Uruchomienie algorytmów wyznaczania trasy dla sieci drogowej

Adrian Fabisiewicz (328935), Julia Gomulska (328936)

8 grudnia 2024

1 Opis ćwiczenia

Celem ćwiczenia było uruchomienie algorytmów wyznaczania trasy dla sieci drogowej. W ramach ćwiczenia zaimplementowano algorytmy Djikstry oraz A*. Stworzony program odczytuje sieć drogową z wybranego źródła oraz uruchamia wyznaczanie trasy pomiędzy dwoma punktami, uwzględniając kierunkowość dróg. Następnie zwraca trasę najkrótszą oraz najszybszą. Program zintegrowano z oprogramowaniem GIS - ArcGIS Pro, dzięki czemu użytkownik ma możliwość ręcznego wybrania punktów początkowego i końcowego na mapie. Zaimplementowano również rozszerzenie, pozwalające na wyznaczenie zasięgu obszaru, do którego można dojechać z danego punktu w określonym przez użytkownika czasie.

2 Środowisko pracy oraz wykorzystane dane

Ćwiczenie zostało wykonane w języku Python z wykorzystaniem biblioteki ArcPy. Do kontroli wersji korzystano z systemu Git. Danymi wejściowymi były dane BDOT dla Torunia i powiatu toruńskiego, udostępnione przez prowadzącego.

3 Opracowane algorytmy

Algorytm Dijkstry i A* został opracowany na podstawie rozwiązania opublikowanego na stronie redblobgames.com dostepnego pod linkiem: https://www.redblobgames.com/pathfinding/a-star/implementation.html. Skorzystano z istniejącego rozwiązania celem zaoszczędzenia czasu i chęci skupienia się na dostosowaniu kodu do postawionych wamagań oraz integracji z oprogramowaniem GIS. Użyto innych struktur danych niż w rozwiązaniu internetowym: zamiast bezpośredniego działania na lokalizacjach opisanych jako para współrzędnych, skorzystano z obiektów wierzchołków i ich identyfikatorów. Zrezygnowano z implemetacji własnej klasy *PriorityQueue*, zamiast tego skorzystano z kolejki priorytetowej *PriorityQueue* modułu *queue* dostępnego w Pythonie. Dodano weryfikację wejściowych wierzchołków w celu upewnienia się, że istnieją w grafie, obsługę kierunkowości dróg i możliwość wyboru kryterium wyznaczania trasy (w algorytmie A*).

3.1 Algorytm Djikstry

Algorytm służy do wykrywania najkrótszej ścieżki między dwoma wierzchołkami w grafie. Algorytm działa w następujący sposób:

1. Inicjalizacja
   - 1.1 Kolejka priorytetowa *frontier* służy do przechowywania wierzchołków z priorytetami opartymi na kosztach dotarcia do tych nich. Jest inicjalizowana z wierzchołkiem początkowym, a jego priorytet wynosi 0.
   - 1.2 Słowniki *came_from* oraz *cost_so_far* są inicjalizowane jako puste. Przechowują informacje o poprzednikach wierzchołków oraz kosztach dojścia do nich.
2. Przetwarzanie wierzchołków
   - 2.1 Algorytm działa do momentu, gdy kolejka priorytetowa zostanie opróżniona lub gdy zostanie odnaleziona ścieżka do wierzchołka końcowego.
   - 2.2 W każdej iteracji z kolejki pobierany jest wierzchołek o najniższym priorytecie.

3. Przetwarzanie sąsiadów danego wierzchołka

   3.1 Dla każdego sąsiada aktualnie analizowanego wierzchołka sprawdzane są dostępne krawędzie i ich kierunkowość.

   3.2 Jeśli droga jest przejezdna w odpowiednim kierunku, obliczany jest nowy koszt dojścia do węzła sąsiedniego.

4. Aktualizacja wartości

   4.1 Jeśli węzęł - sąsiad nie został jeszcze odwiedzony lub nowo obliczony koszt dotarcia jest mniejszy od kosztu zapisanego w słowniku rejestrowany jest nowy, mniejszy koszt dotarcia.

   4.2 Wierzchołek sąsiedni jest dodawany do kolejki priorytetowej z nowym priorytetem. Priorytet jest równy nowemu kosztowi dotarcia do wierzchołka.

   4.3 Wierzchołek aktualnie analizowany jest zapisywany jako poprzednik wierzchołka sąsiedniego.

5. Zakończenie

   5.1 Po zakończeniu algorytmu, zwracane są dwa słowniki: *came_from* - pozwalający na rekonstrukcję przebiegu ścieżki oraz *cost_so_far* - zawierający skumulowane koszty dotarcia do wierzchołków.

3.2 Algorytm A*

Algorytm A* jest rozszerzeniem algorytmu Dijkstry, który wprowadza heurystykę do szacowania kosztu dotarcia do celu. Dodatkowo umożliwia wyznaczenie kosztu dotarcia do wierzchołka na podstawie długości drogi lub czasu przejazdu. Algorytm działa w następujący sposób:

1. Inicjalizacja

   1.1 Kolejka priorytetowa *frontier* służy do przechowywania wierzchołków z priorytetami opartymi na kosztach dotarcia do tych nich. Jest inicjalizowana z wierzchołkiem początkowym, a jego priorytet wynosi 0.

   1.2 Słowniki *came_from* oraz *cost_so_far* są inicjalizowane jako puste. Przechowują informacje o poprzednikach wierzchołków oraz kosztach dojścia do nich.

2. Przetwarzanie wierzchołków

   2.1 Algorytm działa do momentu, gdy kolejka priorytetowa zostanie opróżniona lub gdy zostanie odnaleziona ścieżka do wierzchołka końcowego.

   2.2 W każdej iteracji z kolejki pobierany jest wierzchołek o najniższym priorytecie.

3. Przetwarzanie sąsiadów danego wierzchołka

   3.1 Dla każdego sąsiada aktualnie analizowanego wierzchołka sprawdzane są dostępne krawędzie i ich kierunkowość.

   3.2 Jeśli droga jest przejezdna w odpowiednim kierunku, obliczany jest nowy koszt dojścia do węzła sąsiedniego. Jest on obliczany w oparciu o długość drogi - w przypadku wyboru opcji *distance* lub czas przejazdu - w przypadku wyboru opcji *time*.

4. Aktualizacja wartości

   4.1 Jeśli węzęł - sąsiad nie został jeszcze odwiedzony lub nowo obliczony koszt dotarcia jest mniejszy od kosztu zapisanego w słowniku rejestrowany jest nowy, mniejszy koszt dotarcia.

   4.2 Wierzchołek sąsiedni jest dodawany do kolejki priorytetowej z nowym priorytetem. Priorytet jest sumą kosztu dotarcia do wierzchołka oraz heurystyki.

   4.3 Wierzchołek aktualnie analizowany jest zapisywany jako poprzednik wierzchołka sąsiedniego.

5. Zakończenie

   5.1 Po zakończeniu algorytmu, zwracane są dwa słowniki: *came_from* - pozwalający na rekonstrukcję przebiegu ścieżki oraz *cost_so_far* - zawierający skumulowane koszty dotarcia do wierzchołków.

3.3 Funkcja heurystyki

Funkcja heurystyki jest wykorzystywana w algorytmie A* do szacowania kosztu dotarcia do celu. Zwraca ona odległość euklidesową lub szacowany czas przejazdu (liczony na podstawie maksymalnej prędkości poruszania się po drogach z uwzględnieniem klas dróg).

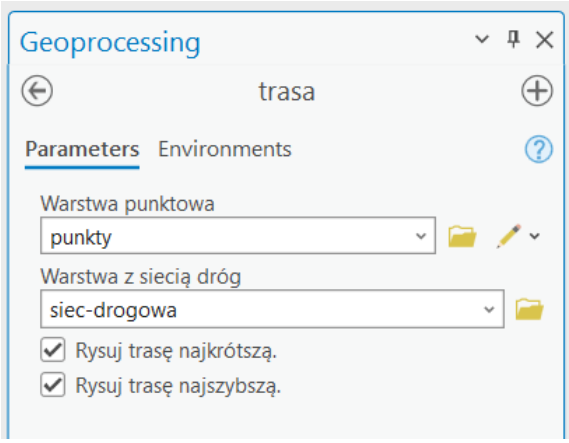
3.4 Algorytm zasięgu

Wyznaczenie zasięgu opiera się na funkcji *create_reachability_map*, która zwraca wierzchołki osiągalne w zadanym czasie. Funkcja ta działa w następujący sposób:

1. Inicjalizacja
    1.1 Kolejka priorytetowa *frontier* służy do przechowywania wierzchołków z priorytetami opartymi na czasie dotarcia do nich. Jest inicjalizowana z wierzchołkiem początkowym, a jego priorytet wynosi 0.
    1.2 Słownik *came_from* jest inicjalizowany jako pusty. Przechowuje informację o poprzedniku każdego węzła.
    1.3 Słownik *time_to_reach* zawiera informacje o czasie dotarcia do każdego z odwiedzonych wierzchołków.
2. Przetwarzanie wierzchołków
    2.1 Algorytm działa do momentu, gdy kolejka priorytetowa zostanie opróżniona.
    2.2 Z kolejki pobierany jest węzeł o najniższym priorytecie.
    2.3 Jeśli czas dotarcia do wierzchołka jest większy od zadanego czasu, wierzchołek jest pomijany.
3. Przetwarzanie sąsiadów danego wierzchołka
    3.1 Dla każdego wierzchołka - sąsiada sprawdzana jest kierunkowość drogi.
    3.2 Jeśli droga jest przejezdna w odpowiednim kierunku, obliczany jest czas dotarcia do sąsiada.
    3.3 Jeśli sąsiad nie ma jeszcze przypisanego czasu dotarcia lub nowo obliczony czas jest mniejszy od zapisanego, czas jest aktualizowany.
    3.4 Wierzchołek sąsiadni jest dodawany do kolejki priorytetowej z nowym priorytetem.
    3.5 W słowniku *came_from* zapisywany jest obecny węzeł jako poprzednik sąsiada.
4. Filtracja wyników
    4.1 Po zakończeniu przetwarzania, zwracane są wierzchołki, do których można dotrzeć czasie krótszym lub równym zadanemu wraz z informacją o ich poprzednikach.

4 Integracja z GIS

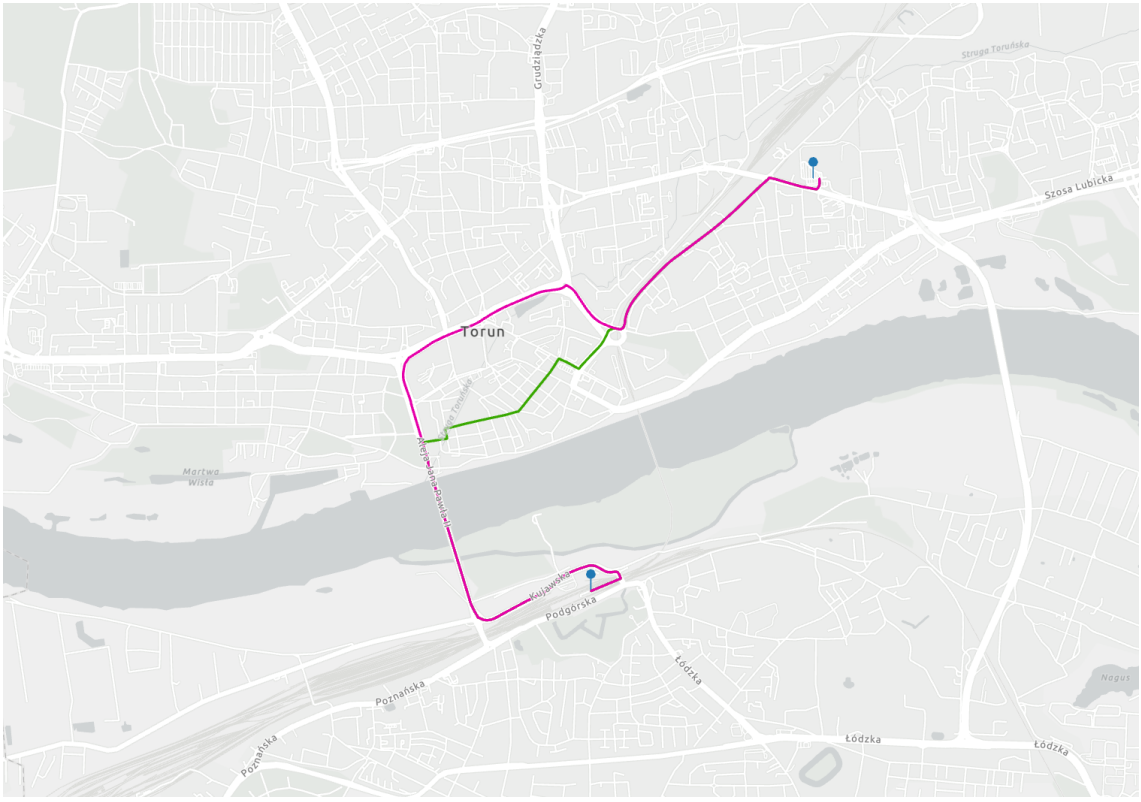
Utworzony program został zintegrowany ze środowiskiem ESRI ArcGIS Pro. Narzędzie w oprogramowaniu pozwala na wyznaczenie trasy pomiędzy dwoma punktami na mapie. Punkty są pobierane z wybranej dwupunktowej warstwy zawierającej punkt początkowy i punkt końcowy, bądź zaznaczane kliknięciem. Należy również wybrać warstwę zawierającą sieć drogową, po której ma zostać wyznaczona trasa. Użytkownik ma możliwość wyboru, czy chce wyznaczyć trasę najkrótszą, czy najszybszą, czy obie z nich. Po uruchomieniu narzędzia, program wyznacza wybrane trasy i dodaje je do mapy. W logach programu zwracane są informacje o czasie najszybszej trasy i długości najkrótszej trasy. Interfejs narzędzia został zaprezentowany na rysunku 1.



Rysunek 1: Interfejs narzędzia do wyznaczenia trasy

5 Przykłady działania algorytmu wyznaczającego trasy

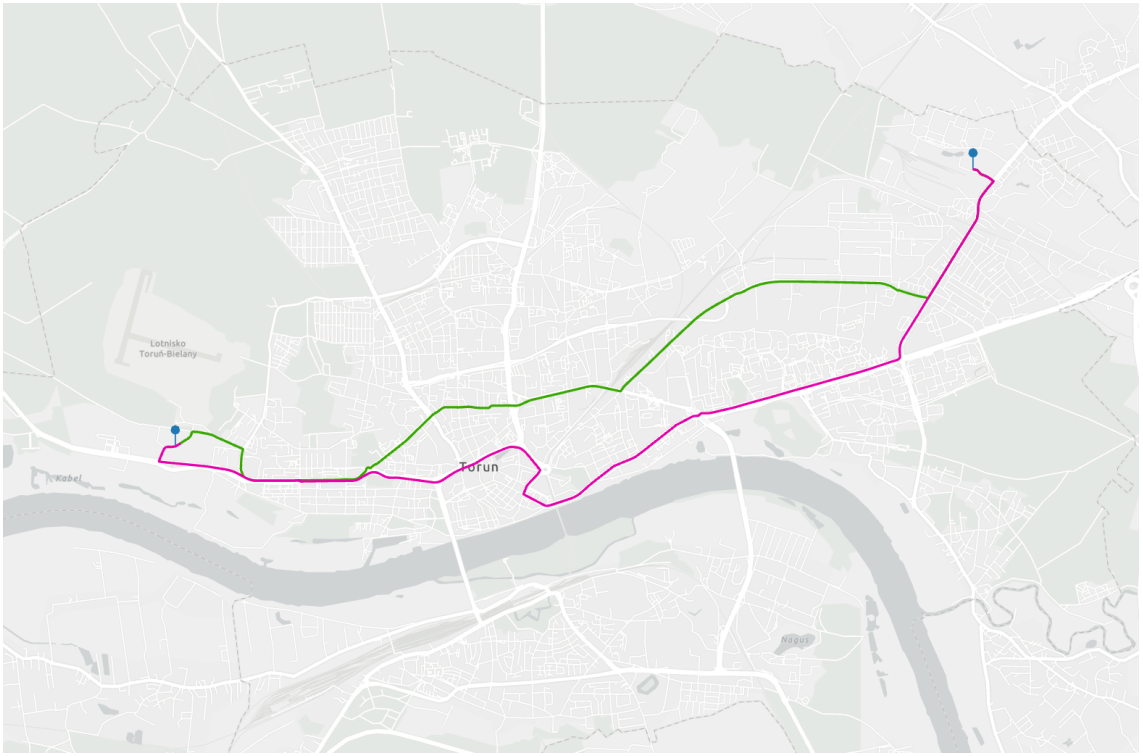
5.1 Przykład 1: Dworzec Główny - Galeria Copernicus



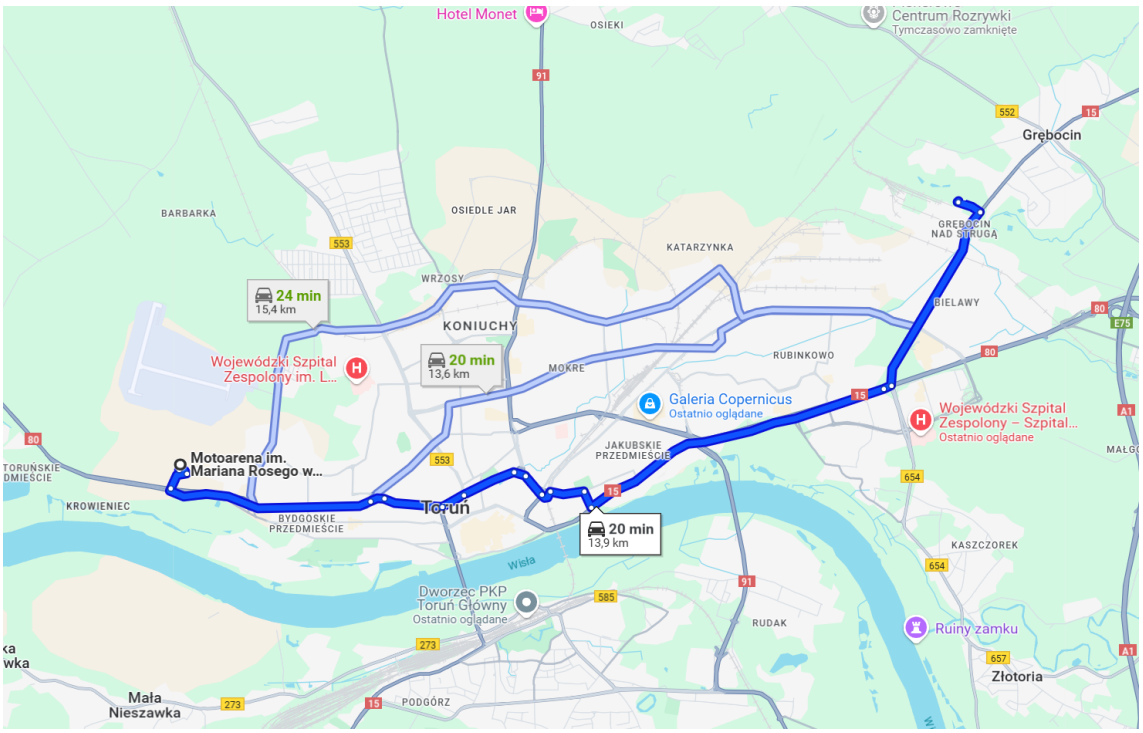
Rysunek 2: Trasy wyznaczone przez stworzone narzędzie (na zielono - trasa najkrótsza, na różowo - najszybsza)

Rysunek 3: Trasy proponowane przez Google Maps

5.2 Przykład 2: Motoarena - ROD Zielone Wzgórze



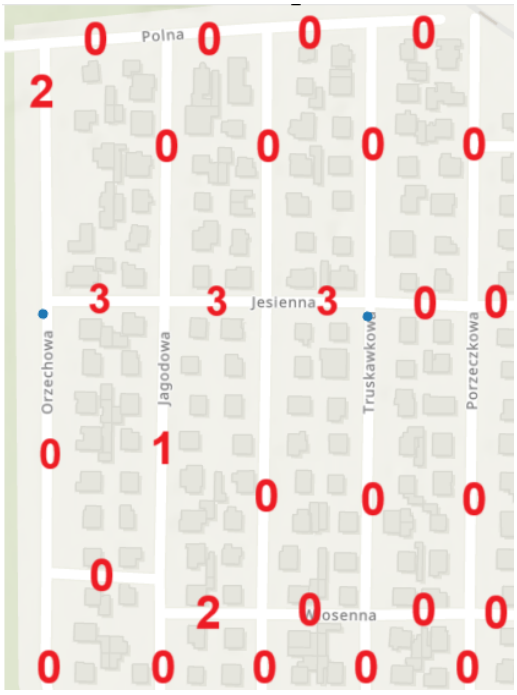
Rysunek 4: Trasy wyznaczone przez stworzone narzędzie (na zielono - trasa najkrótsza, na różowo - najszybsza)



Rysunek 5: Trasy proponowane przez Google Maps

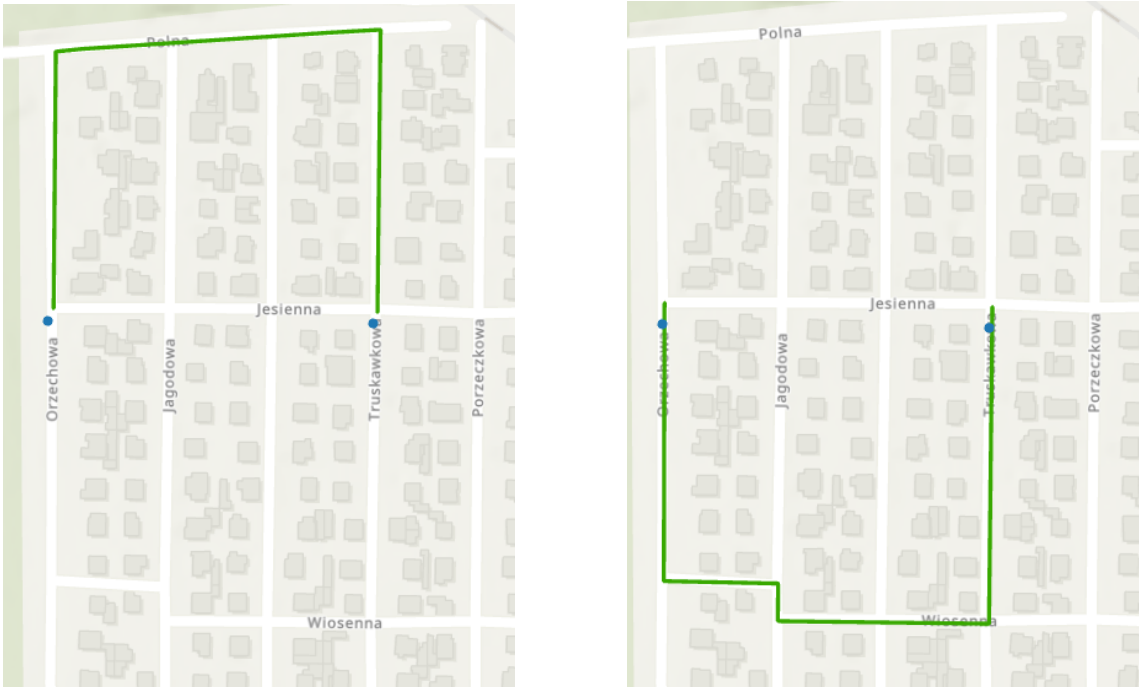
5.3 Przykład 3: Drogi z różną kierunkowością

Do zaprezentowania działania algorytmu w zależności od kierunkowości przypisano wybranym ulicom różne wartości atrybuty kierunkowości. Informacje o kierunkowości danej drogi przedstawiono na poniższym rysunku.



Rysunek 6: Kierunkowość wybranych dróg

Różnicę w przebiegu trasy najkrótszej dla dwóch punktów zlokalizowanych przy ulicy Orzechowej i Truskawkowej przedstawiono poniżej.



(a) Trasa gdy punkt początkowy znajduje się przy ulicy Orzechowej

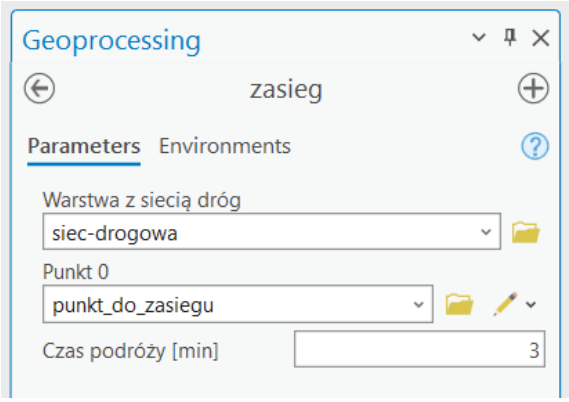
(b) Trasa gdy punkt początkowy znajduje się przy ulicy Truskawkowej

Rysunek 7: Porównanie tras dla puntków przy ulicach Orzechowej i Truskawkowej

6 Zasięg

6.1 Integracja z GIS

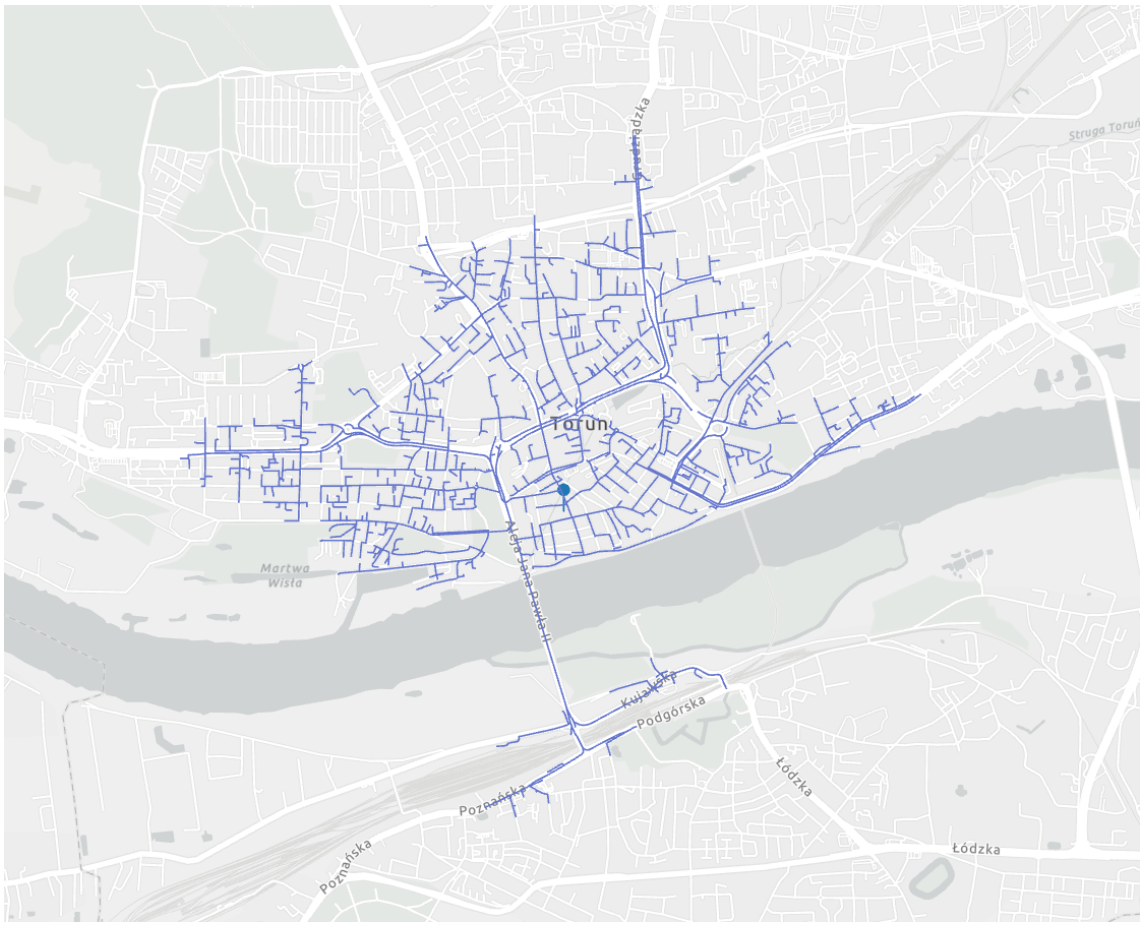
Program do wyznaczania zasięgu również zintegrowano ze środowiskiem ArcGIS Pro. Użytkownik ma możliwość wybrania punktu początkowego na mapie, a następnie określenia czasu, w jakim chce dojechać do punktów z zasięgu. Program zwraca obszar, do którego można dojechać w wybranym czasie. Interfejs narzędzia został zaprezentowany na rysunku 6.



Rysunek 8: Interfejs narzędzia do wyznaczania zasięgu

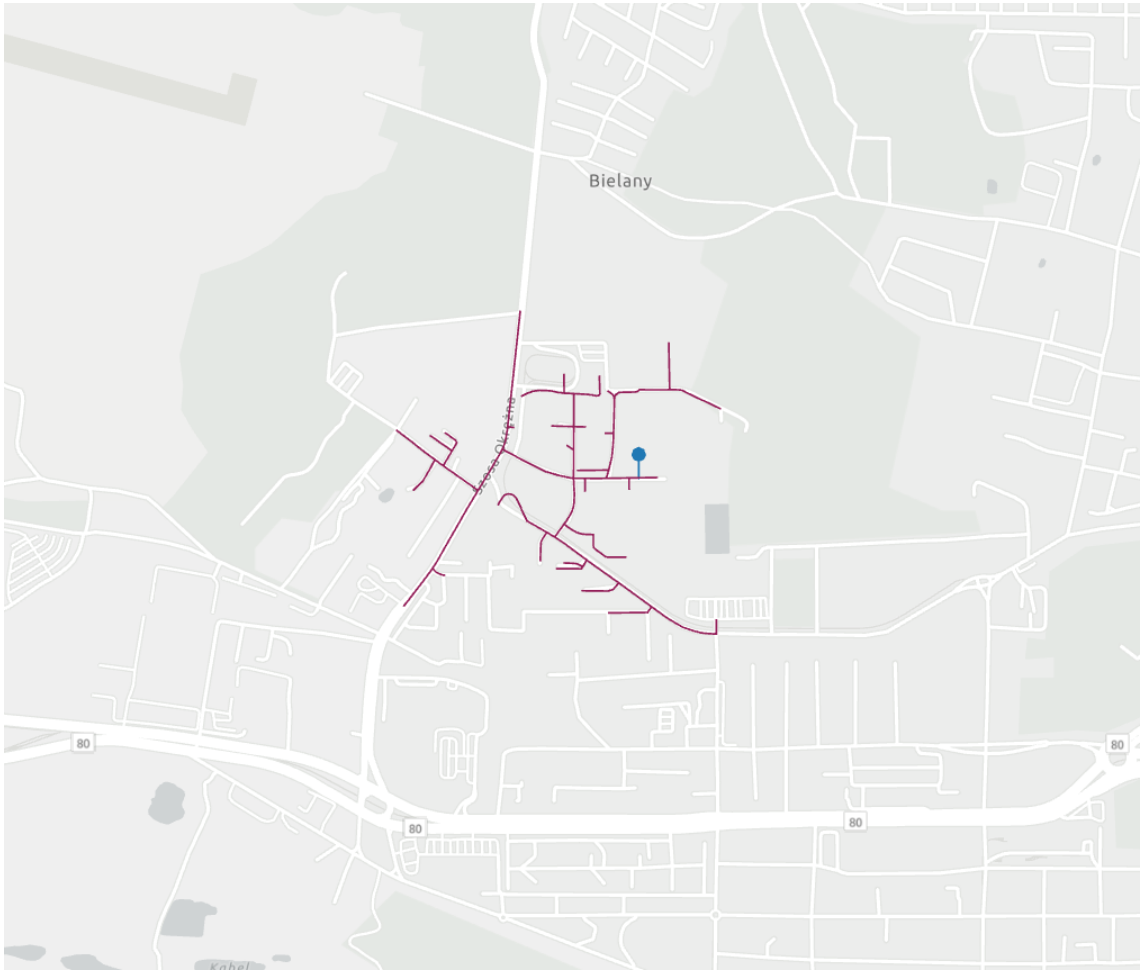
6.2 Przykłady działania

6.2.1 Przykład 1: Rynek Staromiejski



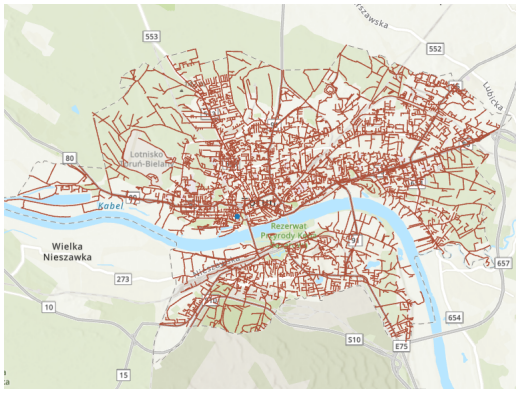
Rysunek 9: Trasy możliwe do pokonania w 2 minuty z punktu startowego na Rynku Staromiejskim

6.2.2 Przykład 2: Biblioteka Uniwersytecka

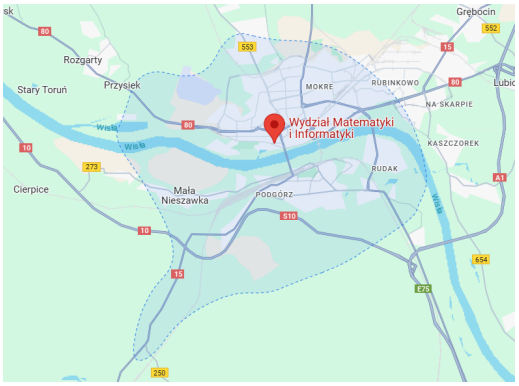


Rysunek 10: Trasy możliwe do pokonania w 1 minutę z punktu startowego przy Bibliotece Uniwesyteckiej

6.2.3 Przykład 3: Wydział Matematyki i Informatyki Uniwersytetu Mikołaja Kopernika



(a) Trasy możliwe do pokonania w 15 minut z punktu startowego przy Wydziale Matematyki i Informatyki Uniwersytetu Mikołaja Kopernika według stworzonego narzędzia



(b) Trasy możliwe do pokonania w 15 minut z punktu startowego przy Wydziale Matematyki i Informatyki Uniwersytetu Mikołaja Kopernika według Google Maps

Rysunek 11: Trasy możliwe do pokonania w 15 minut z punktu startowego przy Wydziale Matematyki i Informatyki Uniwersytetu Mikołaja Kopernika

7 Wnioski

Stworzony algorytm dobrze wyznacza trasę najkrótszą i najszybszą pomiędzy dwoma punktami. W przypadku zasięgu, algorytm zwraca większy obszar niż Google Maps. Rozbieżności między wynikami uzyskanymi przez program a Google Maps mogą wynikac z faktu, że zakładamy poruszanie się z największą dopuszczalną prędkością, a w rzeczywistości poruszamy się wolniej. Ponadto program nie uwzględnia ograniczeń w ruchu takich jak: zakazy ruchu, zakazy skrętu, rzeczywista kierunkowość dróg (kierunkowość dróg niepozwalająca na przejazd została przez nas nadana jedynie wybranym drogom). Nie mamy również informacji o aktualnym natężeniu ruchu, co znacząco wpływa na czas przejazdu.
Podczas pracy napotkano problemy związane z zbyt długim działaniem programu. Bardzo ważne jest stosowanie struktur danych, które zapewnią odpowiednią złożoność czasową.

8 Referencje