Optymalizacja przejazdów liniami autobusowymi

Dawid Glinkowski 266509

Marzec 2024

1 Wstęp

Otrzymano dane w postaci pliku .csv zawierające przystanek początkowy, następny przystanek (wraz z położeniem) oraz czas wyruszenia i dojechania.

W danych występują przystanki o tej samej nazwie aczkolwiek mają one inne koordynaty. Powoduje to problemy chociażby w postaci symulacji przesiadek. Zidentyfikowano 2 strategie rozwiązujące ten problem:

- Symulacja przesiadek na podstawie skalowania odległości od przystanków o tej samej nazwie
- Potraktowanie grupy przystanków jako jednego przystanku o uśrednionych koordynatach

Obie strategie mają swoje wady oraz zalety. Strategia nr 1 opisuje dużo lepiej wycinek rzeczywistości, jednak znacząco wpłynie na wydajność pamięciową oraz obliczeniową. Dodatkowo strategia ta nie uwzględnia przeszkód takich jak np. przekroczenie jezdni, gdyż jedyną informacją jest odległość między przystankami. Natomiast w przypadku strategii nr 2 traktujemy przystanki jako jeden przez co przesiadki w krótkim interwale czasowym, są niewykonywalne. Zaletą strategii grupowania i uśredniania jest jej prostota, która znacząco upraszcza problem. Rozważając zalety i wady przedstawionych strategii zdecydowano się wybrać strategię numer 2.

2 Implementacja grafu

Graf zaimplementowano w języku python z użyciem słownika łączącego nazwę przystanku z klasą Node która zawiera dane:

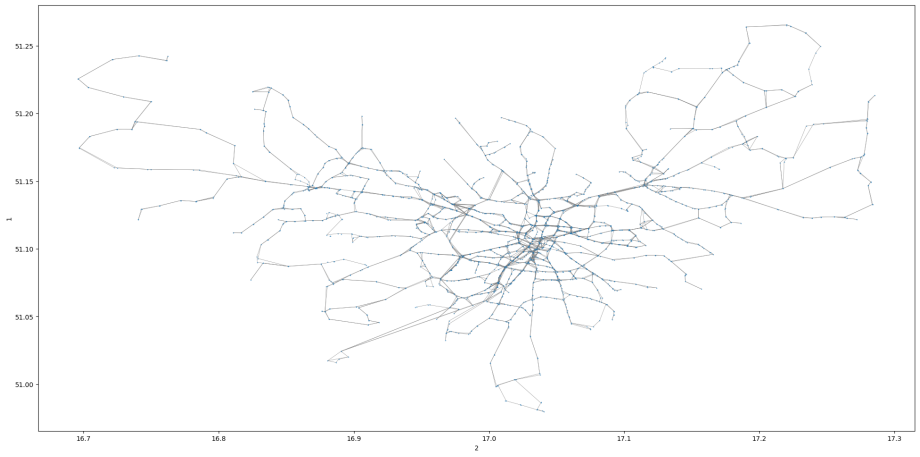
- lokalizację przystanku
- listę krawędzi wychodzących w postaci klasy Edge

Klasa realizująca krawędź - Edge zawiera:

- referencję na przystanek docelowy

- datę przyjazdu
- datę odjazdu
- nr lini
- firmę realizująca trasę

Po wczytaniu danych do grafu utworzono mapę połączeń, przedstawiwoną poniżej:



Rysunek 1: Mapa połączeń

3 Algorym Dijkstry

Zaimplementowano 3 wersje algorytmu dijkstry:

- z własną implementacją kolejki priorytetowej z użyciem LinkedList, algorytm polega na zachłannym wybieraniu najszybszej ścieżki do następnego przystanku i umieszczeniu przystanku. wraz z czasem dotarcia w kolejce. Jest to rozwiązanie pierwotne, pozostałe powstały w celu optymalizacji
- kolejką priorytetową stworzoną na liście z użyciem modułu heapq, dodawano wszystkie krawędzie w o ile ścieżka była szybsza od pozostałych
- kolejką priorytetową z modułu queue - PriortyQueue podobnie jak w przypadku heapq.

Następnie przeprowadzono testy w celu określenia najlepszego algorytmu pod względem odwiedzanych krawędzi, prędkości działania, długości ścieżki. Do tego celu przeprowadzono testy na losowo wybranej próbie o wielkości 50 elementów. Badano czas obliczania ścieżki. śr. liczbę połączeń na trasę (liczba krawędzi),

Wariant	Śr. czas oblicz.	Śr. l. połącz.	Śr. l. sprawdz. przyst.
Wł. kolejka	1.6272	38.38	466.28
Heapq	0.7717	22.22	411.08
Priority queue	0.8573	22.22	411.08

Tabela 1: Porównanie wariantów alg. Dijkstry

liczbę sprawdzanych przystanków. Każdą metodę testowano z wykorzystaniem tych samych danych. Dane testowe wraz z wynikami zostały zamieszczone w folderze extras pod nazwą dijkstra_benchmark.md. W Tabeli nr 1. przedstawiono podsumowanie testów

Najszybszym algorytmem został bezwątpienia algorytm z użyciem modułu heapq. Nieco gorsze okazało się użycie klasy priority queue, zaś najgorsza zarówno pod względem prędkości obliczeń i jakości odnalezionej ścieżki został algorytm oparty na własnej implementacji kolejki. Wybrano heapq i na jego podstawie opracowywano algorytm A*.

4 Algorytm A*

Bazową heurestyką jest funkcja obliczająca odległość między przystankami za pomocą przejścia ze współżędnych sferycznych do układu kartezjańskiego. Przyjęto, że Ziemia jest idealną kulą o promieniu R = 6 371km. Również pominięto krzywiznę planety.

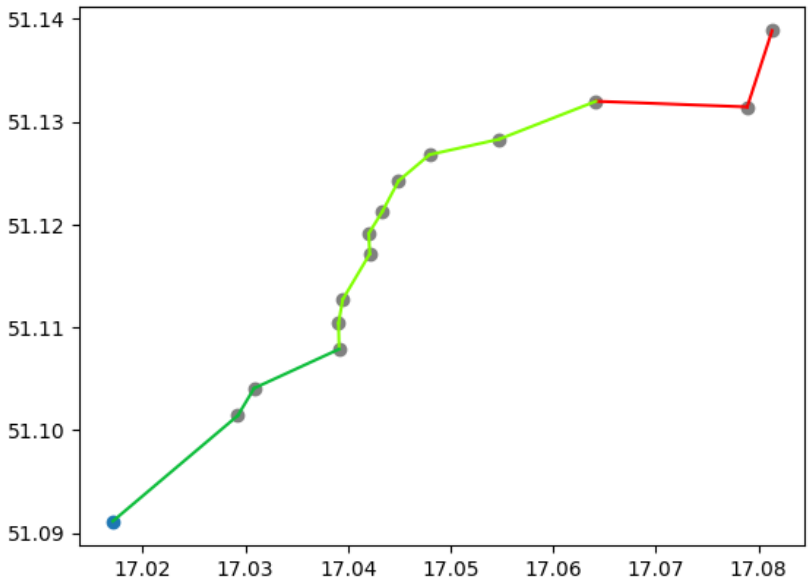
Wartość heurestyki również przeskalowano w celu porównywalności z funkcją kosztu. Uruchomiono testy porównujące działanie dla różnych wartości skali heurestyki (0, 500, 1500, 2000). Dane testowe wraz z wynikami umieszczono w folderze extras w pliku a_star_benchmark.md

	Dijkstra	A*(0)	A*(500)	A*(1500)	A*(2000)
Czas całk.	40.6551	51.8720	10.1340	6.2721	7.5610
Śr. czas na trasę	0.8131	1.0374	0.2027	0.1254	0.1512
Śr. l. spraw. przys.	407.42	407.42	85.06	54.3	53.38
Śr. l. poł.	22.24	22.24	22.34	23.32	23.02
Nieznaleziono	5	5	5	5	5

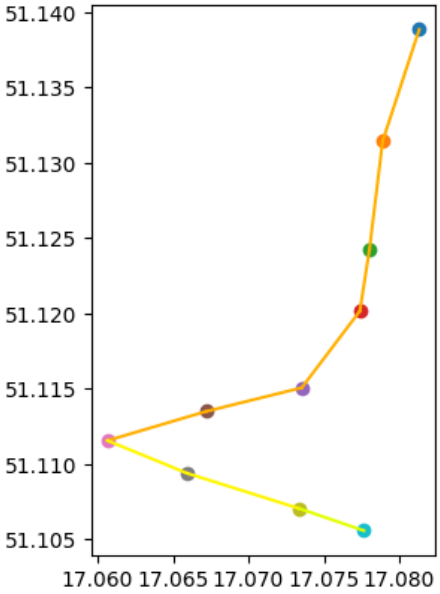
Tabela 2: Porównanie wydajności względem skali w heurystyce A*

Można zauważyć że wraz ze wzrostem skali heurestyki czas obliczeń oraz liczba sprawdzanych przystanków znacząco maleje, natomiast zwiększa śr. liczbę połączeń, czyli przystanków przez które trzeba przejechać. Algorytm A* dla kryterium czasu z wagą 0 upraszcza się do algorytmu Dijkstra, jednak ze względu na obliczanie heurystyki trwa on ok. 25% dłużej

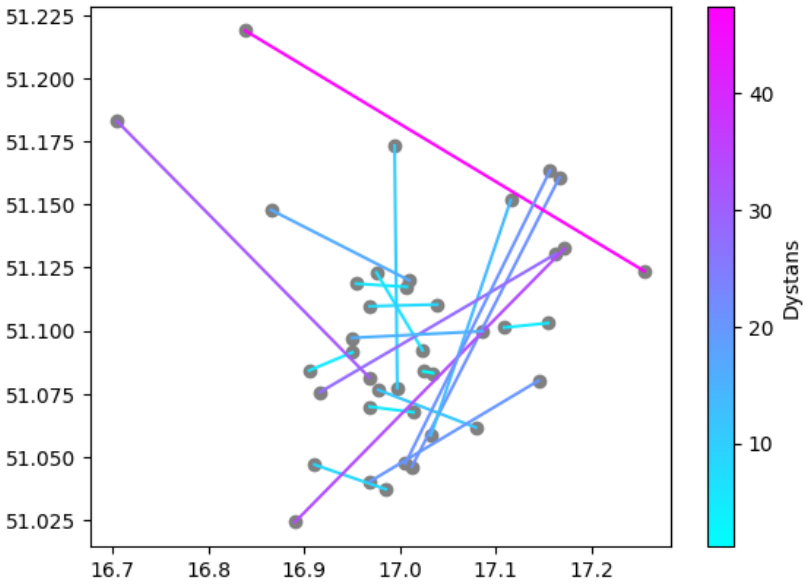
Na rysunkach 2 i 3 zamieszczono przykładowe trasy dla algorytmu A* ze skalą 1500, pokolorowano linie na grafie w celu rozróżnienia zaznaczono kolorem
.



Rysunek 2: Połączenie Brucknera Rondo o godzinie 16:00 względem czasu



Rysunek 3: Połączenie Brucknera ZOO o godzinie 21:00 względem czasu



Rysunek 4: Dane testowe dla wyzn. wagi heurestyki

4.1 Wybór suboptymalnej wagi heurestyki

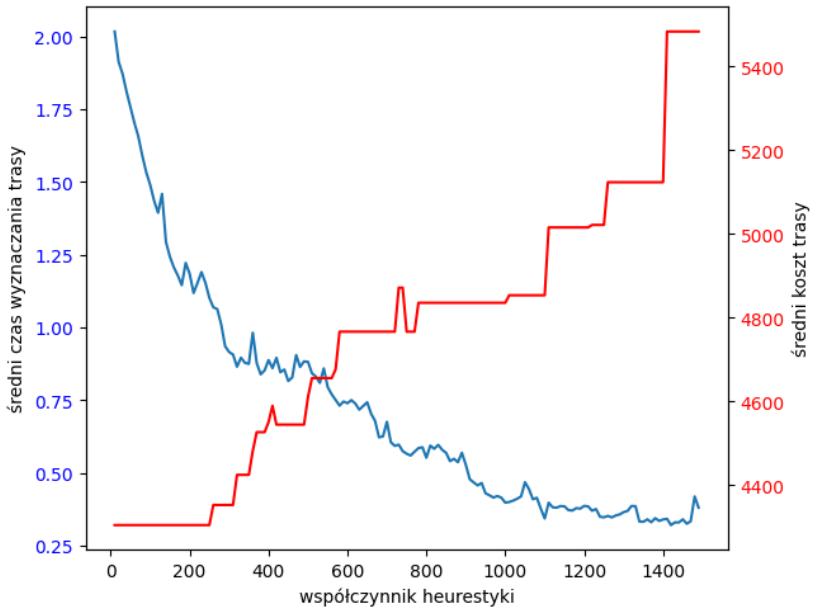
Na nieco mniejszej próbie 20 testów wykonano zestawienie uwzględniając tym razem średni koszt trasy (czyli liczbę sekund od początku przejazdu). Dla każdej skali testy wykonano na tych samych danych. Wyniki przedstawiono na wykresach zależności od skali heurestyki (wartości od 10 do 1500 ze zmianą skali o 10). Dane testowe zamieszczono w extras/a_star_ratio_benchmark.md oraz zamieszczono na grafie (rysunek 4). Na wykresie (rysunek 5) przedstawiano zależności.

Czas obliczania trasy spada wykładniczo, kosztem wydłużającego się czasu przejazdu. Największy średni koszt trasy jest o 27.3% większy od rozwiązania optymalnego. Jednak skraca czas obliczeń około 6 krotnie.

W celu wybrania skali heurestyki przyjęto, że zwiększenie kosztu trasy nie może przekroczyć 15% wartości wynikającej z działania algorymu Dijkstry (skala heurestyki = 0). Wybrano wartość skali 1100 zwiększającą koszt trasy średnio o około 13% w zamian przyspieszającą obliczenia około 5.8 krotnie.

Uwaga: Dane testowe zostały zweryfikowane pod kątem istnienia rozwiązania dopiero po wykonaniu testów. Wpływają one na czas wykonwyania testów i są uwzględnione wliczone w ramach średnich Dane bez rozwiązania:

- *Wysoka do Mirków - Kiełczowska o 19:02:00*
- *Kadłub NŻ do Solskiego o 21:50:00*



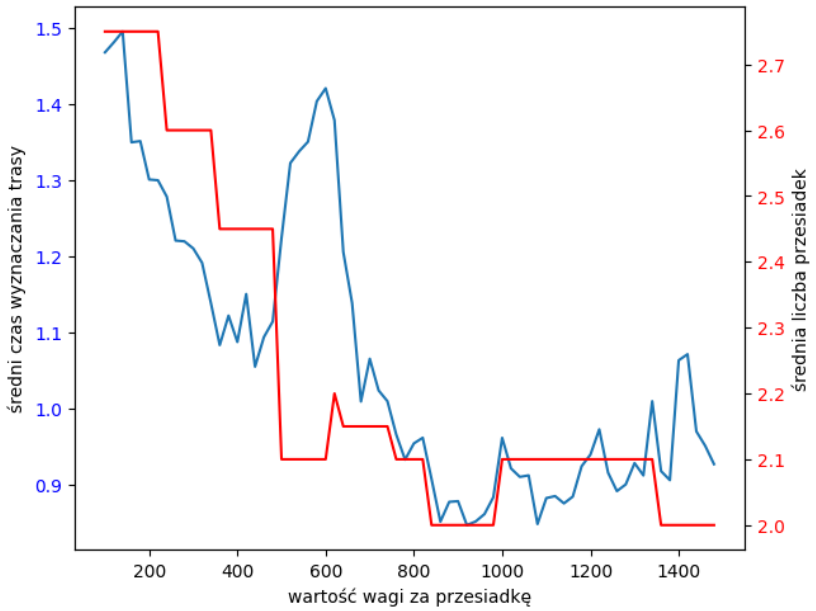
Rysunek 5: Śr. czas wyznaczania trasy i śr. koszt trasy w zależności od skali heurestyki

4.2 Kryterium przesiadkowe

Na podstawie wybranej skali heurestyki, odpowiednio przeskalowano karę za przesiadki. Przesiadki są wybierane z pominięciem kryterium czasu (poza czasami dotarcia na przystanek). Podobnie jak w przypadku wyznaczania wagi funkcji estymacji w A* wykonano testy. Również na tych samych danych. Jednak w celu przyspieszenia działań zmodyfikowano zakres - od 100 do 1500 z krokiem równym 20.

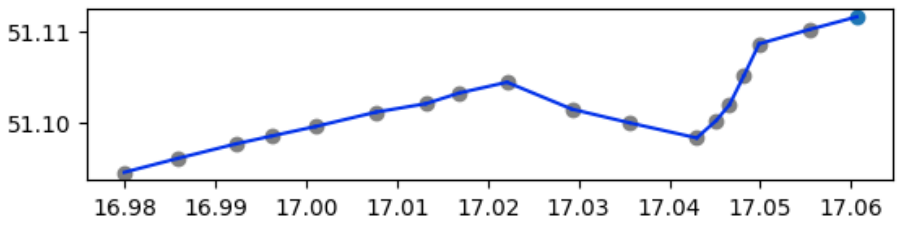
Na rysunku 6. zamieszczono wykres zależności śr. liczby przesiadek i śr. czasu wyznaczania trasy w zależności od liczby przesiadek. Uwagę skupia zakres kary od 800 do 1000, który zarówno minimalizuję liczbę przesiadek jak i czas wykonywania obliczeń.

Wybrano zatem skalę osiągającą minimalną wartość czasu w podanym przedziale - 920



Rysunek 6: Śr. czas wyznaczania trasy i śr. liczba przesiadek w zależności od skali heurestyki

Przykład dla kryterium przesiadkowego o godzinie 12:00 trasa z FAT na PL. GRUNWALDZKI



Rysunek 7: Wynik działania kryterium przesiadkowego FAT - PL. GRUNWALDZKi o 12:00

4.3 Podział danych testowych

W celu dalszej analizy wyników zdecydowano się podzielić dane testowe na kategorie względem odległości między przystankami:

- bliskie $d(n_1, n_2) \leqslant 3km$
- średnie $3km \leqslant d(n_1, n_2) \leqslant 7km$

- dalekie $7km \leqslant d(n_1, n_2)$

Dane testowe dla każdej z kategorii umieszczono w plikach w folderze extras pod nazwami a_star_close.md , a_star_medium.md, a_star_far.md. Dla każdej z kategorii utworzono 100 testów. W tabeli nr 3 przedstawiono wyniki.

	Bliskie	Średnie	Dalekie
Czas całk.	10.4361	21.6628	61.8605
Śr. czas na trasę	0.2166	0.2166	0.6186
Śr. l. spraw. przys.	21.68	39.51	69.80
Śr. l. poł.	6.24	22.24	25.98
Śr. koszt [s]	1690.8	2755.8	4930.8
Nieznaleziono	5	7	14

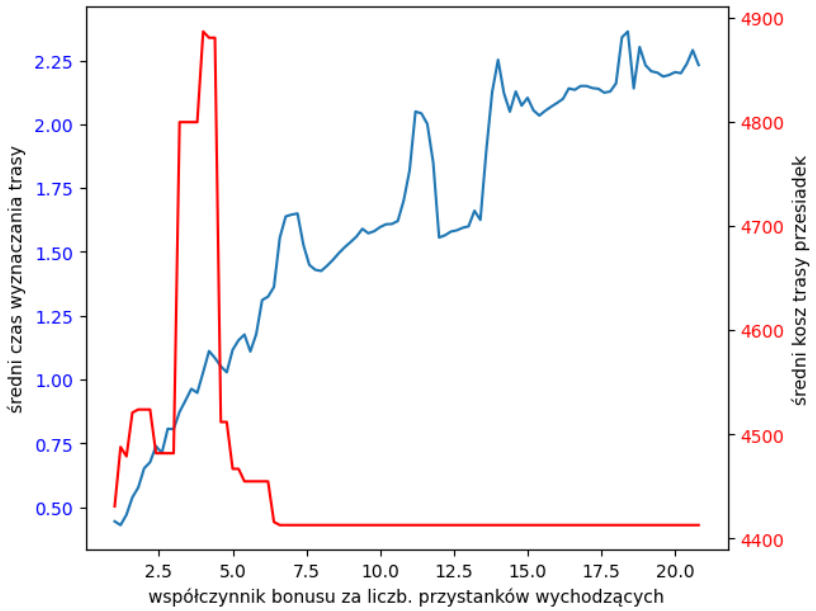
Tabela 3: Wyniki danych testowych kategoryzowanych

4.4 Usprawnienie heurestyki

Usprawnienia heurestyki korzysta z idei Hubu. Oznacza to, że funkcja heurestyczna uwzględnienia liczbę krawędzi wychodzących w przystanku docelowym i preferuje te wartości, które zawierają ich dużą ilość. Zatem heurestykę można określić jako:

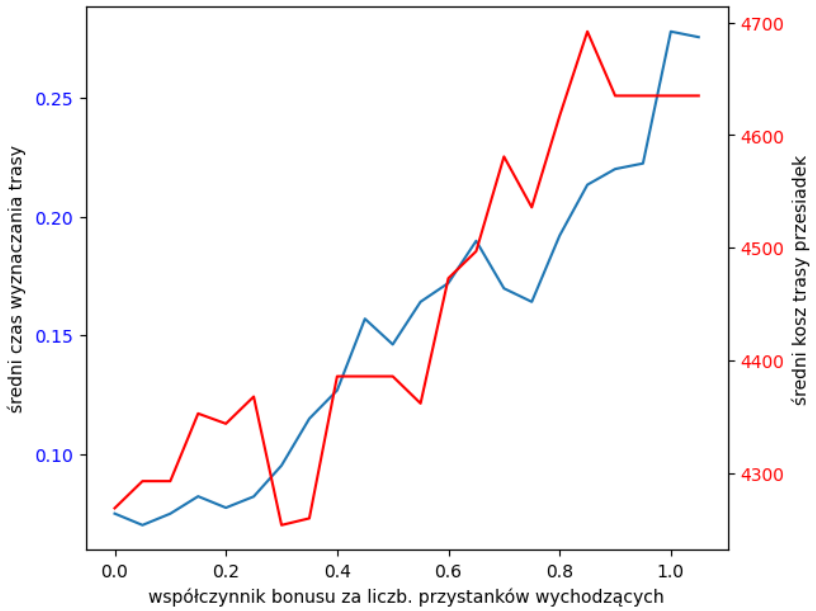
$$h(n_i) = \alpha * distance(n_i, n_e) - \beta * len(n_i.neighbours)$$

Wybór suboptymalnej wartości bonusu będzie przebiegał podobnie jak w przypadku wyboru innych parametrów. Nagroda za liczbę krawędzi wychodzących będzie szukana na przedziale od 1 do 21 włącznie z krokiem 0.2. Zbiorem danych testowanych będzie 20 elementowy losowo wybrany zbiór. A następnie po wybraniu parametru przetestowany z wartościami testowymi kategoryzowanymi względem odległości i porównany z bazową heurystyką.



Rysunek 8: Śr. czas wyznaczania trasy i śr. liczba przesiadek w zależności od wagi bonus za liczbę krawędzi wychodzących

Wraz ze wzrostem współczynnika bonusu, czas obliczania trasy. Natomiast przy wartości współczynnika równym ok. 6 koszt osiąga minimum i utrzymuje się na tym poziomie. Istnieje możliwość, że dla dużych wartości heurestyki bounsu, lokalizacja nie jest brana pod uwagę. Dlatego zdecydowano się powtórzyć testy zmieniając przedział badanych wag na wartości od 0.0 do 1 z krokiem 0.5.

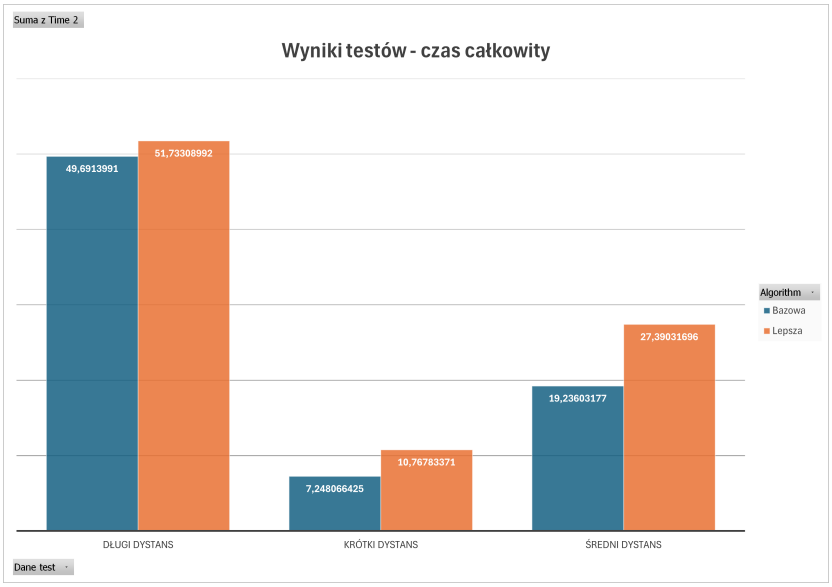


Rysunek 9: Śr. czas wyznaczania trasy i śr. liczba przesiadek w zależności od wagi bonus za liczbę krawędzi wychodzących przedział od 0.0 do 1 z krokiem 0.05

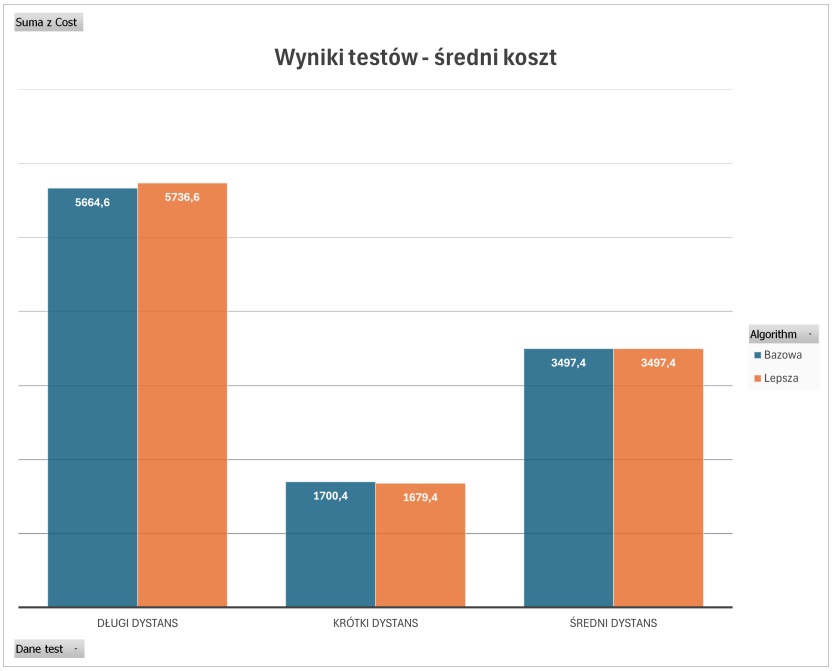
W przypadku nizszych monożników heurestyka w wartościach 0.3 i 0.35 osiąga lepsze wartości funckji kosztu zwiększając czas. Do testów i porównania z dystansami zdecydowano się zastosować wartość 0.35

4.4.1 Porównanie heurestyk

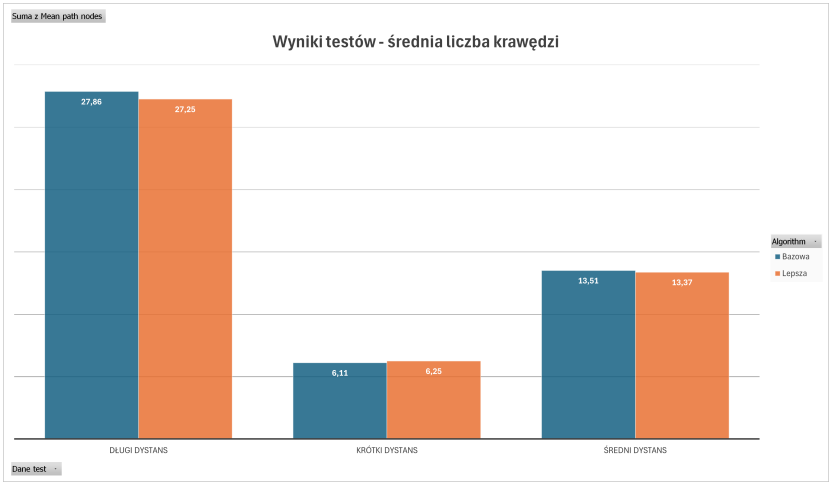
Heurestyki porównano dzieląc testy na kategorie tras krótko, średnio i długo dystansowych. Dane wygenerowano losowo, oraz niesprawdzano istnienia rozwiązania. Dane dla obu algorytmów były takie same.



Rysunek 10: Porównanie czasu działania heurestyki bazowej i usprawniającej



Rysunek 11: Porównanie funckji kosztu heurestyki bazowej i usprawniającej



Rysunek 12: Porównanie średniej liczby krawadędzi w rozwiązaniach heurestyki bazowej i usprawniającej

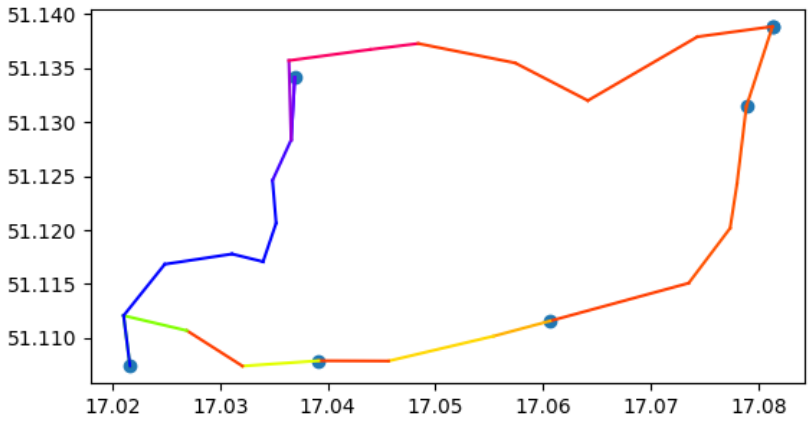
Z wyników testów wynika, że heurestyka niesprawdza się dla tras długo dystansowych i osiąga podobny wynik w trasach średnio dystansowych. Dla tras krótkich osiągneła lepszy wynik niż heurestyka bazowa. Heurestyka usprawniająca zwiększa nieznacznie czas działania algorytmu A*. Co ciekawe heurestyka usprawniająca w przypadkach rozwiązań średnio i długo dystansowych wybiera połączenia mające mniej przystanków po drodze.

Podsumowując rozszerzenie heurestyki sprawdza się jedynie na krótkich dystansach do 3km oraz zwiększa czas działania algorytmu.

5 Wyszukiwanie Tabu

W 1. wersji rozwiązania stworzono algorytm z użyciem tablicy hashującej (set) w celu sprawdzania zbioru tabu w czasie O(1). Również niewprowadzono ograniczenia na wielkość tablicy. W tablicy Tabu zapamiętywane są pełne rozwiązania. Dodatkowo cache'owana jest część rozwiązań tzn. wyniki dla krotki (przystanek_początkowy, przystanek_końcowy, czas_odjazdu) w celu eliminacji powtarzających się obliczeń

Na rysunku 13 przedstawiono wynik działania wyszukiwania Tabu na zbiorze przystanków: Brucknera, Kwidzyńska, PL. GRUNWALDZKI, GALERIA DOMINIKAŃSKA, PL. ORLĄT LWOWSKICH, KARŁOWICE o godzinie 9:00. Pokolorowano linie w celu rozróżnienia. Wartości metaparametrów $\gamma = 10$ powtórzeń dla maksimów globalnych, $\delta = 5$ przejść wgłąb po sąsiadach. Całkowity koszt 4740



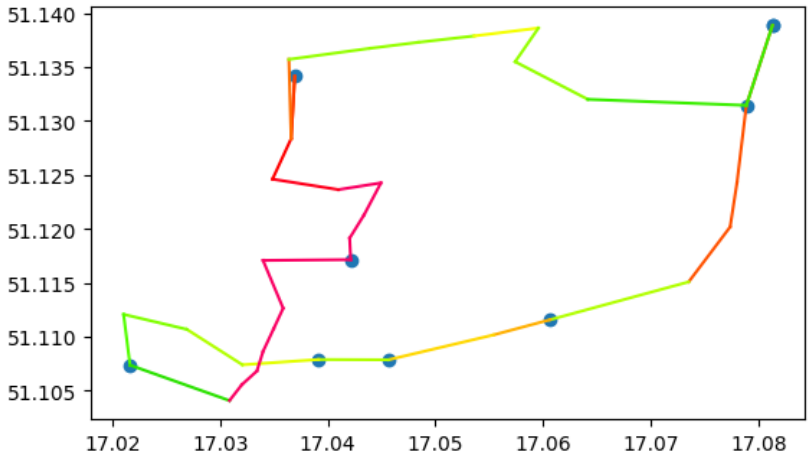
Rysunek 13: Rozwiązanie dla przystanków Brucknera, Kwidzyńska, PL. GRUNWALDZKI, GALERIA DOMINIKAŃSKA, PL. ORLĄT LWOWSKICH, KARŁOWICE, Brucknera o godzinie 9:00

5.1 Wersja 2

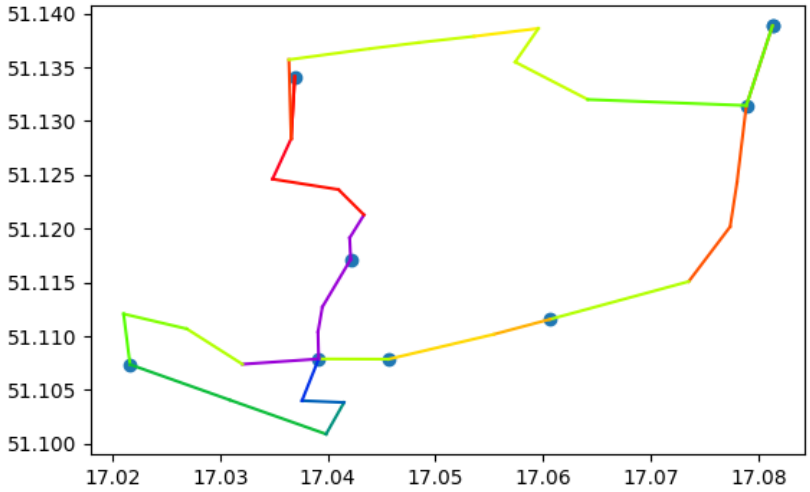
W wersji 2 zamiast zapisywania w tablicy Tabu rozwiązań całkowitych, zapisywane są operacje tzn. zamiany przystanków, oraz wprowadzono parametr ograniczający wielkość tablicy. Zdecydowano się również zamienić tablice hashującą na listę w celu zapamiętywania kolejności dodanych elementów

Do porównania wersji skorzystano z przystanków: Brücknera Kwidzyńska PL. GRUNWALDZKI Poczta Główna GALERIA DOMINIKAŃSKA pl. Orląt Lwowskich KARŁOWICE pl. Bema Brücknera o godzinie 9:00, algortym uruchomiono z paramterami $\gamma = 10, \delta = 20$, a dla wariantu 2 użyto długości tablicy tabu $\tau = 5$

Wariant 1 uzyskał koszt równy 5340, wariant 2 również uzyskał koszt 5340 z nieco inną trasą



Rysunek 14: Rozwiązanie 1 wariant dla $\gamma = 10, \delta = 20$



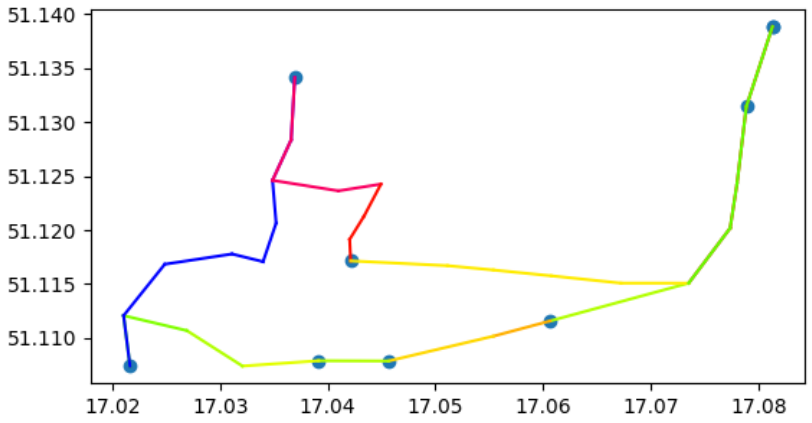
Rysunek 15: Rozwiązanie 2 wariant dla $\gamma = 10, \delta = 20, \tau = 5$

Wariant 1 obliczył trasę w czasie około 35s, zaś wariant 2 w około 13s. Wariant 2 zwraca koszt z bardzo dużą wariancją tzn. przystanki są losowane na przedziale od 4860, aż do około 6500

5.2 Strategia próbkowania sąsiedztwa bieżącego rozwiązania

Zdecydowano się zmienić strategię dobierania rozwiązania. Obecnie rozwiązania zwracane są wszystkie rozwiązania wynikające z zamiany miejscami 2 przystanków. Zdecydowano się zmniejszyć tę liczbę do losowania jedynie n wartości, gdzie n to liczba przystanków. Po wprowadzeniu losowania sąsiedztwa koszt trasy zmalał do 4860 i przy ponownych uruchomieniach algorytmu wartość, ta jest bardziej stabilna i rzadziej wzrasta. Na rysunku 16 przedstawiono wynik.

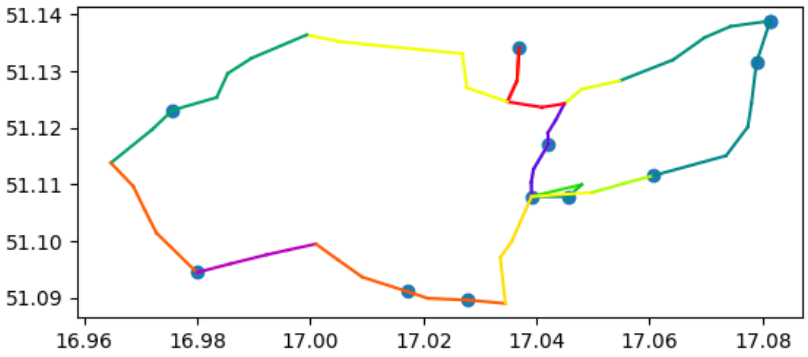
Zauważono, że wartości długości tabu zwracają najlepsze wyniki w przypadku rozmiaru tablicy tabu o 1 mniejszą od liczby przystanków permutowanych (tzn. w przypadu ciągu $(A_0, A_1, ...A_n, A_0)$ wartość tablicy tabu wyniesie $\tau = n - 1$)



Rysunek 16: Rozwiązanie 2 wariant z losowym wyborem sąsiedztwa dla $\gamma = 10, \delta = 20, \tau = 5$

5.3 Test dla dłuższych ciągów wartości

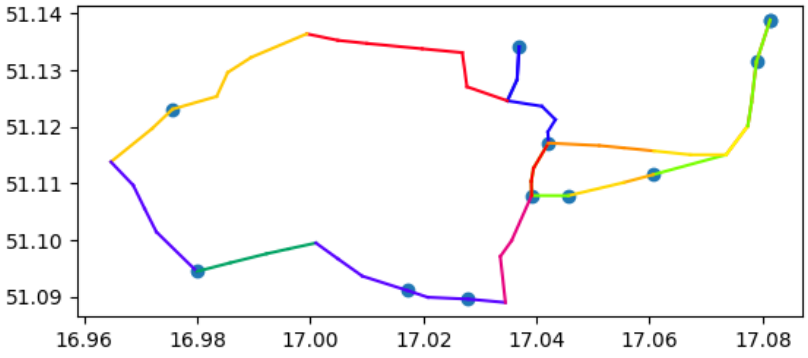
Wykonano test na przystankach Brücknera, pl. Bema, Poczta Główna, KARŁOWICE, Na Ostatnim Groszu, FAT, Rondo, Uniwersytet Ekonomiczny, GALERIA DOMINIKAŃSKA, PL. GRUNWALDZKI, Kwidzyńska, Brücknera (przystanki w kolejności uzyskanego rozwiązania) Dla tego problemu istnieje $10! = 3628800$ permutacji. Wyszukiwanie tabu parametryzowane $\gamma = 30, \delta = 15, \tau = 9$, wykonane zostało 1min 38.5s z kosztem 7860.



Rysunek 17: Test na 10 przystankach, 2 wariant z losowym wyborem sąsiedztwa dla $\gamma = 30, \delta = 15, \tau = 9$

Rozwiązanie nie jest idealne, występuje zbędny przejazd z PL. Bema na Pocztę Główną (przez galerię dominikańską), która jest odwiedzana ponownie.

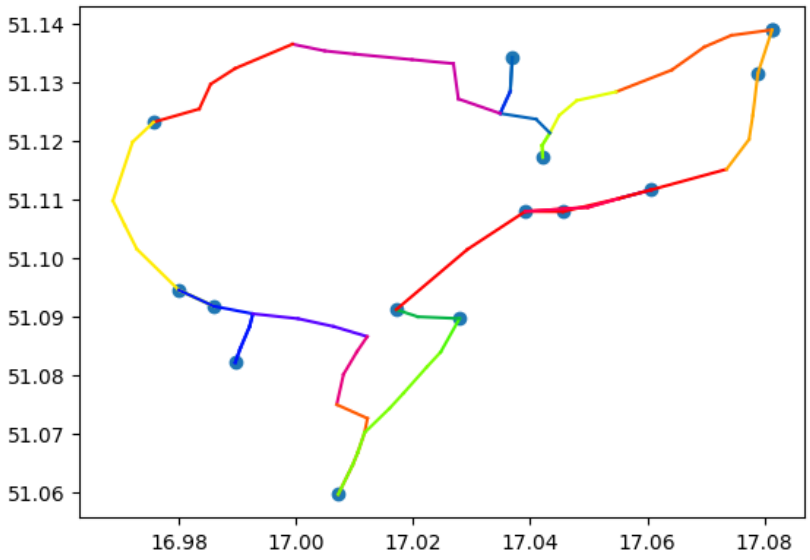
Sprawdzono również rozwiązanie dla 1. wersji tabu rozważającej jedynie całkowite rozwiązania. Na tych samych parametrach uruchomiono i uzyskano trasę: Brücknera, Kwidzyńska, PL. GRUNWALDZKI, Poczta Główna, KARŁOWICE, Na Ostatnim Groszu, FAT, Rondo, Uniwersytet Ekonomiczny, GALERIA DOMINIKAŃSKA, pl. Bema, Brücknera Co ciekawe koszt trasy wyszedł dokładnie taki sam, czyli 7860. Uzyskane rozwiązanie jest również bardzo podobne. Również występuje zbędne połączenie (Pl. Bema z galerią dominikańską). Jednak czas obliczeń był dłuższy - wyniósł on 2min 21s.



Rysunek 18: Test na 10 przystankach, 1 wariant dla $\gamma = 30, \delta = 15$

Sprawdzono dodatkowo w celu sprawdzenia wzrostu długości obliczeń trasę: Brücknera pl. Bema KARŁOWICE Na Ostatnim Groszu Aleja Pracy FAT BLACHARSKA rondo Św. Ojca Pio Uniwersytet Ekonomiczny Rondo GALERIA DOMINIKAŃSKA PL. GRUNWALDZKI Poczta Główna Kwidzyńska

Brücknera w teście zmieniono jedynie rozmiar tablicy tabu $\tau = 12$. Obliczenia zostały wykonane w 2min 32s. Uzyskano koszt równy 10500.



Rysunek 19: Test na 13 przystankach permutowanych, 2 wariant z losowym wybieraniem rozwiązań dla $\gamma = 30, \delta = 15, \tau = 12$

6 Podsumowanie

Czas działania algorytmu A* jest skorelowany ze spadkiem jakości rozwiązania. Przy projektowaniu algorytmu należy wziąć pod uwagę czas w jakim ma działać algorytm i jakość oczekiwanego rozwiązania. Sprawdzono również rozwinięcie heurestyki algorytmu o dodanie liczby krawędzi wychodzących w przystanku docelowym. Rozwiązanie jednak nie spełniło oczekiwanych skutków, zmniejszeniu kosztu trasy nastąpiło jedynie w przypadku tras krótkich (do 3 km) i to nieznacznie, natomiast zmniejszeniu uległa liczba odwiedzanych przystanków.

W przypadku tabu search stworzono 2 wersje algorytmu: 1. definiująca tablicę tabu jako pełne rozwiązania, 2. definująca ostatnie zamiany wraz z ograniczeniem liczby pamiętanych zamiany. W celu poprawienia jakości rozwiązania wprowadzono losowe dobieranie sąsiedztwa w liczbie n sąsiadów, gdzie liczba ta oznacza całkowitą liczbę przystanków. Działanie to poprawiły wyniki. Dla większej liczby przystanków rozwiązanie dawało również satysfakcjonujące wyniki zarówno pod względem czasu jak i pod względem jakości rozwiązań(z niewielkimi błędami wynikającymi z wpadania do minimum lokalnego).

7 Biblioteki

W ramach rozwiązania użyto następujących bibliotek:

- pandas - w celu szybkiego załadowania danych z pliku i wstępnej obróbki (uśrednienie lokalizacji przystanków o tych samych nazwach)
- numpy - wspieranie działania pandas oraz generowanie ciągów z krokiem nie będącym liczbą naturalną w przypadku grafu
- heapify - wbudowany moduł w pythonie, do budowy kolejki priotytetowej
- queue - klasa PriorityQueue, do budowy kolejki priotytetowej w wariancie algorytmu dijkstra, odrzucone w dalszych rozwiązaniach
- random - w celu generowania danych testowych oraz w tabu search w tworzeniu początkowego rozwiązania i losowania rozwiązań sąsiadujących
- functools.cache - wbudowany moduł w pythonie, w celu pamiętania cząstkowych obliczeń w przypadku priority queue oraz dystansów między wartościami
- matplotlib - w celu reprezentacji graficznej rozwiązań i analiz