Kreator długodystansowych tras rowerowych

Krzysztof Fijałkowski, Tomasz Owienko, Jakub Woźniak

28.05.2025

Cel projektu

Celem projektu była implementacja aplikacji pozwalającej na planowanie wielodniowych tras rowerowych.

Wymagania funkcjonalne

- Interaktywne wyznaczanie tras rowerowych przebiegającej przez zadane punkty na mapie
- Uwzględnienie typu roweru oraz rodzajów dróg i ich nawierzchni w wyznaczaniu przebiegu trasy
- Sugerowanie miejsc wartych odwiedzenia oraz noclegów na podstawie przebiegu trasy
- Wspieranie równomiernego rozkładu się w poszczególnych dniach
- Wizualizacja przebiegu trasy
- Eksport wygenerowanych tras do formatu GPX

Architektura rozwiązania

Aplikacja została zaimplementowana z wykorzystaniem biblioteki Streamlit dla języka Python. Jako bazę danych wykorzystano PostgreSQL w wersji 17 z rozszerzeniami PostGIS i pgrouting. Do wyszukiwania punktów zwiedzania oraz noclegów skorzystano z Overpass API.

Model danych

Do reprezentacji topologii sieci wykorzystano dwie tabele: `ways` oraz `ways_vertices_pgr`

Tabela `ways` przechowuje krawędzie grafu oraz powiązane z nimi informacje potrzebne do wyznaczenia ścieżki przez algorytm A*, między innymi:

- `gid` - identyfikator krawędzi grafu
- `osm_id` - identyfikator drogi z której powstała dana krawędź grafu w OpenStreetMap
- `length_m` - długość odcinka w metrach
- `source` / `target` - identyfikatory wierzchołków na końcach krawędzi
- `onr_way` - wskazanie, czy krawędź jest możliwa do przebycia tylko w jednym kierunku
- `the_geom` - geometria danego odcinka (`LineString`)
- `road_type` - rodzaj drogi
- `grid_lat` / `grid_lon` - współrzędne geograficzne środka geometrii

Założone zostały następujące indeksy na następujące kolumny / wyrażenia:

- `gid, road_type` - b-drzewo
- `grid_lat` - b-drzewo
- `grid_lon` - b-drzewo
- `source` - b-drzewo
- `target` - b-drzewo
- `the_geom` - `gist`
- `the_geom::geography` - `gist`

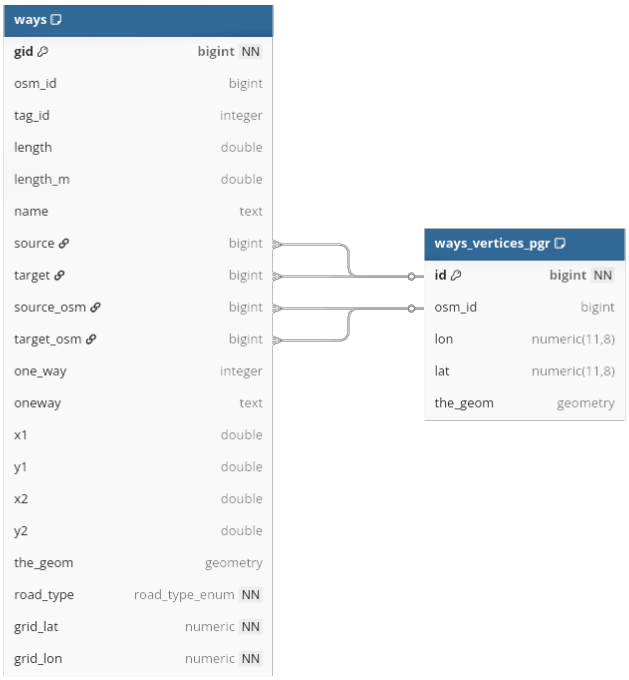

Figure 1: Model danych

Tabela `ways_vertices_pgr` przechowuje wierzchołki grafu wraz z ich współrzędnymi (`lat`, `lon`) i geometrią `the_geom`. Indeksy typu `gist` zostały założone na `the_geom` oraz `the_geom::geography`.

Model danych rozróżnia sześć typów dróg wynikających z tagów w OpenStreetMap:

- `roads_primary` - drogi krajowe
- `roads_secondary` - drogi wojewódzkie
- `roads_paved` - pozostałe drogi, utwardzone
- `roads_unpaved` - pozostałe drogi, nieutwardzone
- `cycleways` - drogi pozwalające na ruch rowerowy
- `roads_unknown_surface` - pozostałe

Mapowanie tagów na typy dróg omówione zostanie w następnej sekcji.

Ładowanie danych

Ładowanie danych odbywa się w kilku etapach:

1. Pobranie danych z bazy OpenStreetMap

- Mapy poszczególnych województw pobierane są z serwisu `geofabrik.de` w formacie PBD. Następnie każdy plik konwertowany jest do formatu XML (`.osm`) za pomocą narzędzia `osmconvert` i usuwane są z niego niektóre tagi nieprzydatne z punktu widzenia projektu - w szczególności tagi `author` i `version`.

2. Budowa topologii sieci

- Dla każdego pliku zawierającego mapę danego województwa wykonywane są następujące akcje:
  - Za pomocą narzędzia `osmfilter` usuwane są tagi nieprzydatne z punktu widzenia projektu, w wyniku czego w pliku pozostają jedynie dane dotyczące dróg - pozwala to zmniejszyć rozmiar pliku o około 65%.
    - Drogi warte rozpatrzenia wyznaczane są na podstawie wartości tagów `cycleway`, `tracktype` oraz `highway`

  - Plik ładowany jest do bazy danych i jednocześnie budowana jest topologia sieci, za co odpowiada narzędzie `osm2pgrouting`. Tym samym, drogi obecne w oryginalnych danych z OpenStreetMap mogą być dzielone na mniejsze odcinki lub scalane, przez co niektóre drogi z OSM nie posiadają odwzorowania w bazie, a innym odpowiada kilka krawędzi grafu. Na tym etapie pomijane jest tworzenie indeksów. Dane umieszczane są w tabelach `ways` i `ways_vertices_pgr`. Należy zaznaczyć, że struktura tabeli zdefiniowana jest przez narzędzie `osm2pgrouting`, w związku z czym niektóre wymagane kolumny muszą zostać dodane później.
  - Ten sam plik jest dodatkowo filtrowany narzędziem `osmfilter` aby wyodrębnić z niego poszczególne typy dróg, a następnie ładowany jest do tabeli `planet_osm_line` za pomocą narzędzia `osm2pgsql`
  - Na podstawie identyfikatorów obiektów w OSM obecnych w obu tabelach uzupełniane są informacje o typie drogi w tabeli `ways`
    * Typ drogi wyznaczany jest na podstawie wartości tagów `highway`, `cycleway` i `surface` (lub faktu ich braku)

3. Analiza danych i optymalizacja modelu

- Graf jest analizowany pod kątem obecności izolowanych podgrafów, następnie wszystkie izolowane podgrafy poza największym są usuwane (przy ładowaniu mapy całej Polski wiąże się to z usunięciem około 150.000 z 11.000.000 krawędzi)
- Dla każdej krawędzi grafu wyznaczany jest jej środek dla szybszego wyszukiwania (wyszukiwanie krawędzi dla algorytmu A* omówiono w kolejnej sekcji)
- Tworzone są indeksy
- Aktualizowane są statystyki optymalizatora

Wyznaczanie tras

Trasa przechodząca przez zadane punkty w określonej kolejności wyznaczana jest jako złączenie tras między kolejnymi parami punktów. W aplikacji za wyznaczanie tras pomiędzy parami punktów odpowiada dwukierunkowy algorytm A*, którego sposób wykorzystania opisany został w kolejnych sekcjach. Aby osiągnąć lepszą wydajność, trasy między wszystkimi rozpatrywanymi parami punktów wyznaczane są przez równoległe zapytania do bazy danych, po czym wyniki są łączone i prezentowane użytkownikowi.

Algorytm A* Wyznaczanie trasy oparte jest o dwukierunkowy algorytm A* oraz jego implementację w rozszerzeniu `pgrouting` dla PostgreSQL. Pozwala on wyznaczyć (w przybliżeniu) najtańszą ścieżkę między dwoma punktami. Zastosowanie wariantu dwukierunkowego pozwala poprawić wydajność w przypadku długich tras o ok. $15 - 25\%$. Uruchomienie algorytmu wymaga podania:

- Treści zapytania wydobywającego krawędzie grafu z bazy danych
  - Zapytanie powinno zwrócić m.in. współrzędne początków i końców krawędzi oraz koszt ich pokonania w obie strony (ujemny koszt oznacza możliwość przebycia krawędzi tylko w jednym kierunku)
- Identyfikatora wierzchołka startowego
- Identyfikatora wierzchołka końcowego
- Jednej z predefiniowanych funkcji heurystycznych

W aplikacji przyjęto funkcję heurystyczną będącą odległością w linii prostej między dwoma punktami: $h(P) = \sqrt{lat(P)^2 + lon(P)^2}$.

Pobranie danych Zapytanie pobierające dane dla algorytmu A* zwraca wszystkie krawędzie spełniające łącznie podane kryteria:

- Ich środki znajdują się w ramce mbb zawierającej punkt początkowy i startowy poszerzonej o pewien margines
- Odległość między ich środkami a odcinkiem od punktu startowego do końcowego mieści się z marginesie

Margines wyliczany jest w oparciu o odległość między punktem startowym i końcowym wg następującego wzoru (współczynniki dobrane eksperymentalnie):

$$m = min(max(4.3 - \frac{4}{1 + e^{-3.5d+1}}, 0.5d), 3d)$$

gdzie m to margines, a d to odległość między punktem startowym i końcowym w stopniach. W ten sposób dla krótszych tras margines jest szerszy (wyznaczenie stosunkowo prostej drogi na krótkim dystansie jest trudniejsze niż na długim) i stopniowo maleje wraz z długością trasy. Jednocześnie wprowadzono minimalny i maksymalne wartości marginesu w odniesieniu do d $(0.5d, 3d)$ dla poprawy stabilności algorytmu.

Odległość między środkami krawędzi grafu a odcinkiem łączącym początek i koniec wytyczanej ścieżki *nie* jest obliczana w oparciu o predykaty przestrzenne PostGIS, gdyż takie rozwiązanie nie zapewniało oczekiwanej wydajności - w przypadku wyznaczania trasy długości ok. 200km pobieranie krawędzi grafu trwało blisko dwukrotnie dłużej, niż faktyczne działanie algorytmu A*. Zamiast tego odległości obliczane są w sposób bezpośredni z wykorzystaniem odpowiednio przekształconego równania odległości punktu od prostej przebiegającej przez dwa punktu, gdzie większość współczynników równania obliczana jest przez aplikację i podawana jako parametry zapytani. Wykorzystywane są współrzędne geograficzne środków punktów zapisane uprzednio jako typ `numeric` w PostgreSQL, po pozwala ograniczyć narzut na konwersję typów. Takie podejście pozwala ograniczyć liczbę zwracanych krawędzi przy pomijalnie niskim koszcie obliczenia predykatów (pod warunkiem założenia indeksu typu B-drzewo na kolumny z współrzędnymi geograficznymi środków krawędzi grafu).

Obliczanie kosztów krawędzi Ponieważ na koszta krawędzi wpływa wybrany typ roweru, muszą być one obliczone w trakcie zapytania. Koszt każdej krawędzi obliczana jest jako iloczyn jej długości i współczynnika kary zależnego od wybranego typu roweru i rodzaju drogi. Przykładowo, dla roweru szosowego współczynnik będzie wynosił 1 dla dróg utwardzonych niebędących drogami krajowymi i wojewódzkimi, ale jego wartość dla dróg nieutwardzonych wyniesie 3, w związku z czym algorytm będzie je wybierał tylko w zupełnej ostateczności. Współczynniki dobierane były eksperymentalnie dla każdego typu roweru wg następujących preferencji:

- Rower szosowy - silnie preferowane drogi utwardzone, niewielka kara za wybór dróg o dużym natężeniu ruchu
- Rower gravelowy / crossowy - koszt dróg nieutwardzonych niewiele większy niż utwardzonych, stosunkowo duża kara za wybór ruchliwych dróg
- Rower trekkingowy - tak samo jak w przypadku gravelowego, ale duża kara za wybór drogi o wysokim natężeniu ruchu
- Rower górski - koszt dróg nieutwardzonych niższy niż utwardzonych, duża kara za wybór drogi o wysokim natężeniu ruchu
- Rower elektryczny - wagi takie same, jak dla roweru trekkingowego (przyjęto założenie, że mają taką samą zdolność do pokonywania poszczególnych rodzajów nawierzchni, za to różnią się osiąganą prędkością)

Koszta przejazdu drogi w obie strony są równe dla dróg dwukierunkowych, a przypadku dróg jednokierunkowych koszt przejazdu w przeciwną stronę przemnażany jest przez -1 (droga nieprzejezdna).

Końcowe przetwarzanie Funkcja `pgr_bdastar` zwraca kolejne krawędzi wchodzące w skład najkrótszej ścieżki (lub nie zwraca nic jeśli nie udało się znaleźć ścieżki). W ramach zapytania obliczane i przekazywane do aplikacji są następujące dane:

- Złączenie geometrii wszystkich krawędzi grafu w jedną ścieżkę zapisane w formacie *GeoJSON*
- Łączna długość trasy
- Łączny dystans pokonywany drogami każdego z typów

Szacowanie czas przejazdu

Poza podaniem typu roweru użytkownik ma możliwość podania planowanej długości wycieczki (w dniach), planowanego dystansu pokonywanego dziennie i oceny swojej sprawności fizycznej w pięciostopniowej skali.

Te parametry nie wpływają na działanie algorytmu, ale pozwalaja użytkownikowi uzyskać lepszy wgląd w zaplanowaną trasę i nanieść ewentualne poprawki.

Wybór typu roweru oraz ocena zdolności fizycznej wpływają na oszacowanie średniej prędkości możliwej do osiągnięcia, a co za tym idzie, szacunkowy czas potrzebny do pokonania trasy. Dla każdego typu roweru na podstawie wiedzy domenowej dobrano prędkość możliwą do rozwinięcia w zależności od poziomu sprawności fizycznej. Dodatkowo, dla każdego typu roweru i każdego rodzaju drogi ustalono współczynnik skalujący możliwą do osiągnięcia prędkość nie większy niż 1. Przykładowo, w przypadku roweru szosowego współczynnik będzie znacznie niższy dla dróg nieutwardzonych niż dla dróg utwardzonych, a w przypadku rowerów górskich różnica będzie niewielka.

Czas przejazdu danego odcinka jest zatem obliczany jako:

$$t(d, biketype, fitness, roadtype) = \frac{d}{v_b(biketype, fitness) \cdot ratio(biketype, roadtype)}$$

gdzie:

- t - czas
- d - długość odcinka
- v_b - prędkość osiągana w korzystnych warunkach
- $ratio$ - współczynnik skalujący prędkość
- $biketype$ - typ roweru
- $roadtype$ - typ drogi
- $fitness$ - poziom sprawności fizycznej.

Ponieważ zapytanie do bazy danych zwraca dystans pokonywany drogami poszczególnych typów, możliwe jest obliczenie czasu potrzebnego na przejazd między każdą parą punktów na trasie. Tym samym, możliwe jest także obliczenie dystansu pokonywanego każdego dnia i zasugerowanie użytkownikowi modyfikacji trasy.

Dobór punktów zwiedzania oraz noclegów

Punkty są dobierane z użyciem overpassAPI do którego są wysyłane zapytanie a konkretne typy punktów takich jak: muzea, zamki, ruiny itp. Analogicznie wyszukiwane są noclegi które są wyszukiwane w stałej odległości od punktów które są N-tym kilometrem trasy, gdzie N to zadany dzienny dystans.

Interfejs użytkownika

Interfejs użytkownika został zbudowany z wykorzystaniem biblioteki Streamlit, która pozwala na szybkie tworzenie interaktywnych aplikacji webowych w Pythonie, bez potrzeby angażowania złożonych frameworków frontendowych. Do obsługi mapy wykorzystano bibliotekę folium wraz z komponentem streamlit-folium, który umożliwia integrację map Leaflet z aplikacją. Dodatkowo, za pomocą streamlit-extras, możliwe było stylowanie elementów interfejsu, takich jak przyciski, kontenery i kolumny, co pozwoliło na jego lekkie dostosowanie wizualne.

Rozwiązanie to, mimo pewnych ograniczeń w zakresie płynności interakcji i estetyki, pozwalało na bardzo sprawne i szybkie budowanie funkcjonalnego interfejsu. Konfiguracja UI była prosta i intuicyjna, co umożliwiło skupienie się przede wszystkim na logice aplikacji. Choć nie dorównuje elastycznością nowoczesnym frontendowym rozwiązaniom takim jak React z Bootstrapem czy Material UI, to w tym przypadku prostota była zdecydowaną zaletą.

Ograniczenia

Głównym ograniczeniem aplikacji jest wydajność - po załadowaniu całej Polski w bazie danych znajduje się około 11 milionów krawędzi, co nawet przy zredukowaniu ilości danych przekazywanych do algorytmu A* istotnie wpływa na szybkość działania. Kosztowne jest nie tylko wyznaczenie trasy, ale i pobranie danych - przy małych dystansach optymalizator decyduje się na skorzystanie z indeksu bitmapowego i na odpowiednio

szybkiej maszynie jest w stanie pobrać dane w czasie poniżej jednej sekundy, ale przy dużych dystansach często wybiera sekwencyjny przegląd tabeli, który może trwać nawet kilka sekund.

Możliwości rozwoju

- Rozdzielenie pojęcia *typ drogi* na klasę drogi (jak istotna) oraz jej nawierzchnię
- Wyświetlanie nawierzchni drogi na poszczególnych odcinkach na mapie
- Poprawa jakości interakcji z mapą w interfejsie użytkownika