

# [MED]

## Dokumentacja Końcowa

Piotr Frątczak  
(300207)

Bartłomiej Binda  
(300188)

7 czerwca 2022

## 1 Zadanie

### 1. Reguły asocjacyjne

Implementacja algorytmu do odkrywania reguł asocjacyjnych (Apriori, Eclat, ...) pozwalająca na zastosowanie hierarchii elementów m.in. 3 poziomowej. Należy dopuścić reguły  $a \rightarrow Ah$ , gdzie  $a$  – jest elementem w transakcji,  $Ah$  – elementem z hierarchii, do którego jest przypisany m.in. element  $a$ , jeśli są transakcje  $t : t \supset (a, b), a \in E(Ah), b \in E(Ah)$ , gdzie  $E(y)$  – zbiór elementów przypisanych do elementu  $y$  hierarchii.

W dalszej części dokumentu, powyżej wyjaśniona reguła opierająca się na hierarchii zostaje określona jako reguła hierarchiczna dla ułatwienia i skrócenia zapisu.

## 2 Realizacja

### 2.1 Wybrany Algorytm

Zaimplementowano algorytm ECLAT, ponieważ jest bardziej wydajny i skalowalny niż algorytm Apriori. Różnica wynika ze sposobu przeszukiwania przestrzeni. Apriori działa w kierunku poziomym, imitując przeszukiwanie grafu wszerek, podczas gdy ECLAT działa w kierunku pionowym i imituje przeszukiwanie grafu wglęb.

### 2.2 Zbiory Danych

Oba zbiory danych zostały przekształcone do formatu SPMF przez Ying Wang et al..

- Fruithut - zbiór danych zawierający 181970 transakcje klientów sklepu detalicznego w USA sprzedającego owoce. Zbiór danych zawiera 1265 różnych produktów. Średnio przypada 3,58 produktów na transakcję. Udostępniono taksonomię elementów zbioru, składa się ona z 4 poziomów i 43 kategorii.
- Liquor\_11 - zbiór danych zawierający 9284 transakcje klientów ze sklepów alkoholowych w stanie Iowa, USA. W zbiorze danych zawarte są wszystkie transakcje, w których maksymalnie było kupionych 11 produktów. Udostępniono taksonomię elementów zbioru, składa się ona z 7 poziomów i 77 kategorii.

## 2.3 Przyjęte Założenia

- Algorytm pozwala na opcjonalne zastosowanie hierarchii elementów, aby odnaleźć reguły hierarchiczne. Jeśli hierarchia nie zostanie podana, algorytm jej nie wykorzysta.
- Algorytm odnajduje podstawowe reguły oraz specjalne reguły hierarchiczne, nie odnajduje innych reguł na podstawie hierarchii.
- Możliwa jest zmiana parametrów uruchomienia algorytmu ECLAT.
- Wykryte reguły asocjacyjne zostają zapisane do pliku CSV.
- Algorytm może zostać uruchomiony na podstawie danych użytkownika.

## 3 Implementacja

### 3.1 Najważniejsze Elementy Implementacji

#### 3.1.1 Wyszukiwanie Zbiorów Częstych

Funkcja: `eclat.core.eclat.frequent_itemsets()`.

- 1: Zainicjuj pusty zbiór zbiorów częstych *frequent* i dodaj do niego pobrane jednoelementowe zbiory częste.
- 2: Dla  $n = 1..$ :
- 3:   Porównaj wszystkie pary zbiorów częstych *itemset1* i *itemset2* od długości  $n$ .
- 4:   Jeśli (posortowane po elementach) zbiory *itemset1* i *itemset2* różnią się tylko na ostatniej pozycji:
- 5:     Znajdź przecięcie *tidlist1* i *tidlist2* (należących do zbiorów *itemset1* i *itemset2*) oraz zapamiętaj je jako *tidlist*.
- 6:   Jeśli *tidlist* ma liczbę transakcji większą niż *min\_sup*:
- 7:     Dodaj nowy zbiór częsty wraz z jego *tidlistą* *tidlist* do *frequent* będący sumą zbiorów *itemset1* i *itemset2*.
- 8:   Jeśli dla  $n$  nie znaleziono żadnego zbioru częstego, zakończ wykonanie i zwróć *frequent*.

#### 3.1.2 Generowanie Reguł Asocjacyjnych

Funkcja: `eclat.core.eclat.rule_gen()`

- 1: Zainicjuj pusty zbiór reguł częstych *ar*.
- 2: Dla  $length = min\_len..max\_len$ :
- 3:   Dla każdego zbioru częstego *itemset* z *frequent* o długości  $length$ :
- 4:     Dla każdej długości następnika  $suc\_len = 1..length$ :
- 5:       Dla każdej kombinacji następnika *suc* o długości  $suc\_len$  i poprzednika *pred*:
- 6:       Przejdź do następnej kombinacji, jeśli któryś z *suc* i *pred* nie jest zbiorem częstym.
- 7:       Oblicz wsparcie reguły, jeśli jest większe niż *min\_conf*, dodaj regułę do *ar*.
- 8: Zwróć *ar*.

### 3.1.3 Generowanie Reguł Hierarchicznych

Funkcja: `eclat.core.eclat.hierarchy_rule()`

- 1: Zainicjuj pusty zbiór reguł *rules*.
- 2: Dla każdego dwuelementowego zbioru częstego *itemset*:
- 3:     Znajdź listę wszystkich przodków *ancestors1* i *ancestors2* obu elementów *item1* i *item2* zbioru *itemset*.
- 4:     Dla każdego wspólnego elementu  $h_{item}$  z list przodków *ancestors1* i *ancestors2*:
- 5:         Dodaj reguły  $item1 \rightarrow h_{item}$  i  $item2 \rightarrow h_{item}$  do *rules*.
- 6: Zwróć *rules*.

## 3.2 Środowisko Deweloperskie

Język programowania: Python 3.9.

Biblioteki:

- Pandas - Przygotowanie danych do przetwarzania.
- Matplotlib - Wizualizacja danych, w szczególności rysowanie wykresów.
- Seaborn - Style wykresów do Matplotlib.
- Scipy - Interpolacja wykresów.
- Numpy - Generacja danych do interpolacji.

## 3.3 Kod Źródłowy

Kod źródłowy jest dostępny w repozytorium: [Piotrfratczak/eclat](#).

## 3.4 Instrukcja Użycia

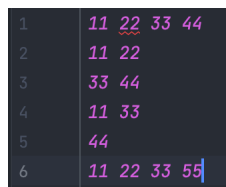
Instrukcja użycia oprogramowania jest zawarta w pliku README w folderze głównym projektu i na stronie głównej repozytorium kodu.

## 3.5 Format Danych Wejściowych

### 3.5.1 Transakcje

Format SPMF: Każda linia to kolejna transakcja, w ramach każdej linii wypisane są elementy oddzielone spacją.

Przykład:



1	11 22 33 44
2	11 22
3	33 44
4	11 33
5	44
6	11 22 33 55

Rysunek 1: Przykład pliku wejściowego dla transakcji.

### 3.5.2 Hierarchia

Każda linia to kolejna krawędź z drzewa hierarchii. Po lewej znajduje się dziecko, po prawej rodzic, są oddzieleni przecinkiem.

Przykład:

1	11,1
2	22,1
3	33,2

Rysunek 2: Przykład pliku wejściowego dla hierarchii.

## 3.6 Format Danych Wyjściowych

Każda linia to kolejna reguła asocjacyjna. Zaczynając od lewej znajdują się elementy poprzednika oddzielone przecinkiem. Po średniku znajdują się elementy następnika reguły oddzielone przecinkiem. Po kolejnym średniku znajduje się wsparcie reguły i po ostatnim średniku zaufanie reguły.

Przykład:

1	predecessor;successor;support;confidence
2	22,33;11;2;1.0000
3	11,33;22;2;0.6667
4	11,22;33;2;0.6667
5	22;11,33;2;0.6667
6	44;5;3;1.0000

Rysunek 3: Przykład pliku wyjściowego.

## 4 Testy Poprawności

Testy poprawności algorytmu zostały zrealizowane jako testy jednostkowe z wykorzystaniem danych syntetycznych. Wprowadzano specjalnie przygotowane dane, które miały zwrócić oczekiwany wynik. Na tej podstawie oceniono, że algorytm działa poprawnie.

Testy znajdują się w module `test.test_eclat`. Instrukcja uruchomienia testów jest dostępna w pliku README.

## 5 Eksperymenty Wydajnościowe

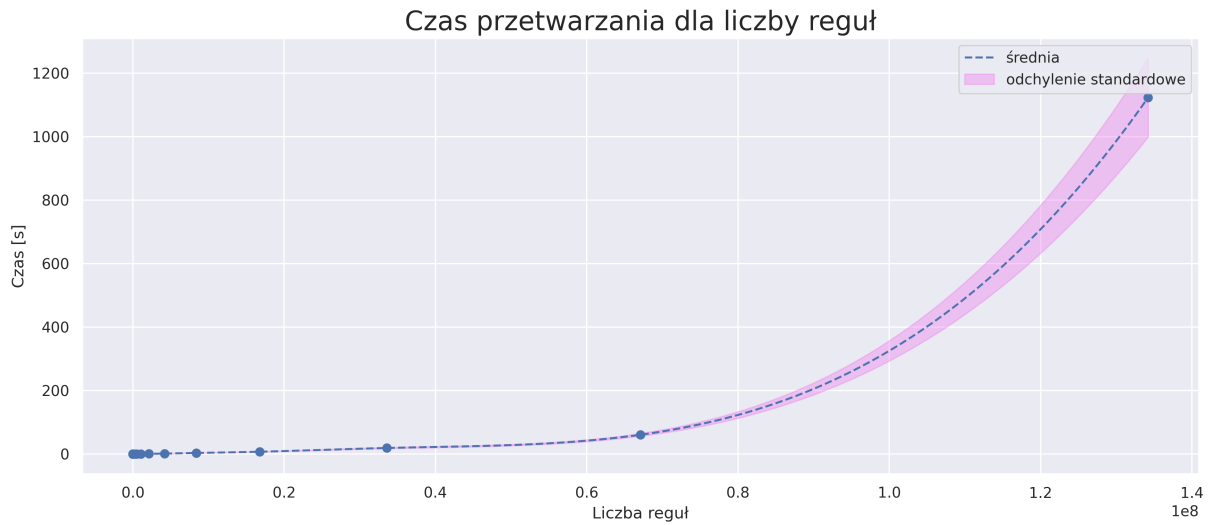
Eksperymenty mające na celu zbadanie wydajności implementacji zostały przeprowadzone na syntetycznym zbiorze danych. Każdy eksperyment został powtórzony 11 razy.

Eksperymenty znajdują się w module `test.test_efficiency`. Instrukcja uruchomienia eksperymentów jest dostępna w pliku README.

## 5.1 Liczba Reguł

Eksperyment został przeprowadzony przez mierzenie czasu wykonania algorytmu dla kolejnego dwukrotnego zwiększania listy transakcji zaczynając od transakcji dwuelementowej. Nie dodano hierarchii elementów.

### 5.1.1 Wyniki



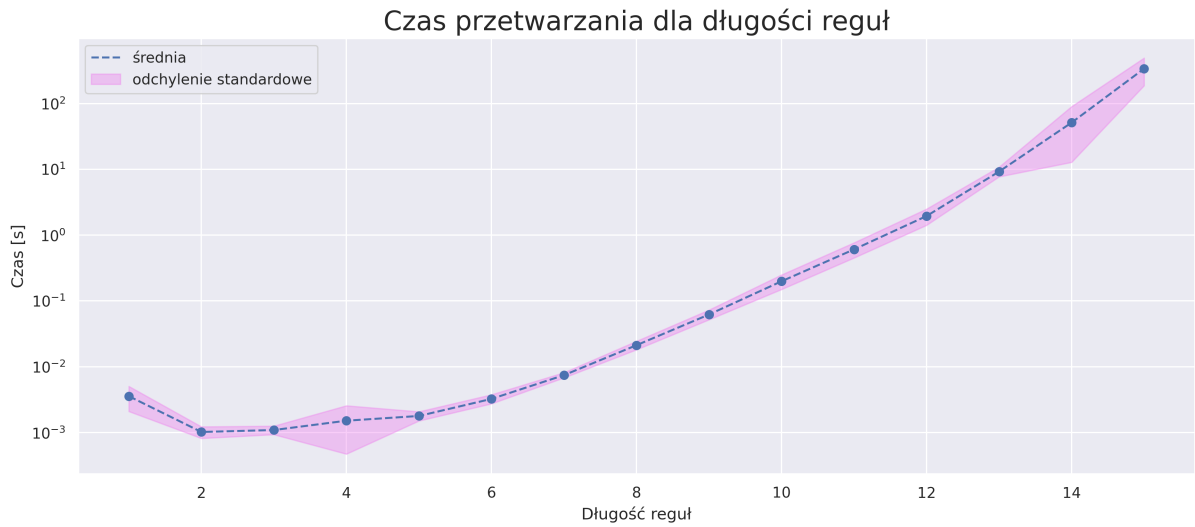
Rysunek 4: Wykres czasu wykonania dla liczby reguł.

Zgodnie z wykresem na Rysunku 4, wraz ze wzrostem liczności listy transakcji, czas wykonania rośnie wykładniczo. Wraz ze wzrostem czasu wykonania rośnie również wartość bezwzględna odchylenia standardowego, ponieważ różnica pomiędzy poszczególnymi czasami wykonania jest coraz większa.

## 5.2 Długość Reguł

Eksperyment został przeprowadzony przez mierzenie czasu wykonania algorytmu przy zwiększaniu liczby elementów w każdej transakcji o kolejny element. Nie dodano hierarchii elementów.

### 5.2.1 Wyniki



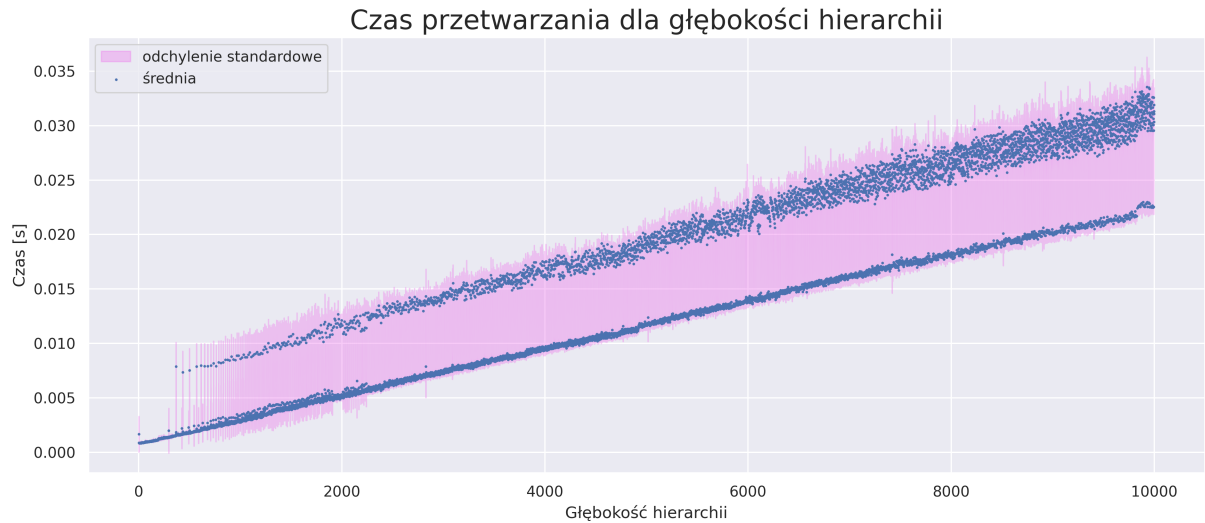
Rysunek 5: Wykres czasu wykonania dla długości reguł.

Wraz ze wzrostem liczności listy transakcji, czas wykonania rośnie wykładniczo, co zostało przedstawione na wykresie na Rysunku 5. Szybkość wzrostu czasu przetwarzania jest zdecydowanie wyższa niż dla zwiększenia liczności transakcji.

### 5.3 Głębokość hierarchii

Eksperyment został przeprowadzony dla pojedynczej transakcji dwuelementowej oraz taksonomii, której głębokość była zwiększana o 1 dla kolejnego pomiaru. Oba elementy w transakcji należały do tego samego elementu hierarchii. W każdym kolejnym pomiarze dla najstarszego elementu w hierarchii dodawano jego rodzica.

#### 5.3.1 Wyniki



Rysunek 6: Wykres czasu wykonania dla głębokości hierarchii.

Według wykresu na Rysunku 6, wraz ze wzrostem głębokości hierarchii elementów, czas wykonania rośnie liniowo. Jednak zauważalna jest pewna nieregularność dla niektórych licznosci hierarchii. Zbadano czy część czasów nie ma wyższego czasu wykonania co pewien okres, jednak okazuje się, że nie ma tutaj regularnej cykliczności. Przeprowadzono eksperyment ponownie w celu zbadania czy nieregularność nie była spowodowana obciążeniem procesora, jednak wyniki wyszły bardzo podobne.

### 5.4 Wnioski

Stopień wzrostu czasu wykonania dla zwiększającej się liczby transakcji i długości transakcji jest wykładniczy zgodnie z przewidywaniami. Jednak bardziej zaskakujący wynik uzyskano dla reguł hierarchicznych. Sugeruje to nieregularność dla niektórych wypadków przetwarzania, które trudno wytłumaczyć.

Jeśli chodzi o szybkość pojedynczego wykonania, algorytm najprawdopodobniej osiągałby krótsze czasy wykonania dla implementacji w języku programowania takim jak C, C++ czy Java, jednak w eksperymentach zwracaliśmy uwagę przede wszystkim na względne wartości czasów wykonania do zwiększających się parametrów liczby transakcji, długości transakcji i głębokości hierarchii.