УДК 004.94
О.К.Жигаревич
Луцький інститут розвитку людини Університету «Україна»

МЕТОДИ ТА ЗАСОБИ ПРОЕКТУВАННЯ ТА РОЗРОБКИ СИСТЕМИ ОПТИМІЗАЦІЇ ТРАНСПОРТНИХ МАРШРУТІВ

У статті розглядається поняття штучного інтелекту та алгоритмів пошуку оптимальних маршрутів, розкрито поняття оптимізації маршруту та пошуку шляху. Досліджується поняття графу та алгоритми пошуку, що працюють на графі.

Ключові слова*: штучний інтелект, оптимізація, пошук шляху, граф, цикл, MongoDB, PHP.*
Форм. 1. Рис. 3. Літ 7.

Постановка проблеми.

Штучний інтелект (ШІ, англ. Artificialintelligence, AI) - наука і технологія створення інтелектуальних машин, особливо інтелектуальних комп'ютерних програм. ШІ пов'язаний з подібною метою використання комп'ютерів для розуміння людського інтелекту, але не обов'язково обмежується біологічно правдоподібними методами. Завданням цієї науки є відтворення за допомогою обчислювальних систем та інших штучних пристроїв розумних міркувань і дій.

Пошук шляху (англ. Pathfinding) — термін в інформатиці і штучному інтелекті, який означає визначення комп'ютерною програмою маршруту між двома точками.

Оптимізацією, що є ключовою темою в галузях інженерії та науки, називають процес знаходження найкращого у найбільш ефективний спосіб рішення поставленої задачі, згодом з певними обмеженнями.

Оптимізація шляху – це визначення найкращого та найбільш оптимального шляху. Більшість алгоритмів визначення оптимального шляху працюють з графами. Графи дуже добре підходять для відображення транспортних зупинок та зв'язків між ними. Вузли графа – це зупинки, а ребра – зв'язки між зупинками.

Види графів для представлення маршрутної мережі

У математичній теорії графів та інформатики граф – це сукупність непорожньої безлічі вершин і безлічі пар вершин.

Об'єкти представляються як вершини, або вузли графа, а зв'язки – як дуги, або ребра. Для різних областей застосування види графів можуть відрізнятися спрямованістю, обмеженнями на кількість зв'язків та додатковими даними про вершини або ребрах.

Багато структур, що представляють практичний інтерес у математиці й інформатиці, можуть бути представлені графами.

Теорія графів не володіє усталеною термінологією. У різних статтях під одними і тими ж термінами розуміються різні речі. Наведені нижче визначення – найбільш часто зустрічаються.

Граф або неорієнтований граф G - це впорядкована пара G: = (V, E), для якої виконані наступні умови:

- V це непорожня множина вершин або вузлів;
- E це безліч пар (у разі неорієнтованого графа - невпорядкованих) вершин, званих ребрами.

V (а значить і E, інакше воно було б мультимножиною) зазвичай вважаються кінцевими множинами. Багато хороших результатів, отриманих для кінцевих графів, невірні (або яким-небудь чином відрізняються) для нескінченних графів. Це відбувається тому, що ряд міркувань стає помилковим у разі нескінченних множин.

Вершини і ребра графа називаються також елементами графа, число вершин у графі | V | - порядком, число ребер | E | - розміром графа.

Вершини u і v називаються кінцевими вершинами (або просто кінцями) ребра $e = \{u, v\}$. Ребро, в свою чергу, з'єднує ці вершини. Дві кінцеві вершини одного і того ж ребра називаються сусідніми. [1]

Два ребра називаються суміжними, якщо вони мають загальну кінцеву вершину.

Два ребра називаються кратними, якщо множини їхніх кінцевих вершин збігаються.

Ребро називається петлею, якщо його кінці збігаються, тобто $e = \{v, v\}$.

Ступенем deg V вершини V називають кількість ребер, для яких вона є кінцевою (при цьому петлі рахують двічі).

Вершина називається ізольованою, якщо вона не є кінцевою для жодного ребра; висячою (або листом), якщо вона є кінцем рівно одного ребра.

Орієнтований граф G - це впорядкована пара G: = (V, A), для якої виконані наступні умови:

- V це непорожня множина вершин або вузлів;
- A це множина (впорядкованих) пар різних вершин, званих дугами або орієнтованими ребрами.

Дуга - це впорядкована пара вершин (v, w), де вершину v називають початком, а w - кінцем дуги. Можна сказати, що дуга $v \rightarrow w$ веде від вершини v до вершини w.

Змішаний граф G - це граф, в якому деякі ребра можуть бути орієнтованими, а деякі - неорієнтованим. Записується впорядкованої трійкою G: = (V, E, A), де V, E і A визначені так само, як вище.[1]

Орієнтований і неорієнтований графи є окремими випадками змішаного.

Шляхом (або ланцюгом) у графі називають кінцеву послідовність вершин, в якій кожна вершина (крім останньої) з'єднана з наступною в послідовності вершин.

Циклом називають шлях, в якому перша і остання вершини збігаються. При цьому довжиною шляху (або циклу) називають число складових його ребер. Зауважимо, що якщо вершини u і v є кінцями деякого ребра, то згідно з цим визначенням, послідовність (u, v, u) є циклом. Щоб уникнути таких «вироджених» випадків, вводять наступні поняття.

Шлях (або цикл) називають **простим**, якщо ребра в ньому не повторюються; **елементарним**, якщо він простий і вершини в ньому не повторюються. З цього видно, що:

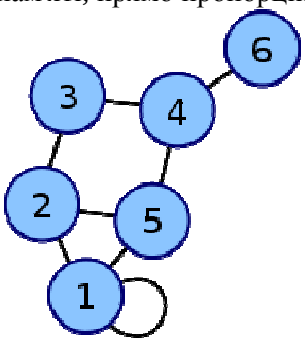
- Усілякий шлях, що сполучає дві вершини, містить елементарний шлях, що з'єднує ті ж дві вершини;
- Всякий простий неелементарний шлях містить елементарний цикл;
- Всякий простий цикл, що проходить через деяку вершину (або ребро), містить елементарний (під-) цикл, що проходить через ту ж вершину (або ребро);
- Петля - елементарний цикл.

Мета роботи є дослідження методів та засобів проектування системи штучного інтелекту оптимізації транспортних маршрутів, алгоритмів розрахунку оптимального маршруту, а також проектування зручного інтерфейсу клієнтської частини проекту.

Результати проведених досліджень. Дослідження,опитування, спостереження, аналіз, синтез, інтроспекція, метод аналогій, узагальнення, вивчення спеціалізованої літератури, перегляд доповідей конференцій, аналіз електронних та друкованих видань з цієї теми.

Способи представлення графів в інформатиці. За допомогою матриці суміжності. Таблиця, де як стовпці, так і рядки відповідають вершинам графа. У кожній клітинці цієї матриці записується число, що визначає наявність зв'язку від вершини-рядка до вершині-стовпцю (або навпаки) (рис. 1).

Недоліком є вимоги до пам'яті, прямо пропорційні квадрату кількості вершин.



$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Рис. 1. Представлення графу за допомогою матриці суміжності
Авторська розробка

За допомогою **матриці інцидентності**. Кожен рядок відповідає певній вершині графа, а стовпці відповідають ребрам графа (рис.2). У комірку на перетині і-го рядка з j-м стовпцем матриці записується:

- 1 у випадку, якщо зв'язок j «виходить» з вершини і;
- -1 якщо зв'язок «входить» у вершину;
- 0 у всіх інших випадках (тобто якщо зв'язок є петлею або зв'язок не інцидентний вершині).

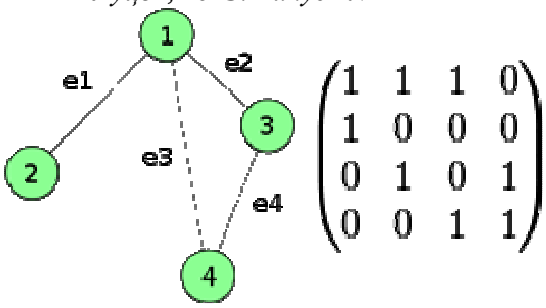


Рис. 2. Представлення графу за допомогою матриці інцидентності
Авторська розробка

Даний спосіб є самим ємним (розмір пропорційний | E | | V |) для зберігання, але полегшує знаходження циклів в графі.

Список ребер -- це тип представлення графа в пам'яті комп'ютерної програми, що передбачає, що кожне ребро представляється двома числами - номерами вершин цього ребра. Список ребер більш зручний для реалізації різних алгоритмів на графах в порівнянні з матрицею суміжності.

Основні алгоритми пошуку оптимального шляху

Пошук – це універсальний механізм для вирішення задач у штучному інтелекті (ШІ). У задачах ШІ, послідовність кроків для розв'язку задачі невідома апріорі, але часто може бути визначена за допомогою систематичного дослідження альтернатив методом проб і помилок.

Алгоритм Дейкстри (Dijkstra'salgorithm) - алгоритм на графах, винайдений нідерландським вченим Е. Дейкстрою в 1959 році. Знаходить найкоротшу відстань від однієї з вершин графа до всіх інших. Алгоритм працює тільки для графів без ребер від'ємної ваги. Алгоритм широко застосовується в програмуванні й технологіях.

Формальне визначення. Дано зважений граф G (V, E) без петель і дуг від'ємної ваги. Знайти найкоротші шляхи від деякої вершини *a* графа G до всіх інших вершин цього графа.

Кожній вершині з V зіставимо мітку - мінімальна відома відстань від цієї вершини до *a*. Алгоритм працює покроково - на кожному кроці він «відвідує» одну вершину і намагається зменшувати мітки. Робота алгоритму завершується, коли всі вершини відвідані.

Ініціалізація. Мітка самої вершини *a* покладається рівною 0, мітки інших вершин - нескінченності. Це відображає те, що відстані від *a* до інших вершин поки невідомі. Всі вершини графа позначаються як невідвіданні.

Крок алгоритму. Якщо всі вершини відвідані, алгоритм завершується. В іншому випадку, з ще не відвіданих вершин вибирається вершина *u*, що має мінімальну мітку. Ми розглядаємо всілякі маршрути, в яких *u* є передостаннім пунктом. Вершини, в які ведуть ребра з *u*, назвемо сусідами цієї вершини. Для кожного сусіда вершини *u*, крім позначених як відвідані, розглянемо нову довжину шляху, що дорівнює сумі значень поточної мітки *u* і довжини ребра, що з'єднує *u* з цим сусідом. Якщо отримане значення довжини менше значення мітки сусіда, замінимо значення мітки отриманим значенням довжини. Розглянувши всіх сусідів, помітимо вершину *u* як відвідану і повторимо крок алгоритму[1].

Пошук у ширину(BFS, Breadth-first search) — алгоритм пошуку на графі.Якщо задано граф G = (V, E) та початкову вершину *s*, алгоритм пошуку в ширину систематично обходить всі досяжні із *s* вершини. На першому кроці вершина *s* позначається, як пройдена, а в список додаються всі вершини, досяжні з *s* без відвідування проміжних вершин. На кожному наступному кроці всі поточні вершини списку відмічаються, як пройдені, а новий список формується із вершин, котрі є ще не пройденими сусідами поточних вершин списку. Для реалізації списку вершин найчастіше використовується черга. Виконання алгоритму продовжується до досягнення шуканої вершини або до того часу, коли на певному кроці в список не включається жодна вершина (рис. 2.3). Другий випадок означає, що всі вершини, доступні з початкової, уже відмічені, як пройдені, а шлях до цільової вершини не знайдений.

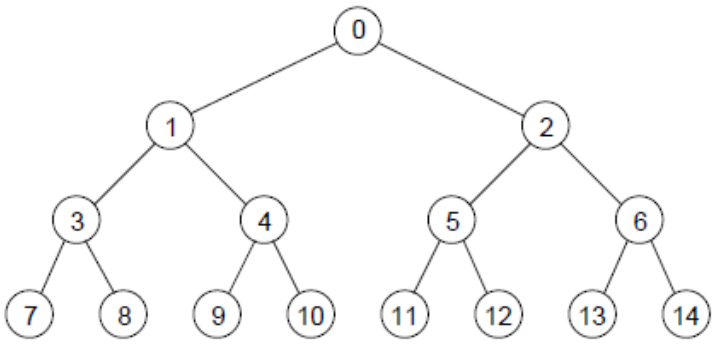


Рис. 3.Порядок обходу вершин алгоритму пошуку в ширину
Авторська розробка

Алгоритм має назву пошуку в ширину, оскільки «фронт» пошуку (між пройденими та непройденими вершинами) одноманітно розширюється вздовж всієї своєї ширини. Тобто, алгоритм проходить всі вершини на відстані k перед тим як пройти вершини на відстані k+1.

Головним недоліком пошуку у ширину є вимоги до пам'яті комп'ютера. Кожен рівень дерева має бути збережений для генерації наступного рівня, а обсяг пам'яті пропорційний кількості вузлів, що зберігаються. Як результат, пошук у ширину виснажить доступну оперативну пам'ять за лічені хвилини [2].

Пошук «Кращий - перший»(Best-firstsearch) - це алгоритм пошуку, який досліджує граф шляхом розширення найбільш перспективних вузлів, обраних відповідно до зазначеного правила.

Цей алгоритм використовує евристичну функцію, яка намагається передбачити наскільки близько кінець шляху до розв'язку, щоб шляхи, які вважаються ближчими до рішення, перевірялися першими.

Ефективний вибір поточного кращого кандидата для розширення пошуку, як правило, реалізується за допомогою черги з пріоритетом.

Алгоритм пошуку A* («А зірочка» або англ. «A star») — належить до евристичних алгоритмів пошуку. Використовується для пошуку найкоротшого шляху між двома вершинами графу з додатніми вагами ребер. Описаний 1968 р. Пітером Хартом, Нільсом Нільсоном та БертрамомРафаелєм.

Алгоритм використовує допоміжну функцію (евристику), аби скеровувати напрям пошуку та скорочувати його тривалість. Алгоритм повний в тому сенсі, що завжди знаходить оптимальний розв'язок, якщо він існує.

Алгоритм A* спершу відвідує ті вершини, які ймовірно ведуть до найкоротшого шляху до мети. Аби розпізнати такі вершини, кожній відомій вершині x співставляється значення f(x), яке дорівнює довжині найкоротшого шляху від початкової вершини до кінцевої, який пролягає через обрану вершину. Вершини з найменшим значенням f обираються в першу чергу.

Функція f(x) для вершини x визначається так:

$$f(x) = g(x) + h(x)$$

де:

- g(x) функція, значення якої дорівнюють вартості шляху від початкової вершини до x;
- h(x) евристична функція, оцінює вартість шляху від вершини x до кінцевої.

Використана евристика не повинна давати завищену оцінку вартості шляху. Прикладом оцінки може служити пряма лінія: загальний шлях не може бути коротшим за пряму лінію.

Алгоритм ділить вершини на три класи:

1. невідомі вершини: ці вершини ще не були знайдені, ще не відомий шлях до них, на початку роботи алгоритму всі вершини, окрім початкової, належать до класу невідомих;
2. відомі вершини (OpenList): вже відомий (можливо не оптимальний) шлях до цих вершин, всі відомі вершини разом зі значеннями f зберігаються в списку, з цього списку вибираються, в першу чергу, найперспективніші вершини, конкретна реалізація цього списку має істотний вплив на швидкодію алгоритму, і зазвичай має вигляд черги з

пріоритетом (наприклад, бінарна купа), на початку роботи алгоритму до відомих вершин належить лише початкова вершина;

3. повністю досліджені вершини (ClosedList): до цих вершин вже відомий найкоротший шлях, повністю досліджені вершини додаються до так званого замкненого списку, аби запобігти багаторазовому дослідженню вже досліджених вершин, список повністю досліджених вершин на початку роботи алгоритму порожній.

Кожна відома або повністю досліджена вершина має вказівник на попередні вершини. Завдяки цьому вказівникові, можна пройти шляхом від цієї до початкової вершини.

Коли вершину *x* буде повністю досліджено (або розкрито, релаксовано), суміжні з нею вершини додаються до списку відомих вершин, а сама вершина додається в список повністю досліджених. Вказівники на попередню вершину встановлюються на *x*. Суміжні вершини, які вже знаходяться в списку повністю досліджених вершин, до списку відомих не додаються, а зворотні вказівники не змінюються. Суміжні вершини, які вже знаходяться в списку відомих, лише оновлюються (значення f та вказівник на попередню вершину), якщо знайдений до них шлях коротший за вже відомий [3].

Алгоритм зупиняється коли кінцева вершина потрапляє до списку повністю досліджених вершин. Знайдений шлях відтворюється із допомогою вказівників на попередню вершину. Якщо список відомих вершин порожніє, то розв'язку задачі не існує і алгоритм припиняє пошук.

Відтворений за зворотними вказівниками знайдений шлях починається з кінцевої вершини та прямує до початкової. Аби одразу отримати шлях в правильному напрямі, з початкової вершини до кінцевої, в умовах задачі слід переставити місцями початок та кінець. Якщо шукати шлях починаючи з кінцевої вершини, відтворений список починатиметься з початкової вершини й прямуватиме до кінцевої.

Вибір засобів та технологій для реалізації проекту

Для зберігання даних про транспорт та зупинки, а також додаткову інформацію, було вирішено обрати документно-орієнтовану СКБД MongoDB.

MongoDB — документо-орієнтована система керування базами даних (СКБД) з відкритим кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів [4].

Код MongoDB написаний на мові C++ і поширюється в рамках ліцензії AGPLv3.

MongoDB підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміни і додавання даних в БД, може працювати відповідно до парадигми Map/Reduce, підтримує реплікацію і побудову відмовостійких конфігурацій. У MongoDB є вбудовані засоби із забезпечення шардінга (розподіл набору даних по серверах на основі певного ключа), комбінуючи який реплікацією даних можна побудувати горизонтально масштабований кластер зберігання, в якому відсутня єдина точка відмови (збій будь-якого вузла не позначається на роботі БД), підтримується автоматичне відновлення після збою і перенесення навантаження з вузла, який вийшов з ладу. Розширення кластера або перетворення одного сервера в кластер проводиться без зупинки роботи БД простим додаванням нових машин.

Нереляційний підхід досить зручний для створення баз даних, у яких горизонтальне масштабування означає розгортання на множині машин. Можливість забезпечувати найкращу продуктивність повинна існувати паралельно з підтримкою більшої функціональності, ніж це дозволяє використання пар «ключ-значення» (у чистому вигляді). Технологія баз даних має працювати скрізь, починаючи з серверів користувача та віртуальних машин і закінчуючи хмарними технологіями.

MongoDB, на думку розробників, має заповнити розрив між простими сховищами даних типу «ключ-значення» (швидкими і легко масштабованими) і великими РСКБД (зі структурними схемами і потужними запитами).

Основні можливості MongoDB:

- Документо-орієнтоване сховище (проста та потужна JSON -подібна схема даних);
- Досить гнучка мова для формування запитів;
- Динамічні запити;

- Повна підтримка індексів;
- Профілювання запитів;

СКБД управляє наборами JSON-подібних документів, що зберігаються в двійковому вигляді в форматі BSON. Зберігання і пошук файлів в MongoDB відбувається завдяки викликам протоколу GridFS. Подібно до інших документо-орієнтованих СКБД (CouchDB тощо), MongoDB не є реляційною СКБД [5].

Є докладна і якісна документація, велике число прикладів і драйверів під популярні мови Java, C++, C#, PHP, Python, Perl, Ruby.

Важливою функцію системи є кешування. Кешування дозволяє зберігати найбільш часто використовувані дані, таким чином уникаючи запитів до СКДБ або до системи пошуку транспортних маршрутів. Для кешування було вирішено обрати систему Memcached.

Memcached - комп'ютерна програма, що реалізує сервіс кешування даних в оперативній пам'яті на основі парадигми хеш-таблиці.

За допомогою клієнтської бібліотеки (для C / C + +, Ruby, Perl, PHP, Python, Java, CSharp / .Net та ін) дозволяє кешувати дані в оперативній пам'яті з багатьох доступних серверів. Розподіл реалізується шляхом сегментації даних за значенням хеша ключа. Клієнтська бібліотека, використовуючи ключ даних, обчислює хеш і використовує його для вибору відповідного сервера. Ситуація збою сервера трактується як промах кеша, що дозволяє підвищувати відмовостійкість комплексу за рахунок нарощування кількості memcached серверів і можливості робити їх гарячу заміну.

В API memcached є тільки базові функції: вибір сервера, установка і розрив з'єднання, додавання, видалення, оновлення і отримання об'єкта, а також Compare-and-swap. Для кожного об'єкта встановлюється час життя, від 1 секунди до нескінченності. При вичерпанні пам'яті більш старі об'єкти автоматично видаляються.

Для створення веб-серверу було обрано фреймворкnode.js.

Node.js - подієво-орієнтований I/O фреймворк, що використовує двигун javascript V8. Призначений для створення масштабованих мережевих додатків, таких як веб-сервер. Node.js по цілям використання подібний з фреймворкамиTwisted мовою Python і EventMachine на Ruby. На відміну від більшості програм JavaScript, цей фреймворк виконується не в браузері клієнта, а на стороні сервера.

Подієво-орієнтоване програмування (англ. event-drivenprogramming; надалі ПОП) - парадигма програмування, в якій виконання програми визначається подіями - діями користувача (клавіатура, миша), повідомленнями інших програм і потоків, подіями операційної системи (наприклад, надходженням мережевого пакету).

ПОП можна також визначити як спосіб побудови комп'ютерної програми, при якому в коді (як правило, в головній функції програми) явно виділяється головний цикл програми, тіло якого складається з двох частин: вибірки події та обробки події.

Для написання самої системи оптимізації транспортних маршрутів було обрано мову **PHP** завдяки її динамічності, наявності великої кількості додаткових бібліотек та підтримці великою кількістю розробників.

Синтаксис PHP подібний синтаксису мови C. Деякі елементи, такі як асоціативні масиви і цикл foreach, запозичені з Perl.

Для роботи програми не потрібно описувати будь-які змінні або використовувані модулі. Будь-яка програма може починатися безпосередньо з оператора PHP.

PHP виконує код, що знаходиться усередині обмежувачів, таких як <?php?>. Все, що знаходиться поза обмежувачами, виводиться без змін. В основному це використовується для вставки PHP-коду в HTML-документ.

PHP підтримує широкі об'єктно-орієнтовані можливості, повна підтримка яких була введена в п'ятій версії мови [6].

Клас в PHP оголошується за допомогою ключового слова class. Методи і поля класу можуть бути загальнодоступними (public, за замовчуванням), захищеними (protected) і прихованими (private). PHP підтримує всі три основних механізми ООП - інкапсуляцію, поліморфізм і успадкування (батьківський клас вказується за допомогою ключового слова extends після імені класу). Підтримуються інтерфейси (ставляться у відповідність з допомогою implements). Дозволяється оголошення фінальних, абстрактних методів і класів. Множинне успадкування класів

не підтримується, проте клас може реалізовувати декілька інтерфейсів. Для звернення до методів батьківського класу використовується ключове слово parent.

Для візуалізації маршрутів, що є результатом системи пошуку транспортних маршрутів було обрано мову для розмітки векторної графіки SVG.

ScalableVectorGraphics (SVG) (з англ. масштабовна векторна графіка) — специфікація мови розмітки що базується на XML та формат файлів для двовимірної векторної графіки, як статичної, так і анімованої та інтерактивної. SVG може бути виключно декларативним, або містити описи сценаріїв. Ця специфікація є відкритим стандартом, розробленим робочою групою SVG WorkingGroup організації WorldWideWebConsortium [7].

Властивості та переваги формату:

- текстовий формат — файли SVG можна читати і редагувати за допомогою звичайних текстових редакторів, працювати з SVG без засобів візуального програмування не складніше ніж з HTML, при прогляданні документів SVG, що містять графіку, є доступ до проглядання коду файлу, що проглядається, і можливість збереження всього документа, крім того, SVG файли зазвичай виходять менше за розміром, чим порівнянні за якістю зображення у форматах JPEG або GIF, а також добре піддаються стисненню;
- масштабованість — SVG є векторним форматом. Існує можливість збільшити будь-яку частину зображення SVG без втрати якості, додатково, до елементів SVG документа можливо застосовувати фільтри — спеціальні модифікатори для створення ефектів, подібних вживаним при обробці растрових зображень (розмиття, витискування, складні системи трансформації тощо), в тексті SVG-коду фільтри описуються тегами, візуалізацію яких забезпечує засіб перегляду, що не впливає на розмір початкового файлу, забезпечуючи при цьому необхідну ілюстративну виразність;
- широко доступне використання растрової графіки в SVG документах, можливість вставляти елементи із зображеннями у форматах PNG, GIF або JPG;
- текст в графіці SVG є текстом, а не зображенням, тому його можна виділяти і копіювати, він індексуються пошуковими машинами, не потрібно створювати додаткові метафайли для пошукових серверів;
- анімація реалізована в SVG за допомогою мови SMIL (SynchronizedMultimediaIntegrationLanguage), розробленої також консорціумом W3C. Підтримуються скриптові мови на основі специфікації ECMAScript. SVG-елементами можна управляти за допомогою JavaScript, застосування скриптів і анімації в SVG дозволяє створювати динамічну і інтерактивну графіку, у SVG забезпечується подієва модель, відстежуються події (завантаження сторінки, зміна її параметрів, події миші, клавіатури тощо). анімація може запускатися по певній події (наприклад «onmouseover» або «onclick»), що додає графіці інтерактивність, к кожного елементу є свої власні події, до яких можна прив'язувати окремі скрипти;
- SVG — відкритий стандарт, на відміну від деяких інших форматів, SVG не є чиєюсь власністю;
- SVG документи легко інтегруються з HTML і XHTML документами, зовнішні SVG підключаються через тег <embed>, значення атрибуту src ім'я файлу з розширенням «.svg», що містить розмітку SVG, атрибути width і height визначають розміри області SVG по-горизонталі і по-вертикалі, елементи SVG сумісні з HTML і DHTML;
- сумісність з CSS (англ. CascadingStyleSheets) — відображенням (форматуванням і декоруванням) SVG елементів можна управляти за допомогою таблиці стилів CSS 2.0 і її розширень, або безпосередньо за допомогою атрибутів SVG елементів.

Висновки.

Було розглянуто поняття штучного інтелекту та алгоритмів пошуку оптимальних маршрутів, розкрито поняття оптимізації маршруту та пошуку шляху. Було досліджено поняття графу та про алгоритми пошуку, що працюють на графі.

Для вибору підходящого алгоритму було проведено емпіричну оцінку обраних алгоритмів. Спираючись на результати оцінки та отримані дані, було обрано алгоритм, як найбільш підходящий. Також було обрано та описано про необхідні засоби та технології, які необхідні при розробці повноцінної системи.

1. Иванов Н.Б. Дискретная математика. Алгоритмы и программы. Расширенный курс [Текст]. - М. : Известия, 2011. - с. 512.
2. Что такое Яндекс.Карты[Електронне видання] // Yandex. — Електр. дан. — Режим доступу: http://help.yandex.ru/maps/.
3. HowStuffWorks "2-D Trilateration"[Електронне видання] // HowStuffWorks. — Електр. дан. — Режим доступу: http://electronics.howstuffworks.com/gadgets/travel/gps1.htm.
4. Josuttis Nicolai M. The C++ Standard Library: A Tutorial and Reference (2nd Edition) [Текст]. - NY, New York : Addison-Wesley, 1999.
5. M Media Type registration for image/svg+xml[Електронне видання] // World Wide Web Consortium. — Електр. дан. — Режим доступу: http://www.w3.org/TR/SVGMobile12/mimereg.html.
6. Pathfinding[Електронне видання]. — Електр. дан. — Режим доступу: http://en.wikipedia.org/wiki/Pathfinding.
7. Scalable Vector Graphics (SVG) 1.2 W3C Working Draft § 4.2-Drawing Order[Електронне видання] // World Wide Web Consortium. — Електр. дан. — Режим доступу: http://www.w3.org/TR/2002/WD-SVG12-20021115/#drawingorder.