

## Wydział Informatyki i Telekomunikacji

## Sztuczna Inteligencja i Inżynieria Wiedzy

## Lista nr 1

# Jakub Krupiński 255356

### Wprowadzenie

#### Algorytm Dijkstry

Funkcja kosztu, która została tutaj użyta to czas pomiędzy czasem przyjazdu do stacji końcowej a czasem obecnym (w sekundach).

#### Algorytm A\* - kryterium czasu

Funkcja kosztu, która została została tu użyta to czas pomiędzy czasem przyjazdu do stacji końcowej a czasem obecnym (w sekundach), do którego doliczona została wartość odległości Manhattan pomiędzy przystankiem początkowym a końcowym pomnożona o 1000 w celu odpowiedniego wyskalowania wyniku.

#### Algorytm A\* - kryterium przesiadek

Funkcja kosztu, która została tu użyta to czas pomiędzy czasem przyjazdu do stacji końcowej a czasem obecnym (w sekundach), do którego doliczana jest wartość ‘50’, jeśli wymagana będzie przesiadka.

#### Algorytm A\* - optymalizacje

* Zaimplementowanie kolejki priorytetowej w oparciu o stertę
* Zaimplementowanie grafu w oparciu o słowniki, w celu uzyskania stałego czasu wyszukiwania wartości
* Odrzucanie tych krawędzi, których czas odjazdu jest wcześniejszy niż obecny czas
* Odrzucenie tych krawędzi, których czas odjazdu jest później niż 2h od czasu obecnego

### Klasy użyte w implementacji

#### Node – wierzchołek grafu, który zawiera w sobie nazwę przystanku, oraz jego długość i szerokość geograficzną.

#### Edge – krawędź grafu, określa połączenie między dwoma wierzchołkami. Zawiera w sobie indeks, nazwę ‘firmy’ (np. MPK Wrocław), wierzchołki: startowy i końcowy, czas odjazdu oraz czas przyjazdu.

#### Graph – klasa reprezentująca pełen graf, zawiera w sobie słowniki, które przypisują wierzchołki oraz krawędzie do odpowiadającym im nazwom przystanków.

#### DataReader – klasa, która służy mi do odczytania i odpowiedniego wypakowania zawartości pliku ‘connection\_graph.csv’ do obiektów klas ‘Node’, ‘Edge’ oraz ‘Graph’.

### Założenia i problemy

Dla ułatwienia założyłem, że wszystkie przystanki o tej samej nazwie można połączyć w jeden przystanek.

Ze względu na brak czasu nie udało mi się zaimplementować przeszukiwania Tabu.

### Analiza wyników

W ramach analizy otrzymanych przeze mnie wyników stworzyłem funkcję, która zapisuje wyniki przeprowadzonych przeze mnie symulacji do plików o rozszerzeniu .csv. Dla każdego algorytmy wyniki zapisywane są oddzielnie w dwóch miejscach, w dwóch postaciach – pliki „{rodzaj\_algorytmu}\_output.csv” zapisują pełną trasę wraz z godzinami odjazdów i przyjazdów z i do poszczególnych przystanków, natomiast pliki „{rodzaj\_algorytmu}\_runtimes.csv” zawierają postaci uproszczone, gdzie widoczne są jedynie: rodzaj algorytmu, czas szukania rozwiązania oraz parametry wejściowe (przystanek początkowy, końcowy i czas początkowy). Dane które zostają zapisane w plikach nie są pełnymi zestawami – na koniec są one skracane o 5% najszybszych i najwolniejszych czasów wykonania, żeby wyeliminować przypadki skrajne.

Przykładowa zawartość pliku „dijkstra\_outputs.csv”:

Obraz zawierający stół

Opis wygenerowany automatycznie

Przykładowa zawartość pliku „dijkstra\_runtimes.csv”:

Obraz zawierający stół

Opis wygenerowany automatycznie

Na podstawie zawartości plików „...runtimes.csv” wygenerowałem następujące wykresy, na których widać jak rozkładały się czasy szukania ścieżki pomiędzy iteracjami:

Połączenie ze sobą danych z trzech tych plików zaowocowało tak wyglądającym wykresem punktowym:

Nie jest to idealna reprezentacja tych danych, która nie jest może najbarzdziej przejrzysta, widać jednak na niej wciąż parę zastanawiających rzeczy:

Po pierwsze, wyniki algorytmu A\* na podstawie przesiadek radzi sobie najgorzej ze wszystkich, jako że najczęściej zdarzają się w jego przypadku wartości 0.0 oraz float(‘inf’) (które – na potrzeby wygenerowania wykresu – zostały przeze mnie wyfiltrowane, przez co niebieskich kropek jest na tym wykresie mniej), jako że algorytm ten w mojej implementacji nie zawsze z sukcesem znajdywał odpowiednią ścieżkę. Algorytm Dijkstra radzi sobie zaskakująco dobrze, co również sugeruje, że być może zaproponowane przeze mnie rozwiązanie A\* w kryterium czasu nie należy do optymalnych, jako że zaimplementowanie heurystyki powinno było spowodować znacznie krótsze czasy przeszukiwania.

Ponizej pokazane sa przykladowe wyniki tej samej trasy dla algorytmow A\* z kryterium przesiadkowym oraz czasu:



Jak widac na zdjeciu, kryterium przesiadkowe pozwala na dotarcie na miejsce nieco pozniej, jednak pozwala tez na zmniejszenie ilosci przesiadek – cztery przesiadki zamiast pieciu.