Sztuczna inteligencja i inżynieria wiedzy
Lista1

Agata Belczyk

March 2024

1 Problem szukania najlepszej ścieżki

1.1 Opis problemu

Problem omawiany w pierwszej części tego sprawozdania będzie dotyczył znalezienia najlepszej ścieżki w sieci komunikacyjnej Wrocławia i okolic z punktu A do punktu B o zadanej porze. Modyfikacje, które odróżniają graf prezentowany w zadaniu od prostszych grafów jest to, że waga krawędzi jest zmienna, zależna od czasu. Dostarczone dane należy dostosować do problemu, tak aby łatwo można było znaleźć sąsiadów przystanków oraz połączenia między nimi.

1.2 Założenia

- Nie można poruszać się pomiędzy przystankami inaczej niż autobusem/tramwajem.
- Przystanki są unikatowe pod względem nazwy, pomimo występowania przystanków o tych samych nazwach, z różnymi współrzędnymi. Jako współrzędne przystanku ustalane są pierwsze, które wystąpiły w pliku .csv. Ma to na celu ułatwienie problemu przesiadek. Połączenia z jednakowym początekiem i końcem(taką samą nazwą przystanku) nie będą brane pod uwagę w rozwiązaniu.
- Czas przesiadki jest równy 0.

Algrotymy wykorzystane do rozwiązania problemu to algorytm Dijskstry oraz algorytm A* z kryterium minimalizacji czasu lub przesiadek.

1.3 Algorytm Dijkstry

Algorytm Dijkstry szuka najlepszej ścieżki do każdego wierzchołka grafu. Koszt dotarcia z wierzchołka A do B jest obliczany ze wzoru:

$$kosztB = kosztA + czas\_oczekiwania + czas\_jazdy$$

Szukanie najlepszej ścieżki jest bardziej czasochłonne niż w przypadku algorytmu A*. Jego zaletą jest jednak to, że gdybyśmy chcieli sprawdzić najlepszy dojazd do innego przystanku o tej samej godzine, moglibyśmy skorzystać z już policzonego rozwiązania.

Oto kilka przykładów połączeń wyliczonych przez algorytm.
⋆ Nr → Nazwa_przystanku oznacza, że dla akutalnego przystanku, jest to przystanek następny do którego jedzie pojazd.

Przykład 1:
start = "Rogowska (P+R)" stop = "FAT" godzina = 23:30

```
Znaleziono trasę
('wsiąść', '23 ->  Strzegomska (krzyżówka)', datetime.time(23, 30), 'Rogowska (P+R)')
('wysiąść', 23, datetime.time(23, 33), 'Nowodworska')
('wsiąść', '106 ->  MUCHOBÓR MAŁY (Stacja kolejowa)', datetime.time(23, 36), 'Nowodworska')
('wysiąść', 106, datetime.time(23, 39), 'Gądowianka')
('wsiąść', '143 ->  Szkocka', datetime.time(23, 41), 'Gądowianka')
('wysiąść', 143, datetime.time(23, 47), 'FAT')
Koszt  ('godziny: ', 0, 'minuty: ', 17)
Czas trwania  46.19796681404114
```

Przykład 2:
start = "Złotniki", stop = "Chełmońskiego", godzina = 1:40

```
('wsiąść', '253 ->  Małopolska (Ośrodek dla niewidomych)', datetime.time(1, 46), 'Złotniki')
('wysiąść', 253, datetime.time(2, 16), 'DWORZEC AUTOBUSOWY')
('wsiąść', '247 ->  DWORZEC GŁÓWNY', datetime.time(2, 19), 'DWORZEC AUTOBUSOWY')
('wysiąść', 247, datetime.time(2, 21), 'DWORZEC GŁÓWNY')
('wsiąść', '244 ->  Wzgórze Partyzantów', datetime.time(2, 24), 'DWORZEC GŁÓWNY')
('wysiąść', 244, datetime.time(2, 28), 'GALERIA DOMINIKAŃSKA')
('wsiąść', '241 ->  Urząd Wojewódzki (Impart)', datetime.time(2, 38), 'GALERIA DOMINIKAŃSKA')
('wysiąść', 241, datetime.time(2, 43), 'PL. GRUNWALDZKI')
('wsiąść', '253 ->  Kliniki - Politechnika Wrocławska', datetime.time(2, 44), 'PL. GRUNWALDZKI')
('wysiąść', 253, datetime.time(2, 49), 'Chełmońskiego')
Koszt  ('godziny: ', 1, 'minuty: ', 9)
Czas trwania  43.3592312335968
```

Przykład 3:
start = "Kliniki - Politechnika Wrocławska", stop = "Chełmońskiego", godzina = 12:40

```
('wsiąść', '146 ->  Hala Stulecia', datetime.time(12, 0), 'Kliniki - Politechnika Wrocławska')
('wysiąść', 146, datetime.time(12, 6), 'Chełmońskiego')
Koszt  ('godziny: ', 0, 'minuty: ', 6)
Czas trwania  47.246755599975586
```

Tabela 1: Przykłady wywołań algorytmu Dijkstra

Wnioski:

- Niezależnie od odległości przystanków, algorytm Dijkstry wykonuje obliczenia przez podobną ilość czasu ≈ 44s

- Algorytm nie minimalizuje liczby przesiadek. W przypadku 2. trasa mogłaby być pokonana tylko autobusem nr.253.

1.4 Algorytm A*

Algorytm A* jest optymalizacją algorytmu Dijkstry. Przeszukuje on wierzchołki, które zbliżają sie do celu. Do obliczania optymializacji funkcji kosztu jest używany wzór

$$f = g + h$$

Gdzie:

- g - funkcja kosztu dla kryterium czasu lub przesiadek w sekundach
- h - heurystyka odpowiadająca za estymację kosztu z wierzchołka do celu w sekundach

W obu wersjach występuje ta sama heurystyka, jednak różne funkcje kosztu. W wynikach prezentowanych użyto wzoru Manhattan, wyniki otrzymane przy użyciu wzoru Euklidesa nie różniły się znacząco.
Heurystyka jest obliczana ze wzoru Manhattan:

$$h(\text{curr}, \text{next}) = (|\text{curr.x - next.x}| + |\text{curr.y - next.y}|) \times \frac{degree\_to\_km}{v\_mpk\_km\_s}$$

lub ze wzoru Euklidesa:

$$(\text{curr}, \text{next}) = \sqrt{(\text{curr.x} - \text{next.x})^2 + (\text{curr.y} - \text{next.y})^2} \times \frac{degree\_to\_km}{v\_mpk\_km\_s}$$

Gdzie:

- x,y - wzpółrzędne przystanków w stopniach
- degree_to_km - liczba kilometrów odpowiadająca stopniowi
- v_mpk_km_s - średnia prędkość Wrocławskiego mkp, wynosząca 16.6 km/h

Heurystyka zwraca przybliżenie czasu potrzebnego na pokonanie odległości od aktualnego przystanku, do końcowego przystanku.

1.4.1 A* z kryterium czasowym

Funkcja kosztu jest obliczana tak samo jak w algorytmie Dijkstry. Oto wyniki algorytmu dla tych samych przykładów co Dijkstra.

Przykład 1:
start = "Rogowska (P+R)" stop = "FAT" godzina = 23:30

```
Rozwiazanie znalezione
Liczba sprawdzonych przystanków:  34
('wsiąść', '23 ->  Strzegomska (krzyżówka)', datetime.time(23, 30), 'Rogowska (P+R)')
('wysiąść', 23, datetime.time(23, 33), 'Nowodworska')
('wsiąść', '106 ->  MUCHOBÓR MAŁY (Stacja kolejowa)', datetime.time(23, 36), 'Nowodworska')
('wysiąść', 106, datetime.time(23, 39), 'Gądowianka')
('wsiąść', '143 ->  Szkocka', datetime.time(23, 41), 'Gądowianka')
('wysiąść', 143, datetime.time(23, 47), 'FAT')
Koszt  ('godziny: ', 0, 'minuty: ', 17)
Czas trwania  0.5340569019317627
```

Przykład 2:
start = "Złotniki", stop = "Chełmońskiego", godzina = 1:40

```
Rozwiazanie znalezione
Liczba sprawdzonych przystanków:  208
('wsiąść', '253 ->  Małopolska (Ośrodek dla niewidomych)', datetime.time(1, 46), 'Złotniki')
('wysiąść', 253, datetime.time(2, 16), 'DWORZEC AUTOBUSOWY')
('wsiąść', '247 ->  DWORZEC GŁÓWNY', datetime.time(2, 19), 'DWORZEC AUTOBUSOWY')
('wysiąść', 247, datetime.time(2, 21), 'DWORZEC GŁÓWNY')
('wsiąść', '244 ->  Wzgórze Partyzantów', datetime.time(2, 24), 'DWORZEC GŁÓWNY')
('wysiąść', 244, datetime.time(2, 28), 'GALERIA DOMINIKAŃSKA')
('wsiąść', '241 ->  Urząd Wojewódzki (Impart)', datetime.time(2, 38), 'GALERIA DOMINIKAŃSKA')
('wysiąść', 241, datetime.time(2, 43), 'PL. GRUNWALDZKI')
('wsiąść', '253 ->  Kliniki - Politechnika Wrocławska', datetime.time(2, 44), 'PL. GRUNWALDZKI')
('wysiąść', 253, datetime.time(2, 49), 'Chełmońskiego')
Koszt  ('godziny: ', 1, 'minuty: ', 9)
Czas trwania  5.830362319946289
```

Przykład 3:
start = "Kliniki - Politechnika Wrocławska", stop = "Chełmońskiego", godzina = 12:40

```
Rozwiazanie znalezione
Liczba sprawdzonych przystanków:  16
('wsiąść', '4 ->  Hala Stulecia', datetime.time(12, 41), 'Kliniki - Politechnika Wrocławska')
('wysiąść', 4, datetime.time(12, 46), 'Chełmońskiego')
Koszt  ('godziny: ', 0, 'minuty: ', 6)
Czas trwania  0.3313736915588379
```

Tabela 2: Przykłady wywołań algorytmu A*

Wnioski:

- Im dalej są od siebie oddalone przystanki, tym więcej przystanków jest sprawdzanych. W przykładzie 1: 335, 2: 288. 3: 16.
- Tak samo jak Dijkstra, algorytm nie minimalizuje liczby przesiadek. W przypadku 2. trasa mogłaby być pokonana tylko autobusem nr.253.

1.4.2 A* z kryterium przesiadkowym

Funkcja kosztu jest zmodyfikowaną funkcją liczenia kosztu w Dijkstrze. Liczy ona koszt tak jak poprzednio, jednak sprawdza także, czy połączenie wymaga przesiadki i dodaje karę w wysokości 1000 sekund do kosztu. Oto wyniki algorytmu dla tych samych przykładów co Dijkstra.

Przykład 1: start = "Rogowska (P+R)" stop = "FAT" godzina = 23:30
Rozwiazanie znalezione Liczba sprawdzonych przystanków: 76 ('wsiąść', '132 -> MIŃSKA (Rondo Rotm. Pileckiego)', datetime.time(23, 31), 'Rogowska (P+R)') ('wysiąść', 132, datetime.time(23, 45), 'OPORÓW') ('wsiąść', '4 -> GRABISZYŃSKA (Cmentarz II)', datetime.time(23, 49), 'OPORÓW') ('wysiąść', 4, datetime.time(23, 53), 'FAT') Koszt ('godziny: ', 0, 'minuty: ', 23) Czas trwania 1.3320720195770264
Przykład 2: start = "Złotniki", stop = "Chełmońskiego", godzina = 1:40
Rozwiazanie znalezione Liczba sprawdzonych przystanków: 100 ('wsiąść', '253 -> Małopolska (Ośrodek dla niewidomych)', datetime.time(1, 46), 'Złotniki') ('wysiąść', 253, datetime.time(2, 49), 'Chełmońskiego') Koszt ('godziny: ', 1, 'minuty: ', 9) Czas trwania 2.2611799240112305
Przykład 3: start = "Kliniki - Politechnika Wrocławska", stop = "Chełmońskiego", godzina = 12:40
Rozwiazanie znalezione Liczba sprawdzonych przystanków: 9 ('wsiąść', '4 -> Hala Stulecia', datetime.time(12, 41), 'Kliniki - Politechnika Wrocławska') ('wysiąść', 4, datetime.time(12, 46), 'Chełmońskiego') Koszt ('godziny: ', 0, 'minuty: ', 6) Czas trwania 0.2666621208190918

Tabela 3: Przykłady wywołań algorytmu A*

Wnioski:

- Korzystanie z kryterium przesiadkowego zmniejszyło liczbę przeszukiwanych przystanków dla każdego przystanku o około połowę, a zatem zmniejszył się czas wyszukiwania.
- Dla przypadku 1 algorytm znalazł trasę z mniejszą liczbą przesiadek, jednak o 6 minut dłuższą niż w A* z kryterium czasu.
- Dla przypadku 2 algorytm znalazł połączenie bez przesiadek, co znacznie poprawiło wynik A* z kryterium czasu, w którym były 4 przesiadki i ten sam czas jazdy.

1.5 Podsumowanie wniosków

Algorytm A* był znacznie szybszy niż algorytm Dijkstry, zwłaszcza dla krótszych tras. Algorytm A* z kryterium przesiadkowym czasem zwaracał identyczne wyniki, jak z kryterium czasowym, a czasem optymalizował liczbę przesiadek. W niektórych przypadkach optymalizacja ta nie zmieniała kosztu podróży, ale w innych wydłużała podróż.

1.6 Problemy implementacyjne

Implementacja została wykonana w języku Python. Wykorzystywane biblioteki to:

1) pandas - wykonywanie operacji na dataframe'ach
2) networkx - tworzenie grafów i wykonywanie na nich operacji
3) numpy - wykonywanie operacji na tablicach, dostarcza wartość nieskończoności
4) datetime - operowanie czasem
5) sys - kierowanie wyników na standardowe wyjście błędów
6) math - działania matematyczne
7) random - generowanie liczb pseudolosowych

Dostarczone dane miały formę pliku .csv. Dane zawierały zduplikowane elementy oraz czas reprezentowany typem String i wykraczający poza 24-godzinny sposób reprezentacji. Czas zamieniono na typ datetime oraz dodano kolumny odpowiadające wartościom czasów w sekundach, w celu łatwiejszego wykonywania operacji na czasie.

Aby umożliwić łatwiejsze przeszukiwanie tych danych zamieniono je na graf skierowany. Użyto w tym celu DiGraph'u z biblioteki networkx.

id	line	departure_time	arrival_time	start_stop	end_stop	start_stop_lat	start_stop_lon	end_stop_lat	end_stop_lon	departure_time_seconds	arrival_time_seconds
0	A	20:52:00	20:53:00	Zajezdnia Obornicka	Paprotna	51.148737	17.021069	51.147752	17.020539	75120	75180
1	A	20:53:00	20:54:00	Paprotna	Obornicka (Wołowska)	51.147752	17.020539	51.144385	17.023735	75180	75240
2	A	20:54:00	20:55:00	Obornicka (Wołowska)	Bezpieczna	51.144385	17.023735	51.141360	17.026376	75240	75300
3	A	20:55:00	20:57:00	Bezpieczna	Bałtycka	51.141360	17.026376	51.136632	17.030617	75300	75420
4	A	20:57:00	20:59:00	Bałtycka	Broniewskiego	51.136632	17.030617	51.135851	17.037383	75420	75540

Rysunek 1: Dane zformatowane

Wierzchołki grafu będą reprezentowane przez nazwę przystanku oraz współrzędne geograficzne. Plik .csv zawierał przystanki, które miały tę samą nazwe, lecz inne współrzędne. Aby umożliwić przesiadki, stworzono graf zawierający unikatowe nazwy przystanków, którym zostały przypisane jedne z odpowiadających temu przystankowi współrzędnych. Dodatkowo w algorytmie A* wierzchołki przechowywują wartości g,h,f.

```
print(graph.nodes.data())
print(len(graph.nodes))
```

✓ 2.4s

```
[('8 Maja', {'lat': 51.11366919, 'lon': 17.09111971}), ('AUCHAN', {'lat': 51.0520648, 'lon': 16.97405354}), ('Adamczewskich', {'lat': 51.12084396,
939
```

Rysunek 2: Reprezentacja przystanków, liczba przystanków to 939

Następnie stworzono krawędzie, które były identyfikowane przez przystanek początkowy i przystanek sąsiedni, natomiast krawędź zawierała 'schedule', czyli listę słowników, zawierających dane o połączeniach między tymi przystankami.

```
for elem in graph['Ostrowskiego']['FAT']['schedule'][0]:
    print(elem + " : " + str(graph['Ostrowskiego']['FAT']['schedule'][0][elem]))
```

✓ 18.6s

```
line : 107
departure_time : 07:11:00
arrival_time : 07:12:00
start_stop : Ostrowskiego
end_stop : FAT
departure_time_seconds : 25860
arrival_time_seconds : 25920
```

Rysunek 3: Przykładowe połączenie między przystankiem Ostrowskiego, FAT

1.7 Zastosowania algorytmów

Algorytmy do znajdywania najkrótszych ścieżek mają szerokie zastosowanie. Poza typowymi dziedzinami, takimi jak **logistyka, transport czy nawigacja**, są one wykorzystywane m.in. **w routingu, grach komputerowych czy z zarządzaniu przepływami wodnymi**. Problem ten reprezentowany jest w postaci grafu, w którym wierzchołki reprezentują pozycje, stany, natomiast krawędzie lub łuki reprezentują koszt przejścia z jednego wierzchołka do drugiego.

2 Problem komiwojażera

2.1 Opis problemu

Problem komiwojażera to rozszerzony problem szukania najkrótszej ścieżki. W tym wypadku należy znaleźć najkrótszą ścieżkę, przechodzącą przez zbiór wierzchołków i wracającą do punktu startowego. Problem komiwojażera można rozwiązać algorytmem Tabu, który wykorzystuje algorytm A*. Algorytm Tabu generuje generuje rozwiązania sąsiednie dla aktualnego i sprawdza ich koszt. Najlepsze rozwiązania trafiają do listy tabu, która jest kolejką FIFO, tzn. rozwiązania, które do niej trafią nie będą sprawdzane ponownie, dopóki nie odczekają swojego czasu w kolejce. Powoduje to, że algorytm nie będzie sprawdzał w kółko tego samego rozwiązania, czyli nie utknie w lokalnym maksimum funkcji.

2.2 Problemy implementacyjne

Aby skrócić czas działania algorytmu, stworzono słownik, którego kluczem jest rozwiązanie(czyli krotka składająca się z nazw ulic), a wartość to koszt.

2.3 Przykładowe wywołania

Oba rozwiązania a) i b) dały najlepszy koszt.

a) start: ”Tyrmanda”

hour: 8: 40

initial_solution: [”Kwiska”, ”FAT”, ”PL. GRUNWALDZKI”,”Złotniki”,”Młodych Techników”]

max_iterations: 8

tabu_list_size: nieograniczona

```
Start:  Tyrmanda Stop:  FAT
('wsiąść', '107 ->  Zagony', datetime.time(8, 41), 'Tyrmanda')
('wysiąść', 107, datetime.time(8, 50), 'FAT')
Start:  FAT Stop:  PL. GRUNWALDZKI
('wsiąść', '11 ->  Hutmen', datetime.time(8, 51), 'FAT')
('wysiąść', 11, datetime.time(9, 9), 'GALERIA DOMINIKAŃSKA')
('wsiąść', 'D ->  Urząd Wojewódzki (Impart)', datetime.time(9, 9), 'GALERIA DOMINIKAŃSKA')
('wysiąść', 'D', datetime.time(9, 15), 'PL. GRUNWALDZKI')
Start:  PL. GRUNWALDZKI Stop:  Młodych Techników
('wsiąść', '4 ->  most Grunwaldzki', datetime.time(9, 15), 'PL. GRUNWALDZKI')
('wysiąść', 4, datetime.time(9, 18), 'Urząd Wojewódzki (Impart)')
('wsiąść', '13 ->  GALERIA DOMINIKAŃSKA', datetime.time(9, 19), 'Urząd Wojewódzki (Impart)')
('wysiąść', 13, datetime.time(9, 32), 'Młodych Techników')
Start:  Młodych Techników Stop:  Kwiska
('wsiąść', '13 ->  pl. Strzegomski (Muzeum Współczesne)', datetime.time(9, 32), 'Młodych Techników')
('wysiąść', 13, datetime.time(9, 33), 'pl. Strzegomski (Muzeum Współczesne)')
('wsiąść', '3 ->  Wrocław Mikołajów (Zachodnia)', datetime.time(9, 34), 'pl. Strzegomski (Muzeum Współczesne)')
('wysiąść', 3, datetime.time(9, 40), 'Kwiska')
Start:  Kwiska Stop:  Złotniki
('wsiąść', '122 ->  Na Ostatnim Groszu', datetime.time(9, 41), 'Kwiska')
('wysiąść', 122, datetime.time(9, 54), 'Chociebuska (C. K. Nowy Pafawag)')
('wsiąść', '148 ->  Rogowska (Ośrodek sportu)', datetime.time(9, 54), 'Chociebuska (C. K. Nowy Pafawag)')
('wysiąść', 148, datetime.time(10, 9), 'Złotniki')
Start:  Złotniki Stop:  Tyrmanda
('wsiąść', '148 ->  Małopolska (Ośrodek dla niewidomych)', datetime.time(10, 12), 'Złotniki')
('wysiąść', 148, datetime.time(10, 30), 'Strzegomska (krzyżówka)')
('wsiąść', '134 ->  Rogowska (P+R)', datetime.time(10, 30), 'Strzegomska (krzyżówka)')
('wysiąść', 134, datetime.time(10, 32), 'Rogowska (P+R)')
('wsiąść', '107 ->  MIŃSKA (Rondo Rotm. Pileckiego)', datetime.time(10, 32), 'Rogowska (P+R)')
('wysiąść', 107, datetime.time(10, 35), 'Tyrmanda')

```

Rysunek 4: Algoryrm Tabu Przykład 1

Koszt: (’godziny: ’, 1, ’minuty: ’, 55), Czas trwania: 10min 7s

b) start: ”Tyrmanda”
hour: 8: 40
initial_solution: [”Kwiska”, ”FAT”, ”PL. GRUNWALDZKI”,”Złotniki”,”Młodych Techników”]
max_iterations: 8
tabu_list_size: 10

```
Start:  Tyrmanda Stop:  FAT
('wsiąść', '107 ->  Zagony', datetime.time(8, 41), 'Tyrmanda')
('wysiąść', 107, datetime.time(8, 50), 'FAT')
Start:  FAT Stop:  PL. GRUNWALDZKI
('wsiąść', '11 ->  Hutmen', datetime.time(8, 51), 'FAT')
('wysiąść', 11, datetime.time(9, 9), 'GALERIA DOMINIKAŃSKA')
('wsiąść', 'D ->  Urząd Wojewódzki (Impart)', datetime.time(9, 9), 'GALERIA DOMINIKAŃSKA')
('wysiąść', 'D', datetime.time(9, 15), 'PL. GRUNWALDZKI')
Start:  PL. GRUNWALDZKI Stop:  Młodych Techników
('wsiąść', '4 ->  most Grunwaldzki', datetime.time(9, 15), 'PL. GRUNWALDZKI')
('wysiąść', 4, datetime.time(9, 18), 'Urząd Wojewódzki (Impart)')
('wsiąść', '13 ->  GALERIA DOMINIKAŃSKA', datetime.time(9, 19), 'Urząd Wojewódzki (Impart)')
('wysiąść', 13, datetime.time(9, 32), 'Młodych Techników')
Start:  Młodych Techników Stop:  Kwiska
('wsiąść', '13 ->  pl. Strzegomski (Muzeum Współczesne)', datetime.time(9, 32), 'Młodych Techników')
('wysiąść', 13, datetime.time(9, 33), 'pl. Strzegomski (Muzeum Współczesne)')
('wsiąść', '3 ->  Wrocław Mikołajów (Zachodnia)', datetime.time(9, 34), 'pl. Strzegomski (Muzeum Współczesne)')
('wysiąść', 3, datetime.time(9, 40), 'Kwiska')
Start:  Kwiska Stop:  Złotniki
('wsiąść', '122 ->  Na Ostatnim Groszu', datetime.time(9, 41), 'Kwiska')
('wysiąść', 122, datetime.time(9, 54), 'Chociebuska (C. K. Nowy Pafawag)')
('wsiąść', '148 ->  Rogowska (Ośrodek sportu)', datetime.time(9, 54), 'Chociebuska (C. K. Nowy Pafawag)')
('wysiąść', 148, datetime.time(10, 9), 'Złotniki')
Start:  Złotniki Stop:  Tyrmanda
('wsiąść', '148 ->  Małopolska (Ośrodek dla niewidomych)', datetime.time(10, 12), 'Złotniki')
('wysiąść', 148, datetime.time(10, 30), 'Strzegomska (krzyżówka)')
('wsiąść', '134 ->  Rogowska (P+R)', datetime.time(10, 30), 'Strzegomska (krzyżówka)')
('wysiąść', 134, datetime.time(10, 32), 'Rogowska (P+R)')
('wsiąść', '107 ->  MIŃSKA (Rondo Rotm. Pileckiego)', datetime.time(10, 32), 'Rogowska (P+R)')
('wysiąść', 107, datetime.time(10, 35), 'Tyrmanda')
```

Rysunek 5: Algoryrm Tabu Przykład 2

Koszt (’godziny: ’, 1, ’minuty: ’, 55), Czas trwania 11min 39s

d) Dobranie strategii próbkowania Dobrana srategia probkówania to wybranie do sprawdzenia spośród sąsiadów losowych 70%. W wyniku zastosowania tej strategii, otrzymano rozwiązanie optymalne w czasie krótszym, równym 6min 46s.

```
('wysiąść', 107, datetime.time(10, 35), 'Tyrm
Koszt  ('godziny: ', 1, 'minuty: ', 55)
Czas trwania  406.72863245010376
```

Rysunek 6: Wynik po dobraniu strategii próbkowania

2.4 Wnioski

Algorytm tabu dla 5 przystanków pośrednich sprawdza połowę możliwych rozwiązań. Możliwych rozwiazań jest 5! = 120, a Tabu sprawdza ok.60. Dla mniej-

szej liczby przystanków pośrednich bardziej opłaca się sprawdzić wszyskie rozwiązania, np dla 4 przystanków pośrednich, rozwiązań jest $4! = 24$. Po dobraniu startegii próbkowania, czas poszukiwania się zmniejszył.

2.5 Zastosowania

Poza poprzednimi:

- **Sekwencjonowanie genów**: W bioinformatyce, algorytm TSP może być stosowany do sekwencjonowania genów, gdzie trzeba znaleźć optymalną kolejność sekwencji nukleotydów w celu zrozumienia struktury genetycznej organizmu.
- **Planowanie tras w turystyce**: W planowaniu podróży turystycznych, gdzie trzeba odwiedzić różne atrakcje turystyczne w możliwie najkrótszym czasie.
- **Projektowanie układów drukowanych**: W projektowaniu układów drukowanych (PCB), gdzie trzeba znaleźć optymalną trasę dla ścieżek przewodzących między elementami układu.