b4_11,a5_1,
b5_1,a5_6,b5_6,a5_8,b5_8,a6_5,b6_5,a6_2,b6_2,c6_2,a6_7,b6_7,c6_7,
a6_9,b6_9,a7_6,b7_6,a7_3,b7_3,a7_10,b7_10,a8_5,b8_5,c8_5,a8_9,
b8_9,a9_6,b9_6,b9_8,a9_8,a9_10,b9_10,a10_7,b10_7,a10_9,b10_9,a10_11,
b10_11, a11_10,b11_10,a11_4,b11_4

 };

public static void initGraph() {

        Random k = new Random();

        int L2_1 = 320;
        double wa2_1 = (1 + (double) k.nextInt(10)) * L2_1;
        double wb2_1 = 1 + (double) k.nextInt(10)  * L2_1;
        double wc2_1 = 1 + (double) k.nextInt(10) * L2_1;
        a1_2.adjacencies = new Edge[]{
                new Edge(a2_1, wa2_1),
                new Edge(b2_1, wb2_1),
                new Edge(c2_1, wc2_1),};

        int L2_3 = 340;
        double wa2_3 = (1 + (double) k.nextInt(10)) * L2_3;
        double wb2_3 = (1 + (double) k.nextInt(10))  * L2_3;
        b1_2.adjacencies = new Edge[]{
                new Edge(a2_3, wa2_3),
                new Edge(b2_3, wb2_3), };

        int L2_6 = 390;
        double wa2_6 = (1 + (double) k.nextInt(10))  * L2_6;
        double wb2_6 = (1 + (double) k.nextInt(10))  * L2_6;
        c1_2.adjacencies = new Edge[]{
                new Edge(a2_6, wa2_6),        };

        int L5_1 = 390;
        double wa5_1 = (1 + (double) k.nextInt(10))  * L5_1;
        double wb5_1 = (1 + (double) k.nextInt(10))  * L5_1;
        int L5_6 = 342;
        double wa5_6 = (1 + (double) k.nextInt(10)) * L5_6;
        double wb5_6 = (1 + (double) k.nextInt(10))  * L5_6;
        int L5_8 = 295;
        double wa5_8 = (1 + (double) k.nextInt(10))  * L5_8;
        double wb5_8 = (1 + (double) k.nextInt(10))  * L5_8;
        a1_5.adjacencies = new Edge[]{
                new Edge(a5_1, wa5_1),
                new Edge(b5_1, wb5_1),
                new Edge(a5_6, wa5_6),
                new Edge(b5_6, wb5_6),
                new Edge(a5_8, wa5_8),
                new Edge(b5_8, wb5_8),  };

        b1_5.adjacencies = new Edge[]{
                new Edge(a5_8, wa5_8),
                new Edge(b5_8, wb5_8), };

        int L1_5 = 390;
        double wa1_5 = (1 + (double) k.nextInt(10))  * L1_5;
        double wb1_5 = (1 + (double) k.nextInt(10))  * L1_5;
        a5_1.adjacencies = new Edge[]{
                new Edge(a1_5, wa1_5),
                new Edge(b1_5, wb1_5), };
```

```
int L1_2 = 320;
double wa1_2 = (1 + (double) k.nextInt(10))   * L1_2;
double wb1_2 = (1 + (double) k.nextInt(10))   * L1_2;
double wc1_2 = (1 + (double) k.nextInt(10))   * L1_2;
b5_1.adjacencies = new Edge[]{
        new Edge(a1_2, wa1_2),
        new Edge(b1_2, wb1_2),
        new Edge(c1_2, wc1_2), };


a2_1.adjacencies = new Edge[]{
        new Edge(a1_2, wa1_2),
        new Edge(b1_2, wb1_2),
        new Edge(c1_2, wc1_2),
        new Edge(a1_5, wa1_5),
        new Edge(b1_5, wb1_5), };

b2_1.adjacencies = new Edge[]{
        new Edge(a2_1, wa2_1),
        new Edge(c2_1, wc2_1), };

c2_1.adjacencies = new Edge[]{
        new Edge(a2_1, wa2_1),
        new Edge(b2_1, wb2_1),  };

int L3_2 = 340;
double wa3_2 = (1 + (double) k.nextInt(10))   * L3_2;
double wb3_2 = (1 + (double) k.nextInt(10))   * L3_2;
double wc3_2 = (1 + (double) k.nextInt(10))   * L3_2;
int L3_4 = 210;
double wa3_4 = (1 + (double) k.nextInt(10))   * L3_4;
double wb3_4 = (1 + (double) k.nextInt(10))   * L3_4;
a2_3.adjacencies = new Edge[]{
        new Edge(a3_2, wa3_2),
        new Edge(b3_2, wb3_2),
        new Edge(c3_2, wc3_2),
        new Edge(a3_4, wa3_4),
        new Edge(b3_4, wb3_4),};


int L3_7 = 295;
double wa3_7 = 1 + (double) k.nextInt(5)   * L3_7;
double wb3_7 = 1 + (double) k.nextInt(5)   * L3_7;
b2_3.adjacencies = new Edge[]{
        new Edge(a3_4, wa3_4),
        new Edge(b3_4, wb3_4),
        new Edge(a3_7, wa3_7),
        new Edge(b3_7, wb3_7),  };

a3_2.adjacencies = new Edge[]{
        new Edge(a2_3, wa2_3),
        new Edge(b2_3, wb2_3),
        new Edge(a2_6, wa2_6),
        new Edge(b2_6, wb2_6), };

b3_2.adjacencies = new Edge[]{
        new Edge(a2_1, wa2_1),
        new Edge(b2_1, wb2_1),
        new Edge(c2_1, wc2_1), };

c3_2.adjacencies = new Edge[]{
```

```
        new Edge(a3_2, wa3_2),
        new Edge(b3_2, wb3_2), };

int L6_2 = 390;
double wa6_2 = (1 + (double) k.nextInt(10)) * L6_2;
double wb6_2 = (1 + (double) k.nextInt(10))  * L6_2;
double wc6_2 = (1 + (double) k.nextInt(10))  * L6_2;
int L6_7 = 270;
double wa6_7 = (1 + (double) k.nextInt(10))  * L6_7;
double wb6_7 = (1 + (double) k.nextInt(10))  * L6_7;


double wc6_7 = 1 + k.nextInt(5)  * L6_7;
int L6_9 = 290;
double wa6_9 = (1 + (double) k.nextInt(10))  * L6_9;
double wb6_9 = (1 + (double) k.nextInt(10))  * L6_9;
a2_6.adjacencies = new Edge[]{
        new Edge(a6_2, wa6_2),
        new Edge(b6_2, wb6_2),
        new Edge(c6_2, wc6_2),
        new Edge(a6_7, wa6_7),
        new Edge(b6_7, wb6_7),
        new Edge(c6_7, wc6_7),
        new Edge(a6_9, wa6_9),
        new Edge(b6_9, wb6_9),  };


int L6_5 = 342;
double wa6_5 = (1 + (double) k.nextInt(10)) * L6_5;
double wb6_5 = (1 + (double) k.nextInt(10))  * L6_5;
b2_6.adjacencies = new Edge[]{
        new Edge(a6_5, wa6_5),
        new Edge(b6_5, wb6_5),
        new Edge(a6_9, wa6_9),
        new Edge(b6_9, wb6_9), };

a6_2.adjacencies = new Edge[]{
        new Edge(a2_1, wa2_1),
        new Edge(b2_1, wb2_1),
        new Edge(c2_1, wc2_1),
        new Edge(a2_6, wa2_6),
        new Edge(b2_6, wb2_6), };

b6_2.adjacencies = new Edge[]{
        new Edge(a6_2, wa6_2),
        new Edge(c6_2, wc6_2), };

c6_2.adjacencies = new Edge[]{
        new Edge(a2_3, wa2_3),
        new Edge(b2_3, wb2_3),};

double wa4_3 = (1 + (double) k.nextInt(10)) * L3_4;
double wb4_3 = (1 + (double) k.nextInt(10))  * L3_4;
a3_4.adjacencies = new Edge[]{
        new Edge(a4_3, wa4_3),
        new Edge(b4_3, wb4_3), };
int L4_11 = 650;
double wa4_11 = (1 + (double) k.nextInt(10))  * L4_11;
double wb4_11 = (1 + (double) k.nextInt(10)) * L4_11;

b3_4.adjacencies = new Edge[]{
        new Edge(a4_11, wa4_11),
        new Edge(b4_11, wb4_11), };
```

```
a4_3.adjacencies = new Edge[]{
        new Edge(a3_2, wa3_2),
        new Edge(b3_2, wb3_2),
        new Edge(c3_2, wc3_2),
        new Edge(a3_4, wa3_4),
        new Edge(b3_4, wb3_4),
        new Edge(a3_7, wa3_7),
        new Edge(b3_7, wb3_7), };

b4_3.adjacencies = new Edge[]{
        new Edge(a3_2, wa3_2),
        new Edge(b3_2, wb3_2),
        new Edge(c3_2, wc3_2),};


int L7_3 = 295;
double wa7_3 = (1 + (double) k.nextInt(10))  * L7_3;
double wa7_6 = (1 + (double) k.nextInt(10)) * L6_7;
double wb7_6 = (1 + (double) k.nextInt(10)) * L6_7;
int L7_10 = 290;
double wa7_10 = (1 + (double) k.nextInt(10))* L7_10;
double wb7_10 = (1 + (double) k.nextInt(10))  * L7_10;
double wb7_3 = (1 + (double) k.nextInt(10))  * L7_3;
a3_7.adjacencies = new Edge[]{
        new Edge(a7_3, wa7_3),
        new Edge(b7_3, wb7_3),
        new Edge(a7_10, wa7_10),
        new Edge(b7_10, wb7_10),  };

b3_7.adjacencies = new Edge[]{
        new Edge(a7_6, wa7_6),
        new Edge(b7_6, wb7_6),
        new Edge(a7_10, wa7_10),
        new Edge(b7_10, wb7_10),   };

a7_3.adjacencies = new Edge[]{
        new Edge(a3_7, wa3_7),
        new Edge(b3_7, wb3_7),
        new Edge(a3_2, wa3_2),
        new Edge(b3_2, wb3_2),
        new Edge(c3_2, wc3_2),  };

b7_3.adjacencies = new Edge[]{
        new Edge(a3_4, wa3_4),
        new Edge(b3_4, wb3_4),  };

int L11_4 = 650;
double wa11_4 = (1 + (double) k.nextInt(10)) * L11_4;
double wb11_4 = (1 + (double) k.nextInt(10)) * L11_4;
int L11_10 = 250;
double wa11_10 = (1 + (double) k.nextInt(10))  * L11_10;
a4_11.adjacencies = new Edge[]{
        new Edge(a11_4, wa11_4),
        new Edge(b11_4, wb11_4), };

int L10_11 = 250;
double wb11_10 = (1 + (double) k.nextInt(10))  * L10_11;
b4_11.adjacencies = new Edge[]{
        new Edge(a11_10, wa11_10),
        new Edge(b11_10, wb11_10),  };
```

```
a11_4.adjacencies = new Edge[]{
        new Edge(a4_3, wa4_3),
        new Edge(b4_3, wb4_3),
        new Edge(a4_11, wa4_11),
        new Edge(b4_11, wb4_11), };

b11_4.adjacencies = new Edge[]{
        new Edge(a11_4, wa11_4),  };

a5_6.adjacencies = new Edge[]{
        new Edge(a6_5, wa6_5),
        new Edge(b6_5, wb6_5),
        new Edge(a6_2, wa6_2),
        new Edge(b6_2, wb6_2),
        new Edge(c6_2, wc6_2),
        new Edge(a6_7, wa6_7),
        new Edge(b6_7, wb6_7),
        new Edge(c6_7, wc6_7),  };

b5_6.adjacencies = new Edge[]{
        new Edge(a6_9, wa6_9),
        new Edge(b6_9, wb6_9),
        new Edge(a6_7, wa6_7),
        new Edge(b6_7, wb6_7),
        new Edge(c6_7, wc6_7),  };

a6_5.adjacencies = new Edge[]{
        new Edge(a5_6, wa5_6),
        new Edge(b5_6, wb5_6),
        new Edge(a5_8, wa5_8),
        new Edge(b5_8, wb5_8), };

b6_5.adjacencies = new Edge[]{
        new Edge(a5_1, wa5_1),
        new Edge(b5_1, wb5_1), };

a6_7.adjacencies = new Edge[]{
        new Edge(a7_6, wa7_6),
        new Edge(b7_6, wb7_6),
        new Edge(a7_3, wa7_3),
        new Edge(b7_3, wb7_3),  };

b6_7.adjacencies = new Edge[]{
        new Edge(a6_7, wa6_7),
        new Edge(c6_7, wc6_7), };

c6_7.adjacencies = new Edge[]{
        new Edge(a7_10, wa7_10),
        new Edge(b7_10, wb7_10), };

a7_6.adjacencies = new Edge[]{
        new Edge(a6_7, wa6_7),
        new Edge(b6_7, wb6_7),
        new Edge(c6_7, wc6_7),
        new Edge(a6_5, wa6_5),
        new Edge(b6_5, wb6_5),
        new Edge(a6_9, wa6_9),
        new Edge(b6_9, wb6_9),  };

b7_6.adjacencies = new Edge[]{
        new Edge(a6_2, wa6_2),
        new Edge(b6_2, wb6_2),
        new Edge(c6_2, wc6_2),
```

```
        new Edge(a6_5, wa6_5),
        new Edge(b6_5, wb6_5),  };


int L8_5 = 295;
double wa8_5 = (1 + (double) k.nextInt(10)) * L8_5;
double wb8_5 = (1 + (double) k.nextInt(10))  * L8_5;
double wc8_5 = 1 + (double) k.nextInt(10)  * L8_5;
int L8_9 = 325;
double wa8_9 = (1 + (double) k.nextInt(10))  * L8_9;
double wb8_9 = (1 + (double) k.nextInt(10)) * L8_9;
a5_8.adjacencies = new Edge[]{
        new Edge(a8_5, wa8_5),
        new Edge(b8_5, wb8_5),
        new Edge(c8_5, wc8_5),
        new Edge(a8_9, wa8_9),
        new Edge(b8_9, wb8_9), };

b5_8.adjacencies = new Edge[]{
        new Edge(a5_8, wa5_8),};

a8_5.adjacencies = new Edge[]{
        new Edge(a5_8, wa5_8),
        new Edge(b5_8, wb5_8),};

b8_5.adjacencies = new Edge[]{
        new Edge(a5_1, wa5_1),
        new Edge(b5_1, wb5_1), };

c8_5.adjacencies = new Edge[]{
        new Edge(a5_6, wa5_6),
        new Edge(b5_6, wb5_6), };


int L9_8 = 325;
double wa9_8 = (1 + (double) k.nextInt(10))  * L9_8;
double wb9_8 = (1 + (double) k.nextInt(10))  * L9_8;
double wa9_6 = (1 + (double) k.nextInt(10))  * L6_9;
int L9_6 = 290;
double wb9_6 = (1 + (double) k.nextInt(10))  * L9_6;
int L9_10 = 300;
double wa9_10 = (1 + (double) k.nextInt(10))  * L9_10;
double wb9_10 = (1 + (double) k.nextInt(10))  * L9_10;
a8_9.adjacencies = new Edge[]{
        new Edge(a9_8, wa9_8),
        new Edge(b9_8, wb9_8),
        new Edge(a9_6, wa9_6),
        new Edge(b9_6, wb9_6),
        new Edge(a9_10, wa9_10),
        new Edge(b9_10, wb9_10),  };

b8_9.adjacencies = new Edge[]{
        new Edge(a9_10, wa9_10),
        new Edge(b9_10, wb9_10),  };

a9_8.adjacencies = new Edge[]{
        new Edge(a8_9, wa8_9),
        new Edge(b8_9, wb8_9), };

b9_8.adjacencies = new Edge[]{
        new Edge(a8_5, wa8_5),
        new Edge(b8_5, wb8_5),
        new Edge(c8_5, wc8_5),  };
```

```
a6_9.adjacencies = new Edge[]{
        new Edge(a9_6, wa9_6),
        new Edge(b9_6, wb9_6),
        new Edge(a9_10, wa9_10),
        new Edge(b9_10, wb9_10),    };

b6_9.adjacencies = new Edge[]{
        new Edge(a9_8, wa9_8),
        new Edge(b9_8, wb9_8), };

a9_6.adjacencies = new Edge[]{
        new Edge(a6_9, wa6_9),
        new Edge(b6_9, wb6_9),
        new Edge(a6_2, wa6_2),
        new Edge(b6_2, wb6_2),
        new Edge(c6_2, wc6_2),
        new Edge(a6_5, wa6_5),
        new Edge(b6_5, wb6_5), };

b9_6.adjacencies = new Edge[]{
        new Edge(a6_2, wa6_2),
        new Edge(b6_2, wb6_2),
        new Edge(c6_2, wc6_2),
        new Edge(a6_7, wa6_7),
        new Edge(b6_7, wb6_7),
        new Edge(c6_7, wc6_7),   };


int L10_9 = 300;
double wa10_9 = (1 + (double) k.nextInt(10)) * L10_9;
double wb10_9 = (1 + (double) k.nextInt(10))  * L10_9;
int L10_7 = 290;
double wa10_7 = (1 + (double) k.nextInt(10))  * L10_7;
double wb10_7 = (1 + (double) k.nextInt(10))  * L10_7;
double wa10_11 = (1 + (double) k.nextInt(10))  * L10_11;
double wb10_11 = (1 + (double) k.nextInt(10)) * L10_11;

a9_10.adjacencies = new Edge[]{
        new Edge(a10_9, wa10_9),
        new Edge(b10_9, wb10_9),
        new Edge(a10_7, wa10_7),
        new Edge(b10_7, wb10_7),
        new Edge(a10_11, wa10_11),
        new Edge(b10_11, wb10_11), };

b9_10.adjacencies = new Edge[]{
        new Edge(a10_11, wa10_11),
        new Edge(b10_11, wb10_11), };

a10_9.adjacencies = new Edge[]{
        new Edge(a9_10, wa9_10),
        new Edge(b9_10, wb9_10),
        new Edge(a9_8, wa9_8),
        new Edge(b9_8, wb9_8), };

b10_9.adjacencies = new Edge[]{
        new Edge(a9_6, wa9_6),
        new Edge(b9_6, wb9_6),
        new Edge(a9_8, wa9_8),
        new Edge(b9_8, wb9_8), };

a7_10.adjacencies = new Edge[]{
```

```
                new Edge(a10_7, wa10_7),
                new Edge(b10_7, wb10_7),
                new Edge(a10_11, wa10_11),
                new Edge(b10_11, wb10_11),};

        b7_10.adjacencies = new Edge[]{
                new Edge(a10_9, wa10_9),
                new Edge(b10_9, wb10_9), };

        a10_7.adjacencies = new Edge[]{
                new Edge(a7_10, wa7_10),
                new Edge(b7_10, wb7_10),
                new Edge(a7_6, wa7_6),
                new Edge(b7_6, wb7_6),
                new Edge(a7_3, wa7_3), };

        b10_7.adjacencies = new Edge[]{
                new Edge(a7_3, wa7_3), };

        a10_11.adjacencies = new Edge[]{
                new Edge(a11_10, wa11_10),
                new Edge(b11_10, wb11_10),
                new Edge(a11_4, wa11_4),
                new Edge(b11_4, wb11_4), };

        b10_11.adjacencies = new Edge[]{
                new Edge(a10_11, wa10_11), };

        a11_10.adjacencies = new Edge[]{
                new Edge(a10_11, wa10_11),
                new Edge(b10_11, wb10_11),
                new Edge(a10_9, wa10_9),
                new Edge(b10_9, wb10_9),
        };
        b11_10.adjacencies = new Edge[]{
                new Edge(a10_7, wa10_7),
                new Edge(b10_7, wb10_7),
                new Edge(a10_9, wa10_9),
                new Edge(b10_9, wb10_9),
        };

    }

    public static void main(String[] args) {
        //initialize the graph map
        initGraph();


        AstarSearch(b1_2, b10_11);
        List<Node> path = printPath(b10_11);
        System.out.println("Path: " + path);
        while (path.size() > 1) {
            Node node = path.get(1);
            node.parent = null;
            for (Node n : nodes) {
                n.parent = null;
            }

            AstarSearch(path.get(1), b10_11);
            path = printPath(b10_11);
            System.out.println("Path: " + path);
        }
```

```
    }

    public static List<Node> printPath(Node target) {
        ArrayList<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>(100, 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;
    }
    public String toString() {
        return "Edge(" + target + ", " + cost + ")";
    }
}
```

У лістінгу 1 умові (2) відповідають інструкції виду

```
double wa2_1 = (1 + (double) k.nextInt(10)) * L2_1 ,
```
(8)

де `wa2_1` являє собою вагу ребра `a2_1`; співмножник `(1 + (double) k.nextInt(10))` представляє собою рендомне число, що змінюється в діапазоні $1 \div 10$ та імітує собою число циклів переключення світлофора (з іншої сторони, цей співмножник дорівнює величині відношення (3)); **`k`** – змінна генератора випадкових чисел **`Random k = new Random();`**

Запустивши програму, можемо отримати результат, приведений знизу (інший запуск програми дає, як правило, інші дані, що обумовлено дією генератора рендомних чисел):

Path1: [b1_2-Liberty avenue, **b2_3-Broad street**, b3_7-Valley street, a7_6-Happy street, a6_9-Joy street, b9_10-Kennedy street, b10_11-Apple street]

Path2: [b2_3-Broad street, **b3_7-Valley street**, a7_6-Happy street, a6_9-Joy street, b9_10-Kennedy street, b10_11-Apple street]

Path3: [b3_7-Valley street, **a7_6-Happy street**, a6_9-Joy street, b9_10-Kennedy street, b10_11-Apple street]
Path4: [a7_6-Happy street, **a6_9-Joy street**, b9_10-Kennedy street, b10_11-Apple street]
Path5: [a6_9-Joy street, **b9_10-Kennedy street**, b10_11-Apple street]
Path6: [b9_10-Kennedy street, **b10_11-Apple street**]
Path7: [b10_11-Apple street]

Ми свідомо замінили у лістингу 1 деякі назви ребер (вулиць) – щоб прокладений маршрут виглядав більш наочно і тому отримали результат роботи програми у вигляді запису, представленого вище. Приведений результат має ряд специфічних особливостей. Зокрема, кожен попередній *Path* пов'язаний з наступним і здійснюється з використанням результату розрахунку на попередньому кроці – програма перехоплює другу позицію попереднього розрахованого маршруту (ці ключові смуги руху виділені) і починаючи із цієї позиції розраховує новий маршрут. Мається на увазі, що цінність розрахованого на першому кроці маршруту в силу високої динамічності міського трафіку буде зберігатись лише на протязі проїзду водієм тільки однієї смуги. Як тільки ТЗ перетне перехрестя, програма розраховує новий маршрут, стартуючи кожен раз із позиції, що стоїть на другому місці попередньо розрахованого *Path*. Наприклад, *Path*2 в якості стартової позиції для розрахунку нового маршруту використовує смугу b1_2-Liberty avenue , яка стоїть на другій позиції у *Path*1. Далі програма в якості стартової позиції для розрахунку маршруту *Path*3 використовує як стартову позицію смугу b3_7-Valley street. Аналогічно обраховуються наступні маршрути аж поки не буде досягнена кінцева мета (у нашому випадку це смуга b10_11-Apple street). Іншими словами, реально використовуються з кожного *Path* лише перші дві позиції; інші – ігноруються. Все це робиться для того щоб працювати у режимі постійного оновлення маршруту.

Як підкреслювалось вище, передача оновленого маршруту ініціюється вхідними сенсорами – як тільки автомобіль перетне вхідний сенсор (про це буде сигналізувати GPS-трекер, установлений на автомобілі) програма на ЦКТ

розрахує та передасть на GPS-навігатор новий $path$. Таким чином, завдяки процесу постійного обновлення маршруту, програмне забезпечення трафіку володіє високим ступенем динамічності і в той самий час є дуже комфортним для кожного водія, адже водій має змогу перелаштуватись на потрібну смугу руху, тому що кожна нова інструкція поступатиме йому зразу після перетину перехрестя, а точніше – вхідного сенсора на цьому перехресті. Таким чином, алгоритм працює у повній відповідності із динамікою змін у транспортній мережі міста, постійно прокладаючи нові маршрути з періодичністю, що визначається часом проїзду ТЗ між сусідніми перехрестями: як тільки автомобіль перетинає перехрестя, то водій зразу-ж отримує оновлений маршрут з урахуванням змін, що відбулися на цей момент часу. Таким чином, система працює синхронно із змінами трафіку, що обумовлює її високу ефективність (на відміну від сучасної GPS-навігації, яка працює із запізненням – затори у місті вже утворились, а навігатор «намагається розрулити» ситуацію). При нашому підході затори у місті будуть зведені до мінімума.

Завдання програми, приведеної вище, прокладати оптимальні маршрути всім ТЗ, що замовили трафік. Припустимо, що водій i знаходиться у початковій позиції S_i, а позиція F_i є кінцевою для його маршруту. Тоді для кожного заявленого маршруту i програма з урахуванням величин евристичних відстаней $h(n)$, формує у відповідності із умовою (2) нерозривний ланцюг, що пов'язує початкове та кінцеве положення кожного конкретного ТЗ. Фактично програма на ЦКТ працює із спектром бінарних відношень наступного виду

$$R = \{(S_1, F_1), (S_2, F_2), (S_3, F_3), \ldots\ldots (S_i, F_i), \ldots (S_m, F_m)\}. \qquad (7)$$

Пари такого типу є ключовими для ЦКТ, адже потрібно прокласти t – оптимальний маршрут кожному автомобілю i, а число m таких об'єктів у мегаполісі може перевищувати мільйон.

З метою адаптації програми лістінг 1 до реальних умов проводилось тестування представленого алгоритму на графах з числом вершин до 165 та числом ребер до 445. Але оскільки складність A*-алгоритму лінійна $O(n)$, то це дозволило без проблем прокладати оптимальні маршрути і в такому середовищі. Значить, така програма є ефективною для задач з великим об'ємом вхідних даних. Це, в свою чергу означає, що можна використати представлену технологію для прокладання маршрутів ТЗ у містах-мегаполісах. Іншими словами, ЦКТ одночасно може супроводжувати мільйон або більше автомобілів. Було використано також і інший підхід з метою апробації описаного алгоритму – проводилось моделювання транспортної мережі з допомогою високоефективної програми імітаційного моделювання AnyLogic 8.5.1[16]. Результати моделювання показали суттєве зростання пропускної здатності міських транспортних мереж у порівнянні із стандартним режимом, коли використовується традиційне світлофорне регулювання та сучасна GPS-навігація. Сказане свідчить про цілковиту прийнятність пропонованого проекту для цілей технічного впровадження.

4.Висновки

Отже, запропонована АІС допоможе позбутись проблеми кожного великого міста – заторів. Для водіїв не буде проблеми вибору маршруту, а сам маршрут стане коротшим по часу та комфортним. Технологія, розроблена на основі представленого алгоритму, – при втіленні її у життя – значно покращить трафік-клімат на вулицях міст та зіграє суттєву роль в організації руху міського транспорту в усьому світі. Все сказане означає, що представлена АІС ставить за мету реальне вирішення проблеми трафіку у великих містах.

5.Список використаних літературних джерел

1. Thiago Sobral, Teresa Galvao and Jose Bornes. Visualization of Urban Mobility Data from Intelligent Transportation Systems/ Sobral Thiago, Galvao Teresa and Bornes Jose// Sensors. – 2019. – V.19. – Is.2. – P.332 – 360.

2. Ahmed Elbery and Hesham Rakha. City-Wide Eco-Routing Navigation Considering Vehicular Communication Impacts/ Elbery Ahmed and Rakha Hesham// Sensors. – 2019. – V.19. – Is.2. – P.290 – 311.

3. Shiva Rahimipour, Rayehe Moeinfar, S. Mehdi Hashemi. Traffic prediction using a self-adjusted evolutionary neural network/ Rahimipour Shiva , Moeinfar Rayehe , Hashemi S. Mehdi // J. Mod. Transport. – 2019. – V. 27. – Is.4. – P.306–316.

4. Azadeh Emami, Majid Sarvi, Saeed Asadi Bagloee. Using Kalman filter algorithm for short-term traffic flow prediction in a connected vehicle environment / Emami Azadeh, Sarvi Majid , Bagloee Saeed Asadi // J. Mod. Transport. – 2019. – V. 27. – Is.3. – P.222–232.

5. Xianglong Luo, Xue Meng, Wenjuan Gan, and Yonghong Chen. Traffic Data Imputation Algorithm Based on Improved Low-Rank Matrix Decomposition/Xianglong , Meng Xue , Gan Wenjuan , and Chen Yonghong Luo// Journal of Sensors. –V. 2019. – Article ID 7092713, 11 pages.

6. Andrea Pompigna, Federico Rupia. Comparing practice-ready forecast models for weekly and monthly fluctuations of average daily traffic and enhancing accuracy by weighting methods/ Pompigna Andrea, Rupia Federico // Journal of Traffic and Transportation Engineering (English Edition). – 2018. – V.5. – Is. 4. – P. 239-253.

7. Hirsh Majid, Chao Lu, Hardy Karim. An integrated approach for dynamic traffic routing and ramp metering using sliding mode control/ Majid Hirsh , Lu Chao, Karim Hardy // Journal of Traffic and Transportation Engineering (English Edition). – 2018. – V.5. – Is. 2. – P. 116-128.

8. Pat. CN104064049B China, A kind of intelligent transportation road capacity note broadcasting system / Xu Chunmao Xu Jin; publ.08.03.2017.

9. Pat. CN110136454 (A) China, Urban artery traffic dynamic green wave signal control system and method based on real-time traffic flow data/ Shu Aibing, Xu Xindong, Xu Leng, Zhang Bin, Liu Chengsheng, Cai Yubao; applicant Traffic Management Res Institute of the Ministry of Public Security; publ.16.08.2019 .

10. Pat. CN109920250 (A) China, Dynamic prediction intelligent traffic management method for urban road / Zhang Hongtao, Li Tianxue, Hu Haitao, Li Lei, Hao Yan, Chen Qingshan, Zhang Weiling, Chu Peng, Jiang Chunhui, Zhou Wei; applicant Li Tianxue; publ. 21.06.2019.

11. Tong Wang , Azhar Hussain, Yue Cao, Sangirov Gulomjon . An Improved Channel Estimation Technique for IEEE 802.11p Standard in Vehicular Communications/ Wang Tong , Hussain Azhar, Cao Yue, Gulomjon Sangirov / Sensors. – 2019. – V.19. – Is.1. – 22p.

12. Emmanuel Kidando, Ren Moses, Thobias Sando, Eren Erman Ozguven. An application of Bayesian multilevel model to evaluate variations in stochastic and dynamic transition of traffic conditions/ Kidando Emmanuel , Moses Ren , Sando Thobias , Ozguven Eren Erman// J. Mod. Transport. – 2019. – V. 27. – Is.4. – P.235–249.

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

14. Boguto D.G., Kadomskiy K.K. Nikolyuk P.K., Pidgurska A.I. Algorithm of Intelligent Urban Traffic/ D.G. Boguto, K.K. Kadomskiy, P.K. Nikolyuk, A.I. Pidgurska// Bulletin of V.Karazin Kharkiv National University, Series “Mathematical Modeling. Information Technology. Automated Control System”.–2019. – Is.42.– P.12-25.

15. 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.

16.Альфа-версия Any-Logic 8.5.1 [Електронний ресурс]. Режим доступу: https://www.anylogic.com/downloads.