Dokumentacja

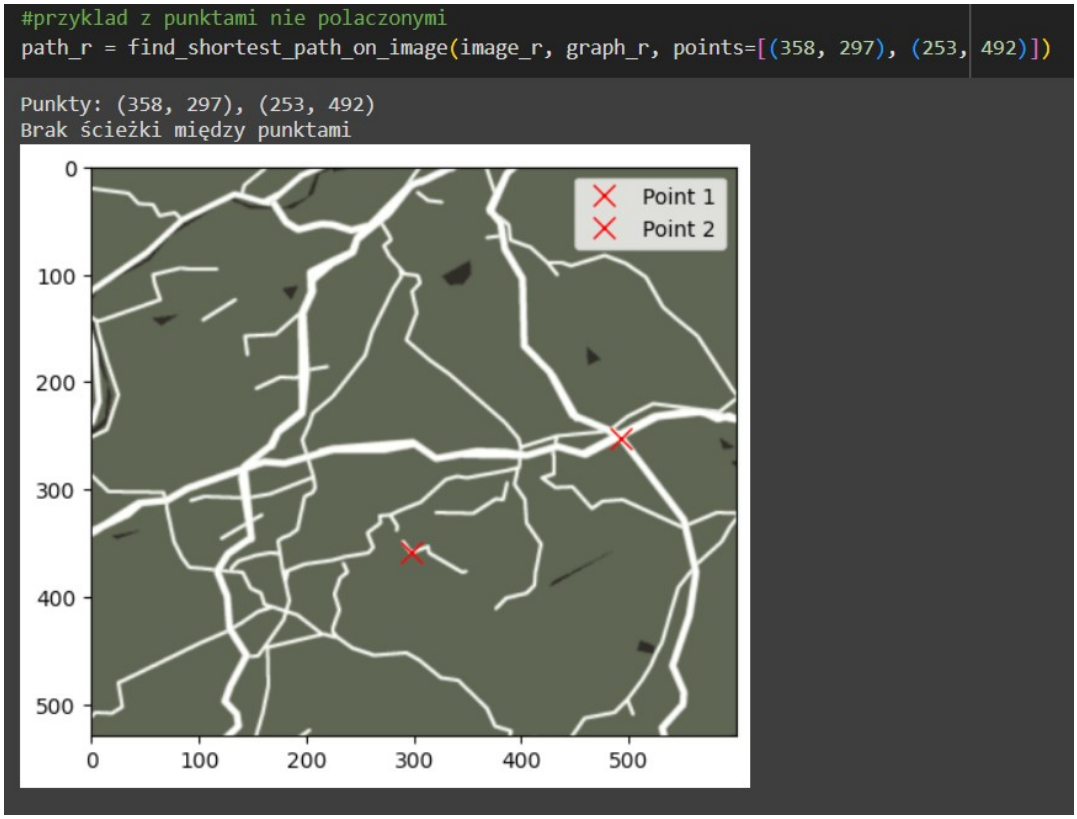
Projekt Analiza i Przetwarzanie Obrazów - Najkrótsza ścieżka

Bartłomiej Leśnicki, Rafał Walkowiak, Maciej Walczak, Tomasz Maczek, Jędrzej Szostak, Mikołaj Sondej

Spis treści

1. Opis zrealizowanego rozwiązania

Mapy generowane są za pomocą algorytmu Stable Diffusion w formacie png na podstawie zdefiniowanego przez nas promptu. Następnie wykonujemy konwersję danych do postaci zoptymalizowanej pod algorytm i zamieniamy obraz mapy png na graf. Jeśli nie zostały zdefiniowane to losowo wybieramy punkty początku i końcu. Na tak przygotowanym grafie dokonujemy poszukiwania najkrótszej ścieżki za pomocą algorytmu Dijkstry, wpierw sprawdzając czy oba punkty znajdują się na drodze i czy jest możliwość poprowadzenia między nimi trasy.



Rys.1. Przykładowa walidacja grafu i punktów.

Jeśli wszystko się zgadza to wykonywany jest algorytm i rysowana jest ścieżka.

Język:

- python

Algorytmy:

- Dijkstry - do znajdowania najkrótszej ścieżki.
- zamiana mapy w formacie png na graf.
- generacja danych pseudolosowych (map).

Narzędzia:

- Stable Diffusion
- biblioteka networkx: funkcja networkx.shortest_path() i networkx.Graph()
- biblioteka openCV
- biblioteka matplotlib
- biblioteka heapq
- biblioteka torch

2. Instrukcja obsługi

Projekt został wykonany w formacie notatnika Jupyter, dzięki czemu jest on łatwy w obsłudze. Aby poprawnie go wywołać należy po kolei uruchomić każdą z komórek. Dla użycia tylko części szukającej najkrótszej drogi należy najpierw zaimportować biblioteki z sekcji generowania obrazów. Przykładowe użycie programu zostało umieszczone w ostatnich dwóch sekcjach notatnika.

W przypadku chęci przetestowania programu dla innego zestawu danych należy zmienić jeden z poniżej opisanych parametrów:

A) Dla części generującej dane:

- **sizes** : tablica zawierająca rozmiary tworzonych map (bazowo: 768, 1024, 1440)

- **generators** : tablica generatorów, podczas jej inicjalizacji można zmienić jakim ziarnem mają posługiwać się poszczególne generatory (bazowo: 42 dla wszystkich trzech)

  Dokładniejszy opis powyższej części można znaleźć w punkcie 4. tej dokumentacji.

B) Dla części zajmującej się znalezieniem najkrótszej drogi, należy wywołać funkcję *find_shortest_path_on_image,* posiadającej trzy parametry:

- **image** : zmienna przetrzymująca mapę, na której będzie działał algorytm. Należy wybrać jeden z wygenerowanych obrazów znajdujący się w tablicy *images*, albo skorzystać z funkcji *load_map,* podając jako argument ścieżkę obrazu, jeśli chcemy wykorzystać już istniejący plik.

- **graph** : zmienna przetrzymująca graf, na której będzie działał algorytm.

Należy wyżej wybrany obraz użyć jako argument *prepare_graph* i wynik działania tej funkcji przypisać temu parametrowi.

- **points** : zmienna przetrzymująca współrzędne dwóch punktów, pomiędzy którymi będzie szukana najkrótsza ścieżka. Należy podać ich wartości w formacie: [(pointA_x, pointA_y), (pointB_x, pointB_y)]. W przypadku nie podania żadnej wartości, punkty te zostaną wylosowane z grafu.

3. Opis konfiguracji każdego stosowanego algorytmu i narzędzia:

3.1. Tworzenie map

Mapy są generowane przy pomocy paczki diffusers.StableDiffusionPipeline. Wykorzystujemy do tego pretrenowany model o id `= "CompVis/stable-diffusion-v1-4”.` Pozwala to nam na tworzenie obrazów na zasadzie text to image. Wykorzystując zdefiniowany przez nas prompt - `"A simple map of an urban area, roads are light color, road are wide, image is clean"` tworzymy w pętli wybraną przez nas liczbę obrazów przedstawiających to co zostało opisane. Przykładowe obrazy:

Rys. 2. Mapy generowane przy pomocy StableDiffusion

Jak widać na powyższym rysunku drogi są zaznaczone, jednak nie są one wyraźnie jaśniejsze od tła. Do osiągnięcia tego celu przeprowadziliśmy progowanie(THRESH_BINARY + THRESH_OTSU z biblioteki OpenCV), co pozwoliło na skontrastowanie dróg i tła, a następnie użyliśmy operacji morfologicznej - dylatacji z jądrem rozmiaru 2x2, co miało pozwolić usunąć szum i dopełnić wybrakowane elementy obiektu. Pozwoliło nam to otrzymać

taki efekt:

Rys. 3. Mapy po dylatacji i progowaniu.

W ten sposób otrzymaliśmy gotowe mapy do dalszego procesowania.

3.2. Konwersja map do grafów

Konwersja mapy na graf odbywa się poprzez wczytanie obrazu jako tablicy pikseli. Wczytywany jest każdy piksel i wokół niego algorytm sprawdza szerokość drogi w każdym z 8 kierunków (całe najbliższe sąsiedztwo piksela - lewo, prawo, góra, dół oraz kierunki ukośne). Algorytm idzie w danym kierunku po jednym pikselu aż natrafi na tło. Z każdym krokiem inkrementowana jest szerokość danego kierunku. Po natrafieniu na tło algorytm wraca do piksela startowego i powtarza procedure w kierunku przeciwnym inkrementując dalej tą samą szerokość(np. idziemy w lewo od piksela startowego potem w prawo zwiększając tą samą szerokość danego kierunku) . Po zbadaniu szerokości w wybranym kierunku oraz kierunku przeciwnym zapisywana jest szerokość danego kierunku i przechodzimy do innej płaszczyzny(np. sprawdzaliśmy w lewo i prawo to przechodzimy do góry i dołu. Po zbadaniu szerokości wszystkich płaszczyzn, wagi krawędzi między pikselem startowym a pikselami w danym kierunku(np. między pikselem startowym a o jeden wyżej i o jeden niżej) zapisywane są jako odwrotność szerokości.

3.3. Znajdywanie ścieżki

Tak stworzony graf jest podawany jako argument wywołania funkcji shortest_path z biblioteki NetworkX. Na wejściu funkcji shortest_path podajemy graf oraz źródłowy i docelowy wierzchołek, dla których chcemy znaleźć najkrótszą ścieżkę. Funkcja sprawdza, czy istnieje ścieżka między

źródłowym i docelowym wierzchołkiem. Jeśli nie istnieje, zostanie zgłoszony wyjątek NetworkXNoPath. Następnie funkcja wykorzystuje algorytm najkrótszej ścieżki, aby znaleźć najkrótszą ścieżkę między wierzchołkami. Jeśli nie podano żadnego algorytmu jako argument, domyślnie używany jest algorytm Dijkstry. Funkcja zwraca listę wierzchołków w kolejności, która tworzy najkrótszą ścieżkę między źródłowym i docelowym wierzchołkiem.

Zwrócona ścieżka jest rysowana na początkowej mapie dróg. (Algorytm zakłada, że dwa podane punkty muszą znajdować się na wyznaczonej drodze).

4. Konfiguracja stosowanych generatorów pseudolosowych.

Mapy są generowane przy pomocy paczki diffusers.StableDiffusionPipeline. Wykorzystujemy do tego pretrenowany model o id = "CompVis/stable-diffusion-v1-4". Pozwala to nam na tworzenie obrazów na zasadzie text to image. Wykorzystując zdefiniowany przez nas prompt - "A simple map of an urban area, roads are light color, road are wide, image is clean" tworzymy w pętli wybraną przez nas liczbę obrazów przedstawiających to co zostało opisane. Do stworzenia obrazów potrzebujemy Cudy, co pozwoli nam na zrównoleglenie operacji. definiujemy map_per_size, czyli liczbę obrazów które chcemy wygenerować dla danego zdefiniowanego w liście sizes. Definiujemy generator, który tworzy map_per_size obrazów na wcześniej opisanych rozmiarach i tworzymy pusty pojemnik na nasze w przyszłości wygenerowane mapy - images. Wszystkie zmienne są zdefiniowane na rysunku 4.

```
#Generowanie obrazów

prompt = "A simple map of an urban area, roads are light color, road are wide, image is clean"
map_per_size = 3
sizes = [768, 1024, 1440]
generators = [torch.Generator("cuda").manual_seed(42), torch.Generator("cuda").manual_seed(42), torch.Generator("cuda").manual_seed(42)]

images = []

for i in range(len(sizes)):
  tmp_images = []
  for j in range(map_per_size):
    image = pipe(prompt, generator=generators[i], height=sizes[i], width=sizes[i], num_images_per_prompt=1).images[0]
    tmp_images.append(image)
```

Rys. 4. Definiowane zmienne do generacji pseudolosowych map

Po wykonaniu tych kroków przechodzimy do generowania map. Wykonujemy to w pętli - w każdym kroku generowane są obrazy. Przekazujemy `prompt`, `generator`, wysokość - `height` oraz szerokość - `width`. Liczba iteracji zależy od rozmiaru listy `sizes`. Następnie dodajemy wygenerowane przez nas obrazy do wcześniej zdefiniowanej listy `images`.

5. Podział ról

Bartłomiej Leśnicki - Opracowanie oraz implementacja metody zamiany mapy na graf. Implementacja znajdowania najkrótszej ścieżki. Udokumentowanie działania tych algorytmów. Implementacja wczytywania oraz rysowania mapy. Testowanie programu do znajdowania ścieżki.

Rafał Walkowiak - Implementacja oraz testowanie generowania map. Implementacja przetwarzania wygenerowanych map na postać pozwalającą na użycie algorytmu znajdowania najkrótszej ścieżki.

Maciej Walczak - przygotowanie dokumentacji.

Tomasz Maczek - integracja części generującej mapy i algorytmu, testowanie ich wspólnego działania.

Jędrzej Szostak - przygotowanie dokumentacji.

Mikołaj Sondej – napisanie i testowanie alternatywnych algorytmów, testowanie zamiany mapy na graf