Geoinformatyka, sem 5

Data oddania: 27.12.2017

Prowadzący: Mgr. Jacek Marciniak

Gradkowski Adam

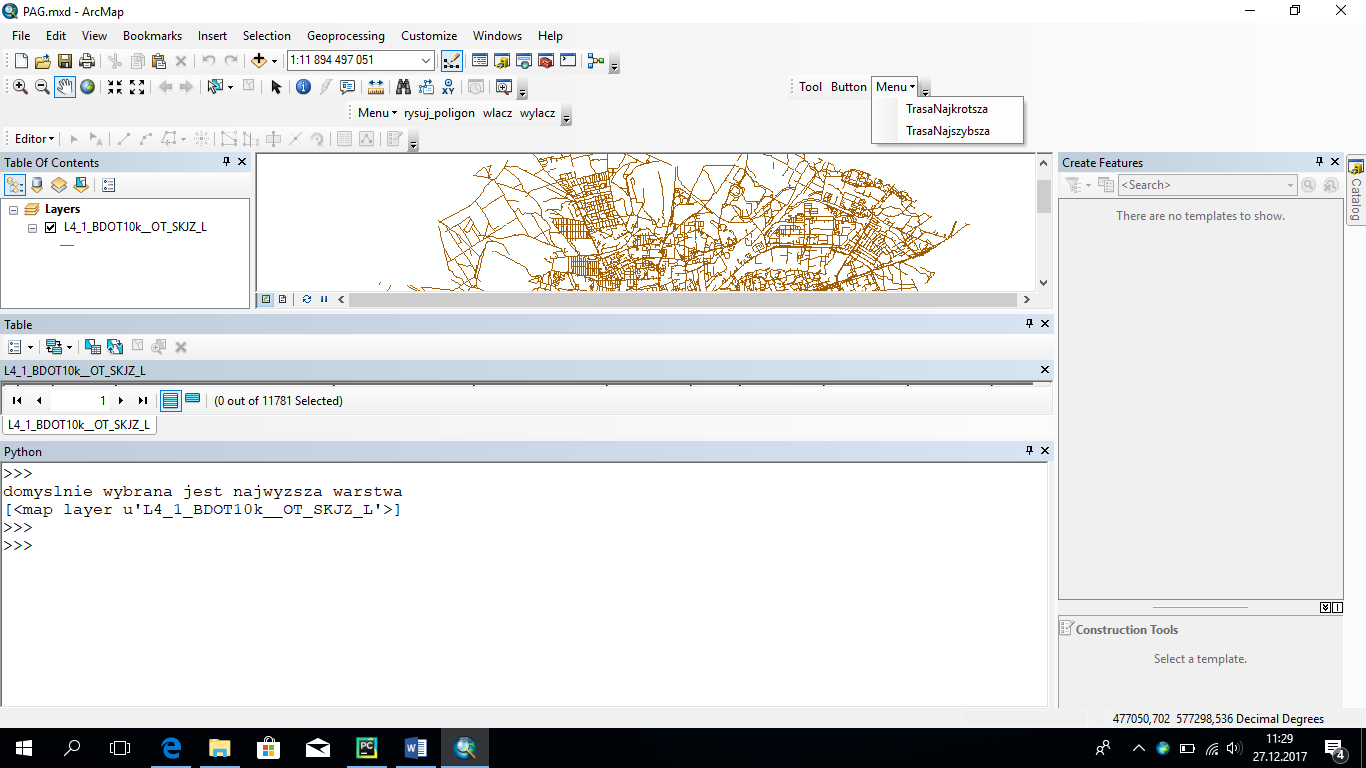
Programowanie Aplikacji Geoinformacyjnych

Temat 2: Trasy Alternatywne

Wstęp.

Projekt składa się z 3 części. Części polegającej na wczytaniu danych z pliku .shp, części obliczeniowej wykorzystującej algorytm A\* z modułu netowrkX oraz części polegającej na przedstawieniu wyników w formie graficznej w programie ArcMapa.

Projekt został wykonany przy użyciu biblioteki ArcMapy Add-ins, służącej do tworzenia dodatkowych narzędzi. Narzędzie stworzone przeze mnie składa się z przycisku - narzędzia służącego do wskazania punktu początkowego na mapie oraz punktu końcowego, oraz menu wyboru obliczenia trasy najkrótszej oraz trasy najszybszej.



Za pomocą modułu arcpy wczytuje dane z pliku .shp. Dane umieszczam do dwóch list :

Pkt = [] : składa się z id oraz współrzędnych x oraz y

Edges = [] : składa się z id , pkt\_od , pkt\_do , odległości oraz maksymalnej prędkości jaka jest przypisywana w zależności od klasy drogi z pliku .shp

Następnym istotnym krokiem było wyszukanie najbliższych punktów od punktów zaznaczonych narzędziem. Wyszukane zostają dwa punktu, które następnie przy obliczaniu trasy są punktami startowym oraz końcowym.

Cześć obliczeniowa została wykonana w funkcji *CreateGraph* , w której należy podać punkt startowy, końcowy, wierzchołki, krawędzie oraz typ wyliczonej trasy jaki chcemy wykonać (ustawiamy go za pomocą narzędzia Menu). Algorytm pochodzi z modułu networkX.

<https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.shortest_paths.astar.astar_path.html>

Po zapoznaniu się z dokumentacją w prosty sposób można użyć algorytmu. Pierwszym kokiem jest stworzenie obiektu, następnie wczytanie wierzchołków oraz krawędzi w zależności od typu wyliczanej trasy. Funkcja *dist* odpowiada za tak zwaną heurestykę. W przypadku trasy najkrótszej obliczana jest odległość za pomocą współędznych. Natomiast w przypadku trasy najszybszej do wszystkich dróg przyjąłem średnią prędkość poruszania się po drogach w miastach.

path = nx.astar\_path(G, start, end, dist, weight='weight')

Wynikiem algorytmu jest lista punktów które zostały obliczone w grafie.

Kolejnym krokiem jest wyszukanie odpowiednich krawędzi oraz stworzenie odpowiedniej kwerendy do wyszukania odcinków w celu późniejszej ich przedstawienia.

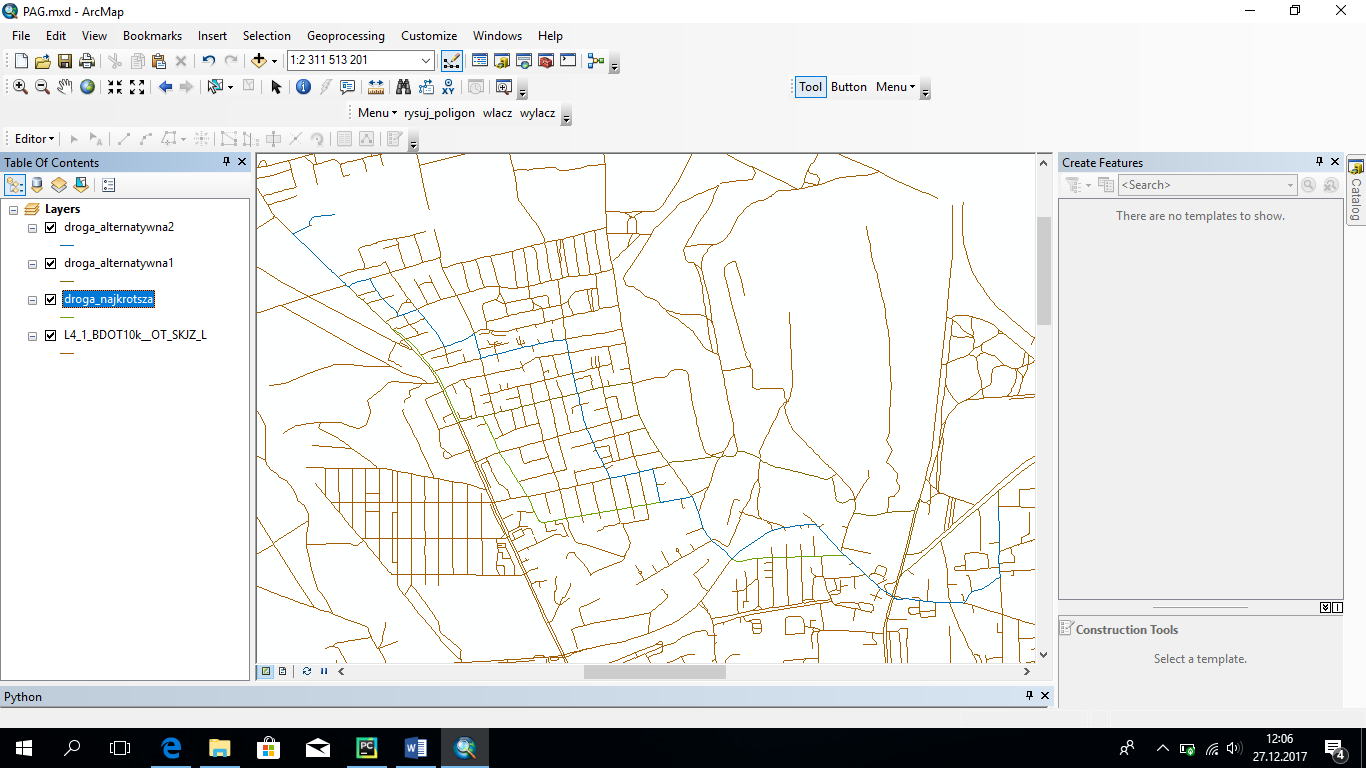
for i in range(0, len(path) - 1):  
 tab.append(G[path[i]][path[i + 1]]['id'])  
query = '"FID" IN ('  
for i in tab:  
 query = query + str(i) + ', '  
query = query[:-2]  
query += ')'

Ostatnich istotnym krokiem było rozwiązanie problemu „Tras Alternatywnych”. Nawiązałem się do sugestii prowadzącego o zwiększeniu wartości kosztów w krawędziach przy każdej następnej iteracji. Tzn iż po wyszukaniu odpowiednich krawędzi, wartości dla nich zostaną zwiększone, a następnie zostanie stworzona trasa alternatywna.

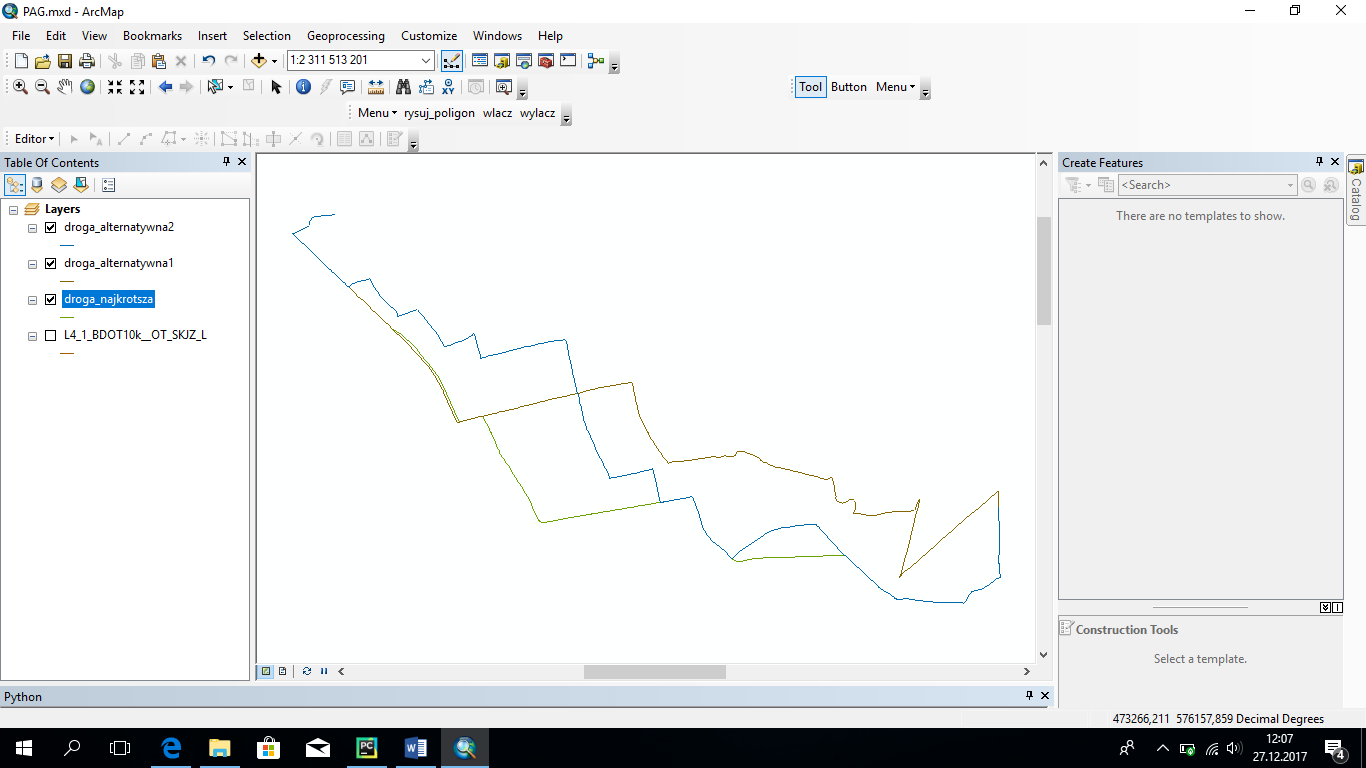
for t in tab:  
 for e in edge:  
 if t == e[0]:  
 e[3] \*= 1.3  
 time += ((e[3] / 1000) / e[4]) \* 60

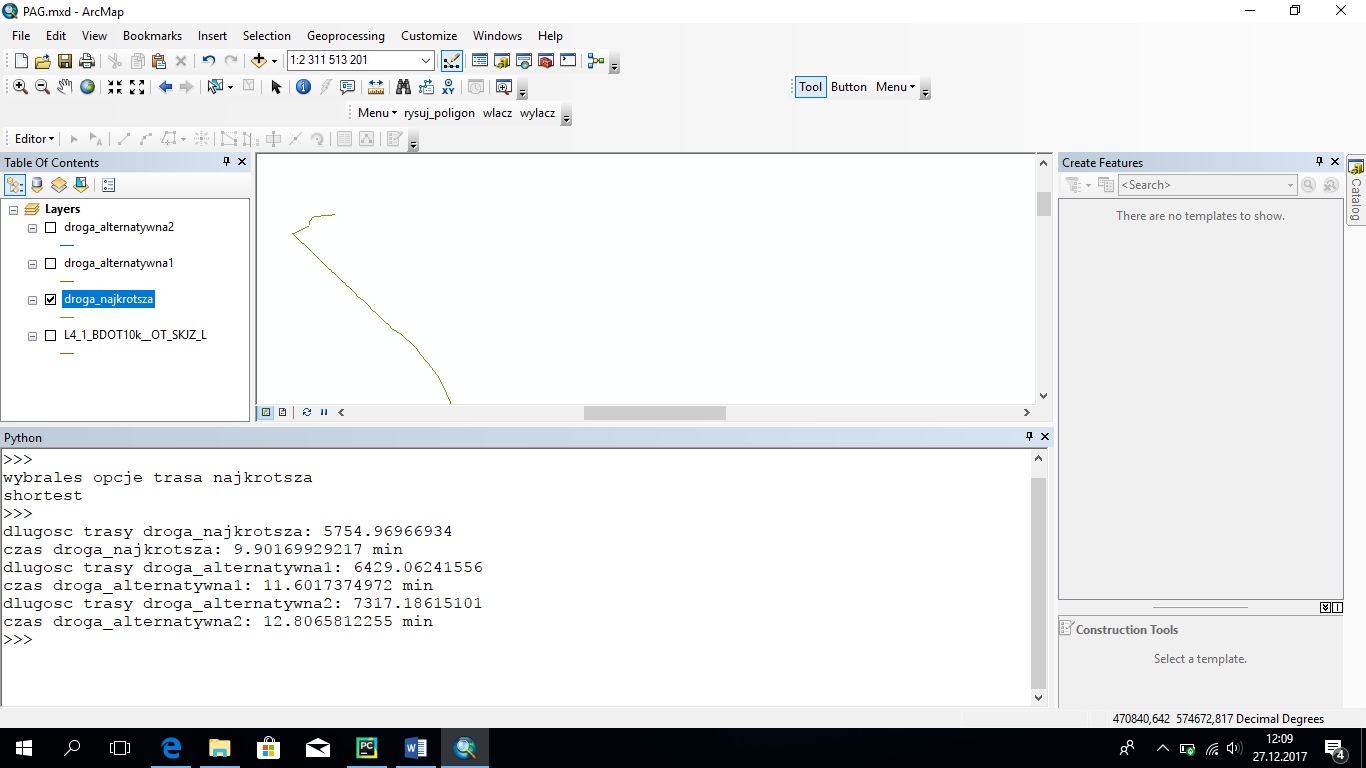
Wyniki.

Wybranie odpowiedniej trasy – najkrótsza:

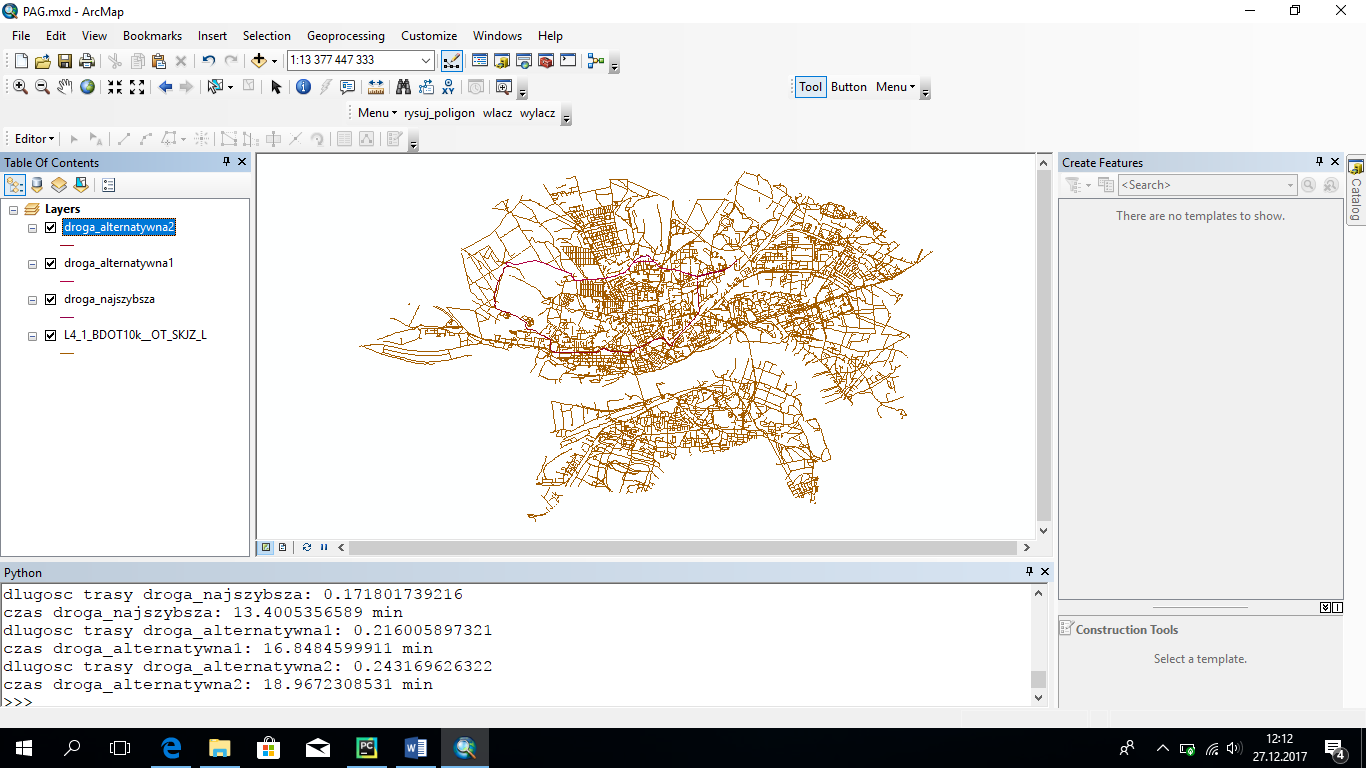


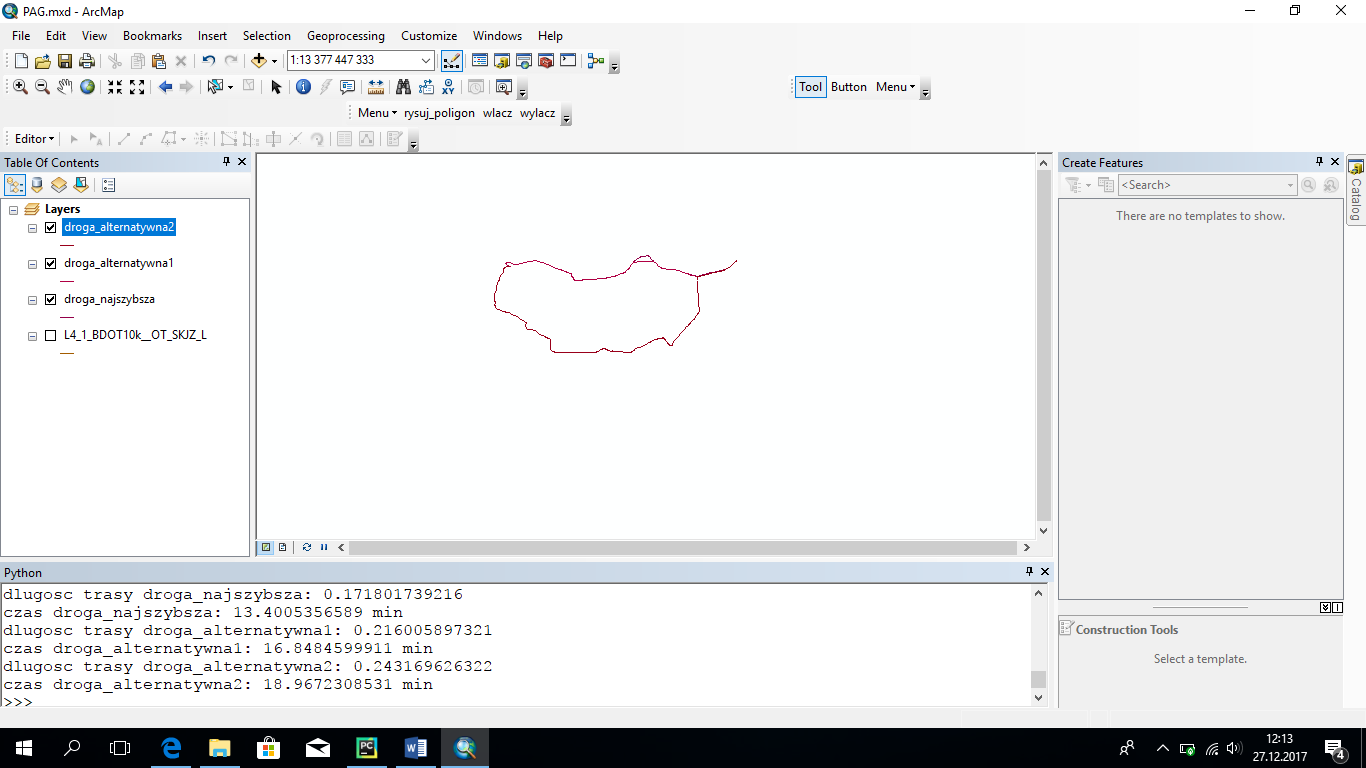
Trasa najkrótsza oraz 2 trasy alternatywne

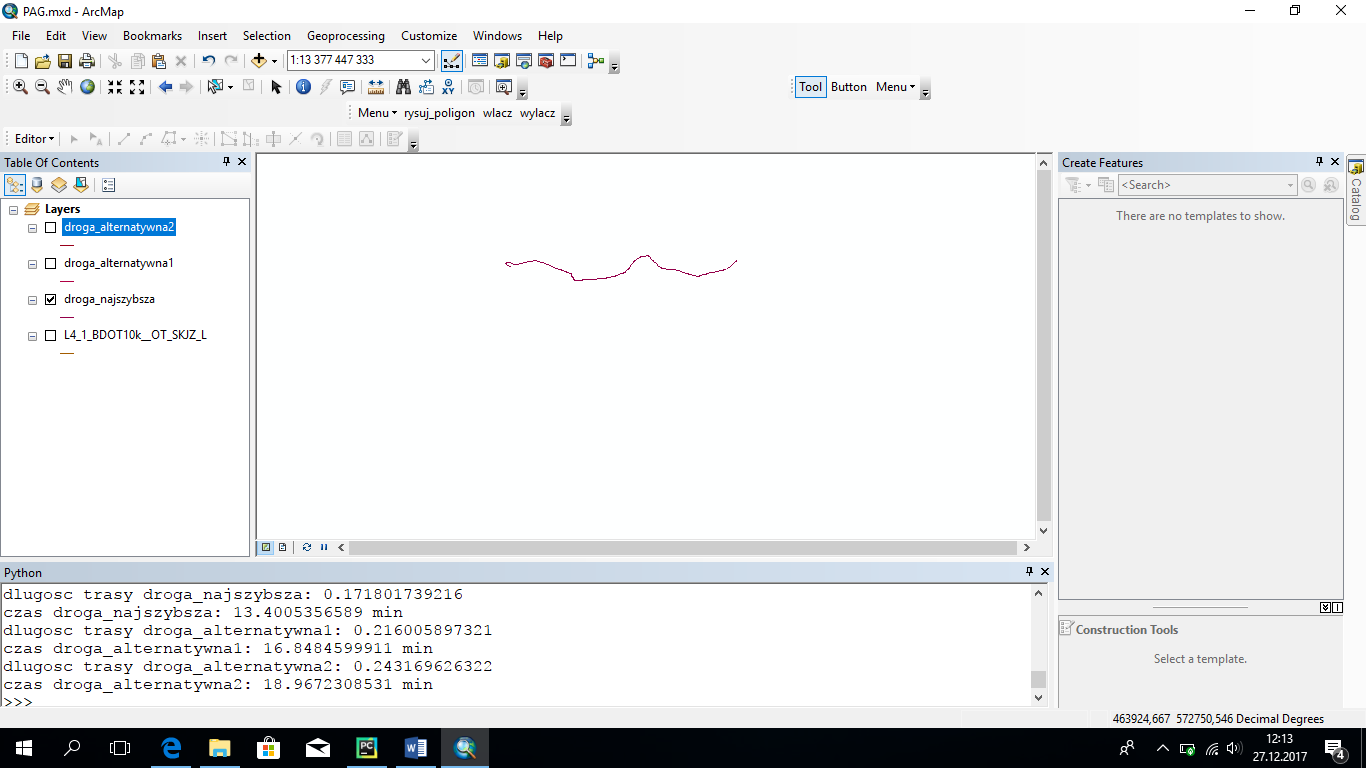


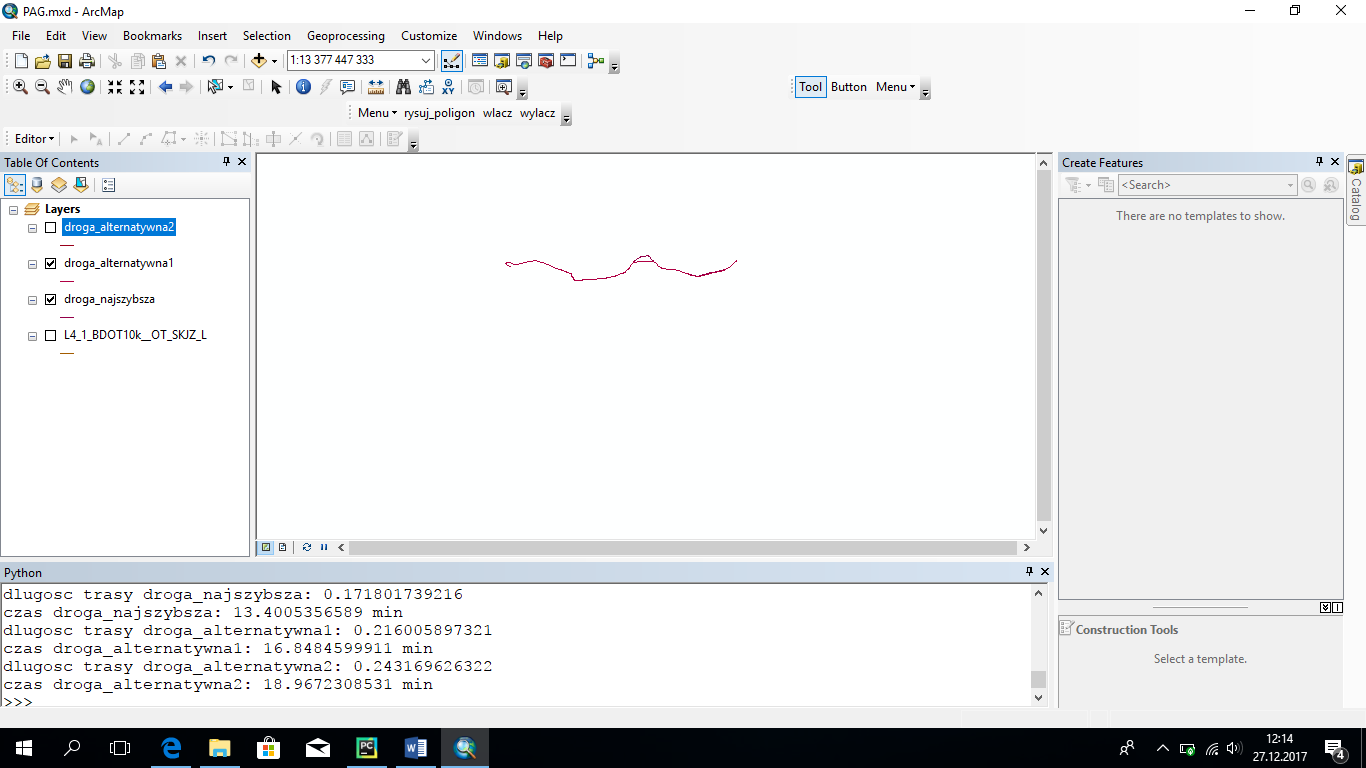


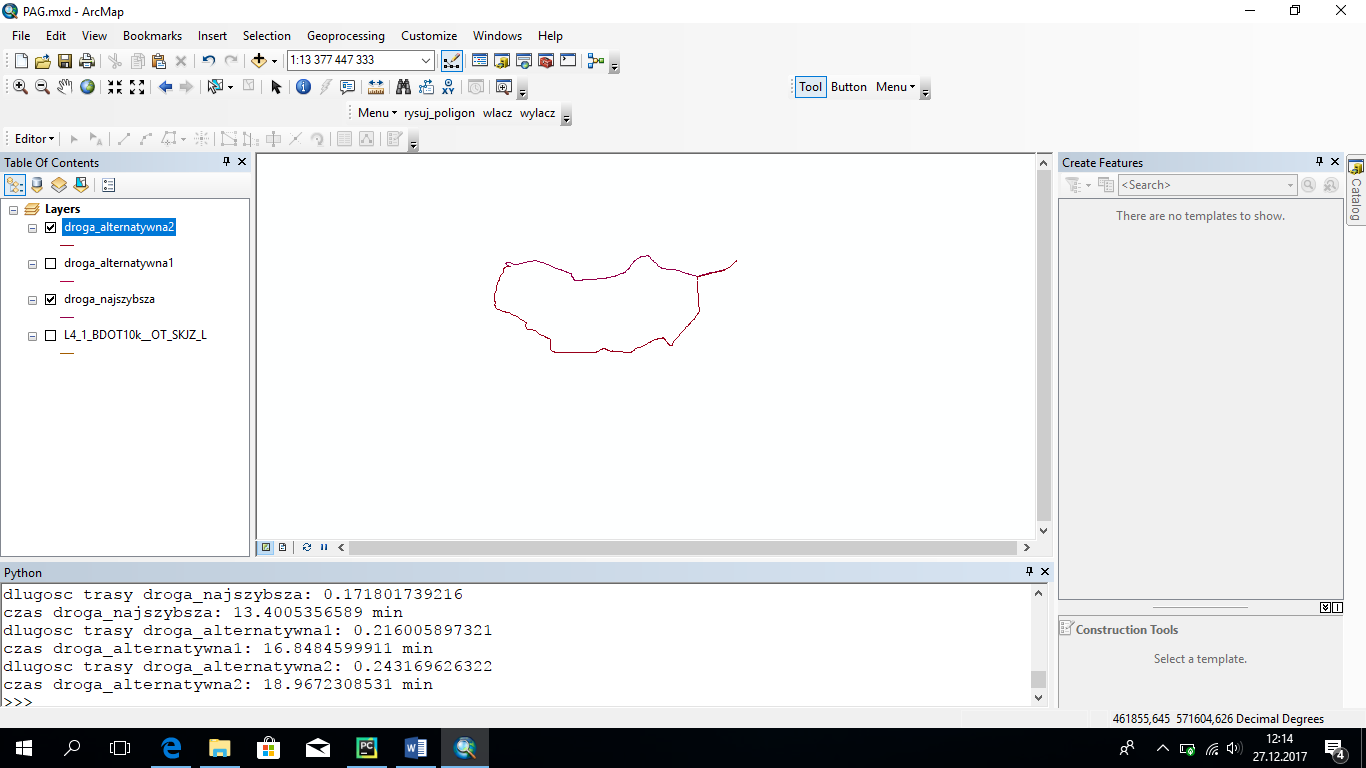
Trasa najszybsza:

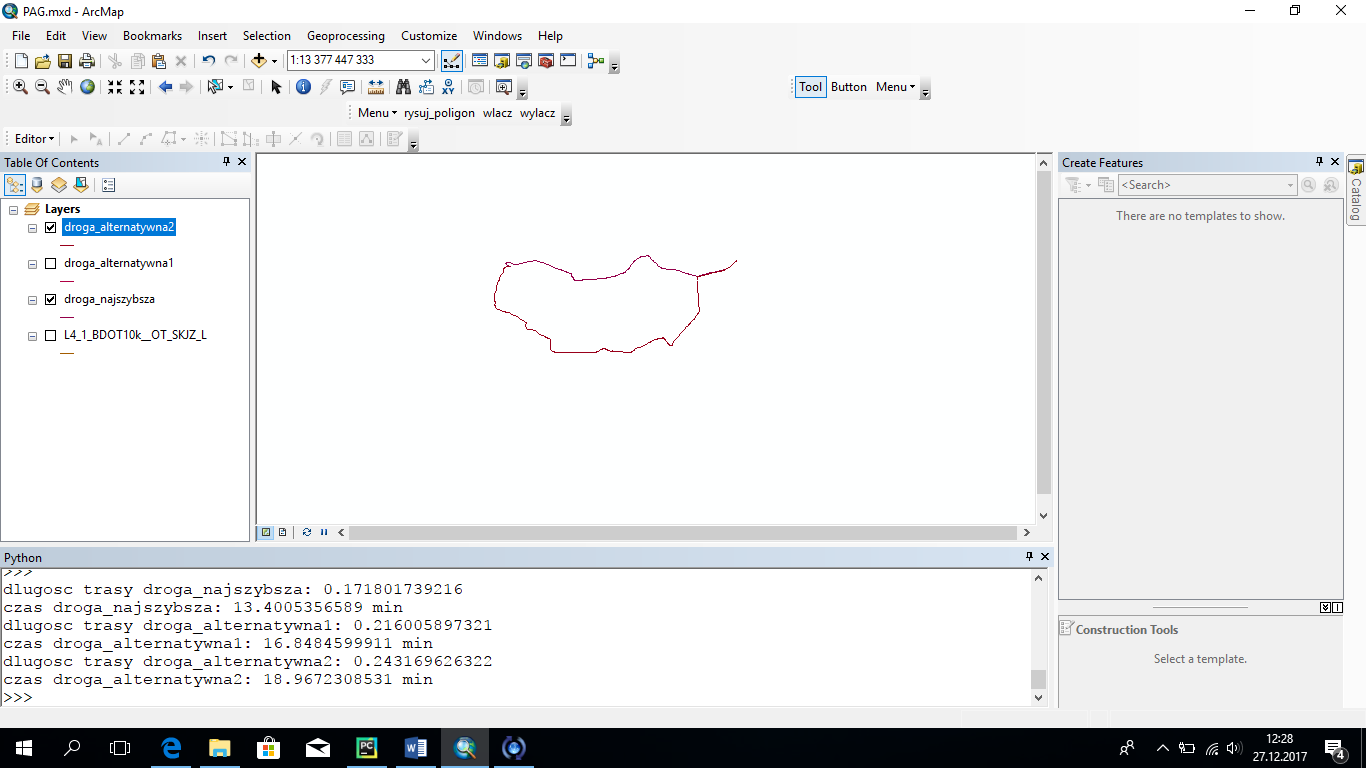












Wnioski.

Zdecydowałem się na zastosowanie algorytmu z modułu netoworkX, ponieważ implementowane przeze mnie algorytmu A\* oraz algorytm Djisktra mogłyby być wykorzystywane jedynie dla małych ilości danych, przy dużych nie jest to możliwe. Rozwiązany przeze mnie problem „alternatywnych tras” polegał na zwiększeniu odpowiednich wartości. Należy przy tym uważać, aby przy następnym wyszukiwaniu ( zmienieniu trasy lub typu ) wprowadzić od nowa dane. Po analizie wyników można zauważyć iż wartość długości nie zawsze oznacza najszybszą trasę. Trasa alternatywna może zajmować mniej czasu w przypadku trasy najkrótszej.