

ChadGuide

Planowanie podróży po Europie

Paweł Kauf, Filip Rabięga

4 lutego 2026

Problem: Planowanie podróży wielomiastowej wymaga sprawdzenia wykładniczej liczby kombinacji kolejności miast. Ręczne porównywanie jest niepraktyczne.

Cel: Znalezienie optymalnych (najtańszych/najkrótszych) tras lotniczych łączących wiele miast europejskich.

ChadGuide jest ChadRozwiązaniem.

- ▶ Oprócz optymalizacji wykonuje jeszcze walidacje lotów
- ▶ Wszystko dzieje się przez wygodną oraz przejrzystą aplikację webową

Użyte technologie

- ▶ Python: poetry, pandas, streamlit, pandera, fastapi, uvicorn, pydantic, sse-starlette, playwright, pytest
- ▶ JavaScript: vanilla js, tailwind css, leaflet.js
- ▶ SQLite
- ▶ nginx
- ▶ API: Duffel, Geoapify (httpx async)

Demo

- ▶ Nasz projekt można obejrzeć na <https://github.com/fiadra/chadguide>.
- ▶ Działającą publiczną instancją jest <https://www.chadguide.site>.

Zbieranie danych o lotach

- ▶ Zbieranie danych o lotach do bazy jest realizowane przez paczkę `flight scanner`, która łączy się z DuffelAPI (148 lotnisk, 89k lotów)
- ▶ Dane w naszej bazie zawsze będą trochę do tyłu względem prawdziwych, ponieważ koszty biletów oraz ich dostępność ciągle się zmieniają. Dane dla tygodnia 01.07.2026–07.07.2026 są ekstrapolowane na tydzień 13.07.2026–19.07.2026.

Zbieranie danych o atrakcjach

- ▶ Zbieranie danych o atrakcjach dzieje się asynchronicznie podczas działania algorytmu. Dane są zaczytywane z bazy statycznej, gdzie mamy ich dane oraz zdjęcia.
- ▶ Zdjęcia atrakcji zostały zebrane metodami webscrapingu.

Algorytm

- ▶ Sercem projektu jest zmodyfikowany algorytm Dijkstry, który wyznacza drogi Pareto-optymalne względem ceny oraz czasu trwania lotu. Jednocześnie radzi sobie on z grafami, które są zmienne w czasie.
- ▶ Droga p *dominuje* drogę q , gdy istnieje kryterium, w którym p jest lepsze od q oraz nie istnieje kryterium, w którym p jest gorsze niż q .
- ▶ Droga jest *Pareto optymalna*, gdy nie jest dominowana przez żadną inną drogę.

Algorytm

- ▶ Interesują nas tylko te drogi, które są Pareto optymalne, oraz które wychodzą i wracają do wyznaczonego miasta, przelatując przy tym przez wszystkie miasta, które użytkownik chce odwiedzić.
- ▶ Inne drogi są automatycznie odrzucane.
- ▶ Przed uruchomieniem algorytmu wykonujemy prosty *pruning*, tzn. pozbywamy się tych połączeń w grafie, które prawie na pewno nie będą częścią optymalnego rozwiązania.

Algorytm

Input: a timetable graph and a query

Output: a set of **Pareto**-optimal labels at the terminal

```
1 foreach node  $v$  do
2    $\text{list}\langle\text{Label}\rangle \text{labelListAt}(v) := \emptyset;$ 
3  $\text{PriorityQueue } pq := \emptyset;$ 
4 foreach node  $v$  in start interval do
5    $\text{Label } \text{startLabel} := \text{createStartLabel}(v);$ 
6    $pq.\text{insert}(\text{startLabel});$ 
7 while  $\neg pq.\text{isEmpty}()$  do
8    $\text{Label } \text{label} := pq.\text{extractLabel}();$ 
9   foreach outgoing edge  $e=(v,w)$  of  $v=\text{label.getNode}()$  do
10    if  $\text{isInfeasible}(e)$  then  $\text{continue};$  // ignore this edge
11     $\text{Label } \text{newLabel} := \text{createLabel}(\text{label}, e);$ 
12    if  $\text{newLabel}$  is dominated then  $\text{continue};$ 
13    //  $\text{newLabel}$  is not dominated
14     $pq.\text{insert}(\text{newLabel});$ 
15     $\text{labelListAt}(w).\text{insert}(\text{newLabel});$ 
16     $\text{labelListAt}(w).\text{removeLabelsDominatedBy}(\text{newLabel});$ 
```

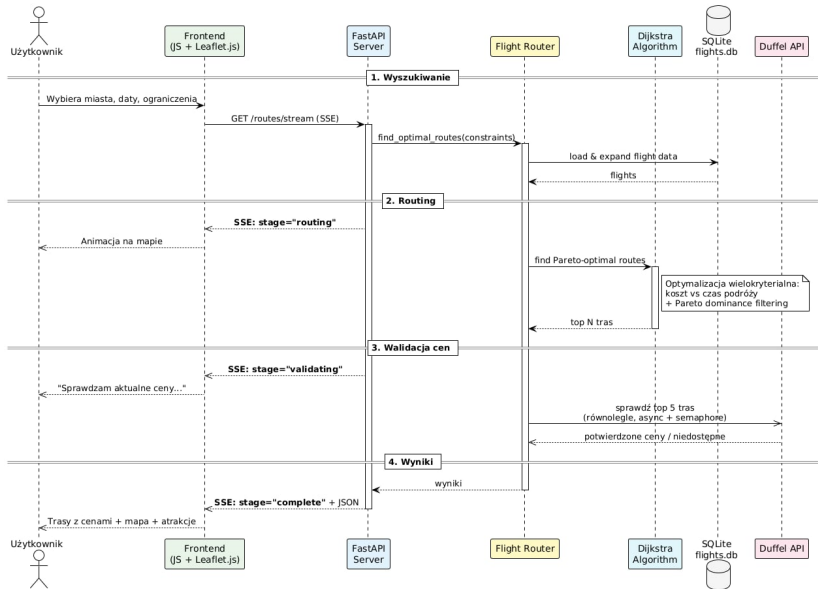
Algorithm 1. Pseudocode for the generalized Dijkstra algorithm

Walidacja

- ▶ Znalezione drogi są *walidowane*, tzn. sprawdzamy, czy sprawdzamy ich dostępność oraz zmieniamy ceny na aktualne.
- ▶ Top 5 tras jest weryfikowanych asynchronicznie z API w czasie rzeczywistym.

Diagram

ChadGuide - Flow wyszukiwania trasy



Git, dokumentacja, testy, ...

- ▶ Całość powstała dzięki systemowi kontroli wersji Git.
- ▶ Staraliśmy się prowadzić dokładną dokumentację poszczególnych modułów oraz paczek.
- ▶ Do wszystkich istotnych części projektu napisaliśmy testy, również te wydajnościowe. Do testów została użyta paczka `pytest`.

Co mogliśmy zrobić lepiej?

- ▶ Za dużo czasu zajęło nam znalezienie odpowiedniego dostawcy danych lotów.
- ▶ Niektóre części kodu są nadmiernie złożone.
- ▶ Złożoność zasadniczej części alogytmu jest duża, jeżeli miast które użytkownik chce odwiedzić jest dużo. Padł pomysł, aby tę część kodu przepisać na C, ale nie było na to czasu.

Podsumowanie

Czy wypełniliśmy cele? *Tak.*

Jest to nasz pierwszy *akademicki* projekt, którego, naszym zdaniem, nie powinniśmy się wstydzić.

Pytania?

Dziękujemy za uwagę.