OBLICZENIA WYSOKIEJ WYDAJNOŚCI

Autorzy
KACPER KLUPŚ 252870
SZYMON CHORAŁA 252941

Prowadzący

Dr inż. Radosław Idzikowski

Spis treści

1 Etap 1 - wybór tematu

1.1 Temat i zakres projektu

Dynamic time warping to algorytm służący do pomiaru podobieństwa między dwoma sekwencjami czasowymi obliczający odległość np.: eukidlesową, w celu uzyskania optymalnego dopasowania. Aby tego dokonać dane wejściowe muszą zostać umieszczone w wektorach jednowymiarowych. Algorytm stosuje się nie tylko do dopasowywania wzorców, ale także do wykrywania anomalii, które mogą zostać zauważone poprzez nakładanie się na siebie szeregów czasowych. Poniższe ilustracje przedstawiają wykresy obrazujące dopasowywanie charakterystyk przy wykorzystaniu jedynie odległości Eukildesa i algorytmu DTW. Algorytm DTW gwarantuje równe, optymalne dopasowanie dwóch krzywych. Nie muszą być one idealnie zsynchronizowane.
Ograniczenia i reguł algorytmu DTW:

- Pierwszy indeks z pierwszej sekwencji musi być dopasowany do pierwszego indeksu z drugiej sekwencji,
- Ostatni indeks z pierwszego ciągu musi być dopasowany do ostatniego indeksu z drugiego ciągu,
- Odwzorowanie indeksów z pierwszej sekwencji na indeksy z drugiej sekwencji musi być monotonicznie rosnące i odwrotne. Żaden z przebiegów nie będzie cofał się w czasie w procesie dopasowywania,
- Każdy indeks z pierwszej sekwencji musi być dopasowany do jednego lub więcej indeksów z drugiej sekwencji i odwrotnie.

Podstawowe zastosowania algorytmu to:

- rozpoznawanie mowy - można określić czy dane wyrażenie pasuje do drugiego pomimo szybszego lub wolniejszego mówienia. Takie rozpoznawanie mowy wykorzystywane jest w smartfonach czy inteligentnych budynkach,
- rozpoznawanie podpisu online czy offline,
- rynki finansowe - porównywanie danych dotyczących obrotu akcjami w podobnych przedziałach czasowych, nawet jeśli nie są one idealnie dopasowane.
- fitness trackers - dokładniejsze obliczanie prędkości spacerowicza i liczby przebytych kroków,
- obliczanie planowanego przyjazdu kierowcy znając jego nawyki podróży.

W celu zrealizowania algorytmu planowane jest podzielenie obliczania macierzy dtw na więcej wątków procesora wykorzystując do tego bibliotekę multiprocessing w środowisku Python.

Multipreprocessing to technika polegająca na równoczesnym wykonywaniu wielu zadań (procesów) w jednym programie w celu zwiększenia wydajności i wykorzystania wielordzeniowych procesorów. Równoległe wykonywanie zadań planujemy rozwiązać na:

- Pierwszy sposób - główny, dotyczy równego podziału obu sygnałów dźwiękowych według ilości wątków procesora i porównywania według utworzonych przedziałów.
- Pozostałe możliwe do zrealizowania sposoby dotyczą np.: podziału jednego z sygnałów dźwiękowych według ilości wątków procesora i szukanie podobieństw względem całego zakresu sygnału drugiego lub porównywanie części sygnału 1 do kolejnych części sygnału 2 i wybranie minimum.

1.2 Przyspieszenie

Przyspieszenie jest miarą przyrostu wydajności wykonywanego zadania przy wykonywaniu go jako obliczenia równoległe. Można wyrazić je następującym wzorem:

$$S_p = \frac{T_1}{T_p} \tag{1}$$

Dla komputera o przykładowych parametrach procesora tj.: 4 rdzenie i 8 wątków dobre przyspieszenie pochodziłoby z zakresu (3,5).

Link do repozytorium na Github: https://github.com/Kacper314/OWW-AIP.git

2 Implementacja metody sekwencyjnej

Idea algorytmu polega na porównywaniu dwóch sekwencji punkt po punkcie, w taki sposób, aby minimalizować całkowitą różnicę pomiędzy nimi. Algorytm DTW jest w stanie dopasować sekwencje o różnej długości poprzez dopasowanie każdego punktu sekwencji do punktu w drugiej sekwencji, z uwzględnieniem możliwości "skoku"wzdłuż osi czasu.

Algorytm DTW składa się z kilku kroków:

1. Obliczenie macierzy odległości: Pierwszym krokiem jest obliczenie macierzy, zawierającej informacje o koszcie dopasowania punktów w obu sekwencjach. Owy koszt uzyskuje się obliczając odległość eukidlesową pomiędzy każdą parą punktów z obu sekwencji. Jest to tak zwany koszt lokalny.

```
    def dtw_distance(s1, s2):
    n, m = len(s1), len(s2)
    dtw = np.zeros((n+1, m+1))
    for i in range(1, n+1):
        dtw[i][0] = float('inf')
    for j in range(1, m+1):
        dtw[0][j] = float('inf')
    #dtw[0][0] = 0

    for i in range(1, n+1):
        for j in range(1, m+1):
            cost = abs(s1[i-1] - s2[j-1])
            dtw[i][j] = cost + min(dtw[i-1][j], dtw[i][j-1], dtw[i-1][j-1])

    return dtw[n][m], dtw

```

2. Utworzenie macierzy kumulacyjnej: Następnie tworzona jest macierz kumulacyjna, która odzwierciedla minimalną drogę pomiędzy początkiem i końcem obu sekwencji. W macierzy tej każdy element reprezentuje minimalną odległość pomiędzy danym punktem w pierwszej sekwencji, a punktem w drugiej sekwencji.

3. Wybór ścieżki o minimalnym koszcie: Ostatnim krokiem jest wybór ścieżki o minimalnym koszcie pomiędzy początkiem i końcem obu sekwencji. Ścieżka ta zostanie następnie użyta do dokładnego dopasowania obu sekwencji.

Algorytm DTW może być modyfikowany w zależności od konkretnego zastosowania. Na przykład, w przypadku analizy sygnałów dźwiękowych, można wykorzystać funkcję wag, aby uwzględnić fakt, że niektóre częstotliwości są bardziej istotne niż inne. Algorytm ten jest również w stanie poradzić sobie z sytuacjami, w których jedna sekwencja jest znacznie dłuższa niż druga, co może wynikać z braku pewnych zdarzeń lub pojawienia się dodatkowych zdarzeń w krótszej sekwencji.
Metoda sekwencyjna została również zaimplementowana poprzez wykorzystanie okien, które służą do określenia przedziałów porównywania sygnałów. Aby okno miało wpływ na wynik

działania algorytmu musi być ono większe od wartości bezwzględnej z różnicy długości porównywanych sygnałów.

3 Implementacja metody równoległej

Wykorzystana technika obliczeń równoległych to użycie wielu wątków procesora. Biblioteka Multiprocessing udostępnia funkcję do podziału programu na dostępne wątki procesora. W pierwszym etapie należy stworzyć środowisko Pool i wprowadzić do niego parametr informujący o ilości dostępnych wątków w urządzeniu. Druga część to odpowiedni podział danych w zależności od zakresów, które chcemy porównywać. Wykonanie funkcji przy wykorzystaniu wielu wątków umożliwia funkcja starmap. Do jej wywołania potrzebna jest zaimplementowana funkcja którą ma wykonywać oraz zakres danych. Zakresy danych do porównywania różnią się w zależności od ilości używanych wątków procesora.

```
ranges1 = [(i * chunk_size1, (i + 1) * chunk_size1) for i in range(num_processes)]
ranges2 = [(i * chunk_size2, (i + 1) * chunk_size2) for i in range(num_processes)]
ranges_test = [(data11[ranges1[i][0]:ranges1[i][1]], data22[ranges2[i][0]:ranges2[i][1]]) for
    i in range(num_processes)]
```

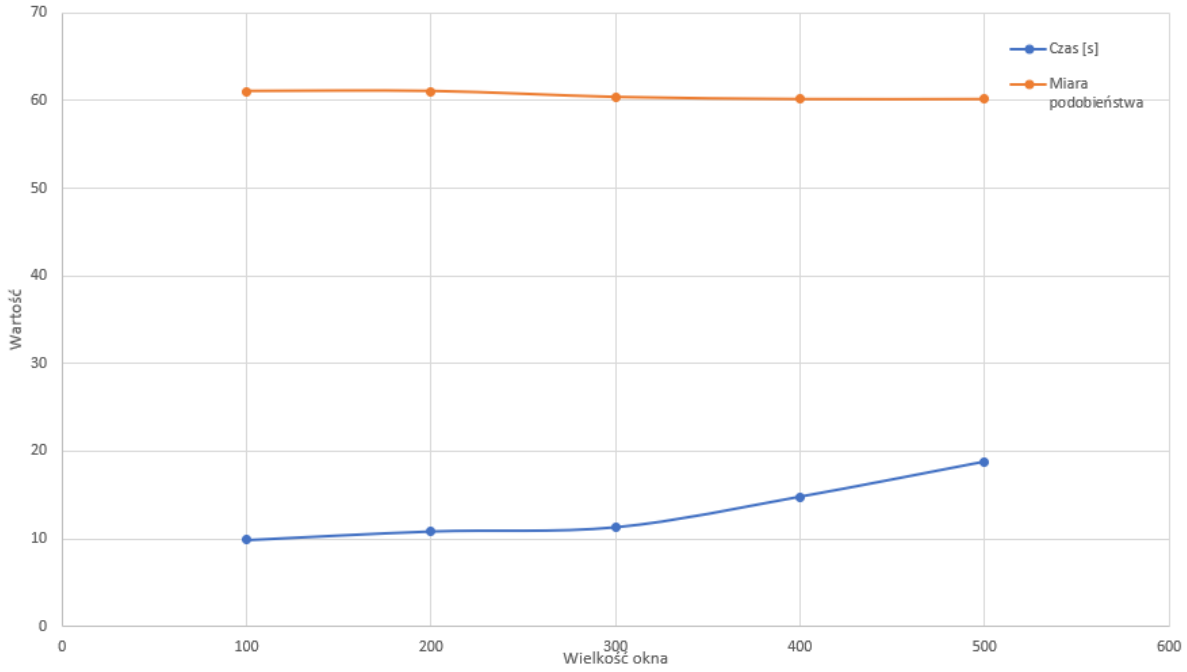
Pierwszy sposób implementacji metody równoległej przedstawia porównanie do siebie poszczególnych części sygnału pierwszego do sygnału drugiego. W tym przypadku nie ma możliwości dopasowania jednej próbki pierwszego sygnału do kilku próbek drugiego sygnału.
Metoda równoległa posiada również drugi sposób implementacji polegający na porównywaniu wyznaczonych fragmentów sygnału w pętli przez co każda część pierwszego sygnału została porównana do każdej części drugiego sygnału. W takim przypadku kilka próbek pierwszego sygnału mogło zostać dopasowanych do jednej próbki sygnału drugiego.
W następnych paragrafach przedstawione zostały przykłady wywołania funkcji dla podzielonych fragmentów. Policzone zostały czasy wykonywania algorytmów, na których postawie obliczono przyspieszenie oraz przedstawiono wnioski na podstawie przeprowadzonych badań.

4 Badania

Czas działania algorytmu dla badanych sygnałów metodą sekwencyjną oraz miara podobieństwa wynoszą:

Czas wykonania Algorytmu	6,309 s
Podobieństwo	60,159

Poniżej przedstawione zostały czasy wykonywania algorytmu oraz wartości miary podobieństwa dla metody sekwencyjnej z różnymi wartościami okien.



Rysunek 1: Porównanie wskazań dla różnych wielkości okna - metoda sekwencyjna

- Badanie 1 - podział sygnałów czasowych na wszystkie wątki procesora.

Tabela 1: Wyniki dla badania 1

Ilość wątków	8	6	4
Czas wykonywania algorytmu	1,094 s	1,232 s	1,811 s
Podobieństwo	63,744	63,102	62,687

Tabela 2: Miary podobieństwa dla 8 wątków

Miara podobieństwa dla poszczególnych wątków							
1.	2.	3.	4.	5.	6.	7.	8.
9,712	13,405	8,751	5,219	9,383	7,157	6,066	4,047

Dla powyższego przypadku przyspieszenie wynosi:

$$S_p = \frac{T_i}{T_p} = \frac{6,30}{1,09} \approx 5,77 \tag{2}$$

- Badanie 2 - podział obydwu sygnałów na części i szukanie podobnych części.

Tabela 3: Wyniki dla badania 2

Ilość wątków	8	6	4
Czas wykonywania algorytmu	2,298 s	2,478 s	2,728 s
Podobieństwo	52.465	52.406	54.899

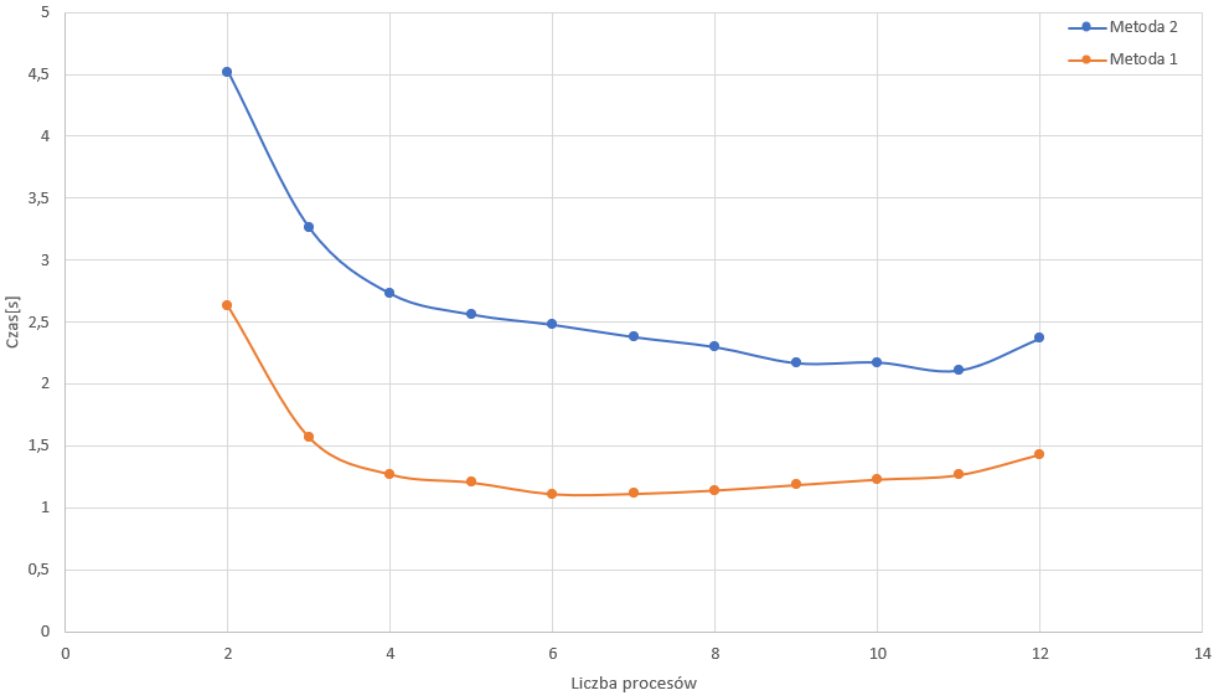
Tabela 4: Miary podobieństwa dla 8 wątków

Miara podobieństwa dla poszczególnych wątków							
1.	2.	3.	4.	5.	6.	7.	8.
8,901	11,701	6,469	4,169	7,546	6,486	3,148	4,047

Dla powyższego przypadku przyspieszenie wynosi:

$$S_p = \frac{T_i}{T_p} = \frac{6,30}{2,20} \approx 2,86 \tag{3}$$

Przyspieszenie jest znacznie mniejsze niż w poprzednim przypadku. Wynika to z tego, że każdy wątek ma do wykonania znacznie więcej pracy niż w poprzedniej metodzie. Wynik działania algorytmu jest lepszy niż w metodzie sekwencyjnej co może oznaczać, że algorytm powiązał ze sobą fragmenty występujące daleko od siebie.



Rysunek 2: Otrzymany rezultat dla metody równoległej

5 Wnioski

- Pierwszy sposób implementacji metody równoległej znacznie przyspieszył działanie programu. Jednak miara dopasowanie jest nieznacznie większa. Może to wynikać z podziału danych na przedziały do porównania. Próbki bardziej odpowiednie mogą znajdować się w innym przedziale i nie zostaną przypisane.
- Drugi metoda implementacji metody równoległej przyspieszyła działanie programu w mniejszym stopniu niż metoda pierwsza, jednak wynik działania programu jest nieznacznie lepszy niż w metodzie sekwencyjnej. Może to wynikać z dopasowania kilku przedziałów do tego samego przedziału lub dopasowaniu przedziałów w nieodpowiedniej kolejności.
- Z przeprowadzonej analizy wynika, że optymalna liczba wątków procesora różni się w zależności od zastosowanej metody. W przypadku pierwszej metody równoległej, stwierdzono, że najlepsze rezultaty uzyskuje się przy użyciu około 6-7 wątków procesora, natomiast dla drugiej metody stwierdzono, że optymalna liczba użytych wątków wynosi 11. Wskazuję to na zależność między złożonością metody/problemu, a wydajnością programu przy użyciu różnej ilości wątków.
- Wykorzystanie okien czasowych w metodzie sekwencyjnej wiąże się z dodatkowym podziałem i porównywaniem sygnałów przez co czas obliczeń jest większy niż w przypadku klasycznej metody sekwencyjnej. W tym przypadku przyspieszenie osiągnie większe wartości. Okno powyżej wartości 350 wpływa jedynie na długość obliczeń (dal tego konkretnego przypadku).
- Analiza przeprowadzonych eksperymentów nad zrównolegleniem problemu DTW wykazała, że obie przedstawione metody posiadają istotne wady, co czyni zrównoleglenie tego problemu wyzwaniem. Pierwsza zastosowana metoda, mimo użycia większej liczby wątków, charakteryzuje się zawsze delikatnie gorszym wynikiem w porównaniu do metody sekwencyjnej. Co istotne, obserwuje się, że im większa liczba użytych wątków, tym gorszy jest osiągany rezultat. Z kolei druga metoda, mimo osiągania zawsze nieznacznie lepszego wyniku niż metoda sekwencyjna, budzi uzasadnione wątpliwości co do rzetelności rezultatów. Wyniki te sugerują, że osiągnięta poprawa jest zakłamana, ponieważ nie może być lepsza niż metoda sekwencyjna. W związku z powyższym, wnioskiem jest stwierdzenie, że problem DTW nie nadaje się idealnie do zrównoleglenia, a obie metody wymagają dalszych analiz i ewentualnych modyfikacji celem uzyskania bardziej wiarygodnych wyników przy zastosowaniu tej techniki.

Literatura

[1] https://www.databricks.com/blog/2019/04/30/understanding-dynamic-time-warping.html.

[2] Muller, Meinard. "dynamic time warping."information retrieval for music and motion: 69-84. 2007.

[3] S. Salvador, P. Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007.