HiPerA*

Paweł Świder, Modelowanie i symulacja systemów

1. Analiza problemu i dziedziny.

Problemem, którym będziemy się zajmować jest implementacja efektywnego algorytmu znajdowania najkrótszych ścieżek w grafie w symulacji ruchu miejskiego na wielką skalę. Problem jest rozwiązywalny algorytmicznie gdzie do najprostszych algorytmów można algorytm Dijkstry, lub algorytmy heurystyczne takie jak GreedyBFS, GreedyBFS jest oparty o prostą heurystykę, szybszy niż

Dijkstra ale może dawać błędne wyniki. Algorytm Dijkstry poświęca czas na eksploracje nieobiecujących kierunków, natomiast GreedyBFS daje złe wyniki, połączeniem zalet obu algorytmów jest A* który jest podobny do Dijkstry, posiada jednak heurystykę która pozwala mu na poruszanie się w optymalnym kierunku i znajdywanie poprawnej trasy. Warunkiem gwarantującym to że A* zwróci poprawną trasę (najkrótszą) jest heurystyka która nie estymuje większego kosztu na dotarcie do celu niż w rzeczywistości. Poza tym heurystyka może być dowolna.

Algorytmem ulepszającym A* jest HPA (Hierarchical pathfinding A*) – polegające na stworzeniu hierarchii obszarów, najpierw wyznaczamy trasę na największym poziomie abstrakcji, potem schodzimy na niższe obszary i tam znajdujemy bardziej szczegółową trasę. Taka metoda działania jest kilka razy szybsza niż zwykły A*, może dawać nieco mniej optymalne wyniki. Takie podejście wymaga jednak dodatkowych obliczeń w celu stworzenia hierarchii – jest to jednak koszt jednorazowy.

Do wyszukiwania drogi dla pojazdu powstały specjalizowane algorytmu takie jak:

SHPA* - lepsza wydajność pamięciowa i czasowa, przeznaczona dla stałych grafów.
DHPA* - więcej pamięci, szybsze obsługiwanie zapytań w porównaniu do HPA
SHP - Significant path based Hub Pushing – używające Hub Labelling
LPA* - Livelong planning A*, wagi się zmieniają wraz z czasem (wolniejszy od HPA)
transit node routing – technika, pozwalająca przyspieszyć znajdowanie ścieżek poprzez wcześniejsze obliczenie niektórych tras
HHL – Hierarchical Hub Labelling

Odnośniki do prac naukowych znajdują się tutaj: [hiperastar/Analiza Problemu I Dziedziny.docx at main · TheTryton/hiperastar (github.com)]

Przystępne wprowadzenie do tematu: [Introduction to the A* Algorithm (redblobgames.com)]

2. Analiza i wybór narzędzi:

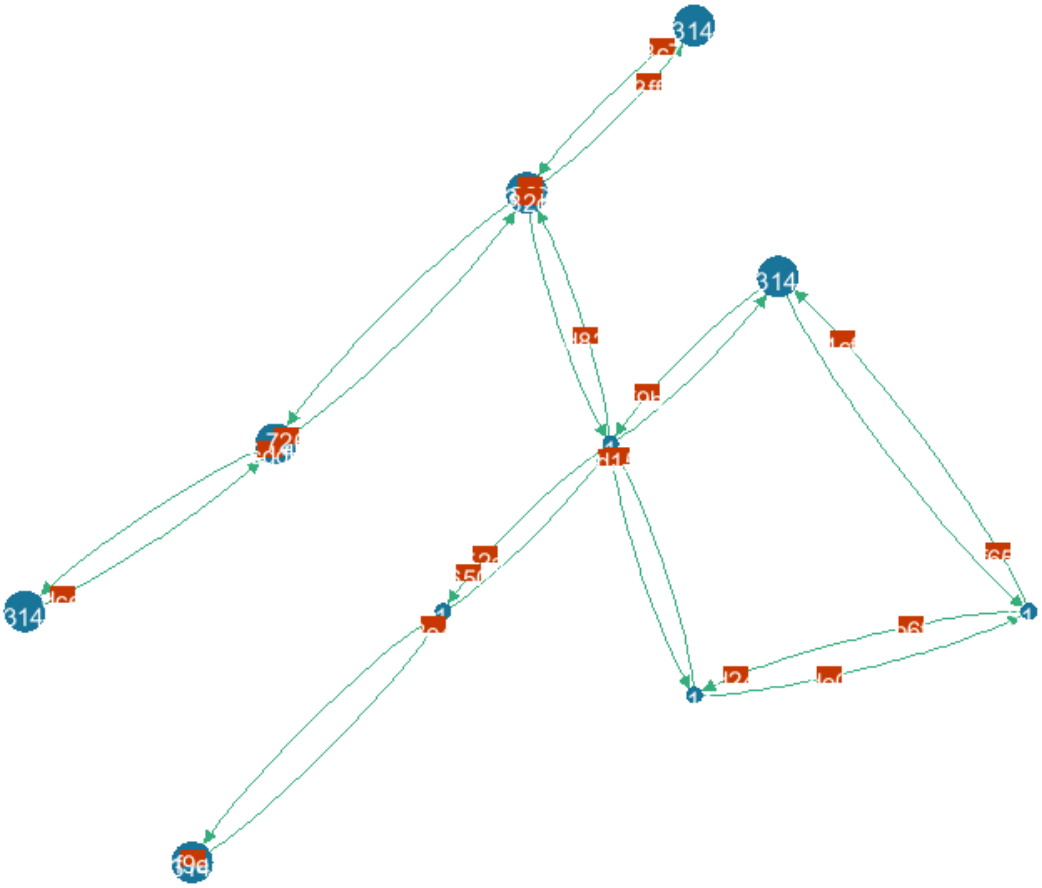
Odpalenie i analiza HiPUTS

W celu odpalenie HIPUTS należało zmodyfikować plik: settings.json ustawić wartości:

```
"testMode":false
"mapPath": "src\\test\\resources\\minimalMap.osm"
"pauseAfterStep":10,
```

Wyłączają one testMode, ustawiają ścieżkę do pliku osm z mapą a także przespieszają animację.

Działanie można symulatora widać poniżej:



Obecnie ścieżka jest losowana w funkcji:

```
private RouteWithLocation generateRoute(LaneId startLaneId, int hops)
```

która jest wywoływana przez funkcję:

```
public Car generateCar(double position, LaneId startLaneId, int hops,
double length, double maxSpeed)
```

Odpalenie I analiza poprzedniego projektu:

W projekcie znajduje się infrastruktura potrzebna do tworzenia grafów, generowaniu losowych punktów (start, koniec) oraz kilka algorytmów znajdywania najkrótszej ścieżki jak np.:

```
- ContractionHierarchyBidirectionalDijkstra
- TransitNodeRoutingShortestPath
- AStarShortestPath
```

Całość korzysta z biblioteki JGraphT, specjalizującej się w algorytmach grafowych. Algorytm który w niej nie występuje i którego była próba implementacji polega na zamienieniu Dijkstry na AStara:

```
ContractionHierarchyAStarShortestPath
```

Jednak ten kod zawiera wiele błędów uniemożliwiających jego skompilowanie.

Dokumentacja JGraphT:

Overview (JGraphT : a free Java graph library)

Oprócz tego jest możliwość zaimplementowania algorytmu HPA* przykładowy link:

GitHub - Maceris/HPAStar: A java implementation of the HPA* algorithm