# Algorytmy optymalizacji zapytań eksploracyjnych z wykorzystaniem materializowanej perspektywy eksploracyjnej

**Jerzy Brzeziński, Mikołaj Morzy,**
**Tadeusz Morzy, Łukasz Rutkowski**

**RB-006/02**

# 1. Wstęp

## 1.1. Rozwój technologii informatycznych

Rozwój technologii informatycznych w ostatnich latach doprowadził do znacznego postępu w metodach zdobywaniu wiedzy, czego widocznym wynikiem jest zjawisko powstawania „społeczeństwa informacyjnego". Społeczeństwo informacyjne charakteryzuje się dążeniem do ciągłej wymiany informacji i korzysta z już istniejącego dorobku informacyjnego. Wzrost ilości gromadzonych danych jest możliwy dzięki rozwojowi systemów baz danych. Relacyjne systemy baz danych znajdują zastosowanie w przechowywaniu, porządkowaniu i udostępnianiu rosnącej ilości informacji.

Jednym z głównych powodów gromadzenia i udostępniania informacji jest możliwość szczegółowej analizy informacji za pomocą technik analitycznych. W tym celu powstały wielowymiarowe modele danych, wykorzystywane w środowisku magazynów danych. Magazynem danych (ang. *data warehouse*) nazywamy uporządkowany tematycznie, zintegrowany i wzbogacony o wymiar czasowy zbiór danych. Głównym powodem, dla którego dane są ładowane do magazynu danych, jest możliwość wykorzystania wiedzy zawartej w danych do wspomagania podejmowanie decyzji ekonomicznych. W praktyce magazyn danych to narzędzie biznesowe zintegrowane z systemem zarządzania relacyjną bazą danych służące menedżerom firm do podejmowania decyzji strategicznych.

Podstawową techniką służącą do odkrywania wiedzy zawartej w danych jest eksploracja danych (ang. *data mining*, *knowledge discovery in databases*). Eksploracja danych to proces odkrywania nowych, nieznanych, pożytecznych i zrozumiałych wzorców i zależności w dużych wolumenach danych. Wiedza odkrywana w danych może być prezentowana za pomocą wielu różnych modeli, m.in. w postaci reguł asocjacyjnych, zbiorów częstych, wzorców sekwencji i innych.

## 1.2. Proces eksploracji i materializacja wyników

Eksploracja danych to proces iteracyjny i interaktywny, w którym użytkownik definiuje zbiór interesujących go wzorców, określa eksplorowany zbiór danych oraz wartości parametrów eksploracji. Iteracyjność procesu wynika z tego, że użytkownik dochodzi do odkrycia interesujących go wzorców stopniowo, w każdym kroku dostrajając wykorzystywany algorytm lub coraz dokładniej określając zbiór eksplorowanych danych. Interaktywność procesu wynika z modelu przetwarzania zapytań. Użytkownik wprowadza zapytanie, uzyskuje w odpowiedzi zbiór wzorców, ocenia ich przydatność i na podstawie uzyskanych wyników może zmodyfikować swoje zapytanie. Jest bardzo ważne, aby czas odpowiedzi systemu był wystarczająco krótki, aby umożliwić użytkownikowi interaktywną formę pracy.

Z punktu widzenia użytkownika wynik eksploracji jest zatem rodzajem odpowiedzi na zaawansowane zapytanie do bazy danych. Z punktu widzenia projektanta systemu informatycznego służącego do eksploracji danych istnieją trzy podstawowe czynniki, które powodują, iż system do eksploracji nie spełnia stawianego warunku interaktywności. Są to czas przetwarzania zapytań, rozmiar danych źródłowych oraz rozmiar wyników.

Z jednej strony należy zachować krótki czas przetwarzania, z drugiej strony proces eksploracji dokonywany jest zwykle na danych o dużych rozmiarach (rzędu tysięcy i milionów rekordów) i jest procesem obliczeniowo kosztownym i czasochłonnym. Często zdarza się, że rozmiar odpowiedzi na zapytanie eksploracyjne przekracza wielokrotnie rozmiar eksplorowanych danych. Taka charakterystyka procesu eksploracji powoduje, że na obecnym etapie rozwoju systemy eksploracji danych nie umożliwiają przetwarzania interaktywnego.

Jednym ze sposobów rozwiązania powyższego problemu jest zastosowanie materializacji wyników poprzednio wykonanych eksploracji. Ponieważ użytkownik dokonuje eksploracji wielokrotnie, nieznacznie zmieniając parametry procesu, zatem wykorzystanie wcześniejszych wyników do przetwarzania następujących po sobie zapytań może znacznie zredukować czas całego procesu.

## 1.3. Wcześniejsze prace

Prace nad perspektywami materializowanymi rozpoczęto w latach 80-tych. Początkowo perspektywy były narzędziem do przyśpieszania wykonywania zapytań i udostępniania starszych kopii danych. Ze względu na to, że zawartość perspektywy staje się nieaktualna po zmianach dokonanych w tabeli bazowej, na której ta perspektywa jest zdefiniowana, opracowano algorytmy pielęgnacji perspektyw materializowanych. Podsumowanie i klasyfikację różnych technik pielęgnacji perspektyw można znaleźć w [5], a obszerne omówienie tematu w [6].

Problem odkrywania reguł asocjacyjnych po raz pierwszy sformułowano w [1]. W [2] wprowadzono pojęcie zbioru częstego i zaproponowano algorytm *Apriori*, który stał się podstawą bardzo wielu różnych metod odkrywania wzorców. W [3,4] zaproponowano algorytmy korzystające z uzyskanych wcześniej wyników do odkrywania zbiorów częstych. Skracają one znacznie czas działania algorytmu *Apriori* przez przetwarzanie jedynie zmienionej części bazy danych. Inną propozycję przedstawiono w [11]. Zaprezentowany tam algorytm skracał czas odkrywania wzorców zarówno w zmniejszonej, jak i w zwiększonej bazie danych. Podstawą jego działania było pojęcie negatywnej granicy, zdefiniowanej w [12].

W [10] po raz pierwszy przedstawiono proces znajdowania zbiorów częstych jako proces interaktywny i iteracyjny, w którym użytkownik wielokrotnie zadaje zapytania eksploracyjne ze zmienionymi parametrami. Zaproponowano tam stworzenie podręcznej pamięci wiedzy (ang. *knowledge cache*), przechowującej odkryte zbiory częste i wartość ich wsparcia. Koncepcja obejmowała współdzielenie pamięci podręcznej przez różnych użytkowników i różne aplikacje. Dzięki tej metodzie użytkownicy mogą wykorzystywać wzajemnie wyniki swoich badań. W pracy tej zaproponowano również kilka schematów zarządzania zawartością pamięci podręcznej. W [7] zaproponowano pojęcie Systemu Zarządzania Wiedzą i Danymi (ang. *Knowledge Data Management System*). Według autorów system taki powinien być następcą współczesnych systemów zarządzania bazami danych i powinien być zintegrowany z istniejącymi rozwiązaniami informatycznymi, takimi jak systemy zarządzania bazami danych i magazyny danych.

W [13] zaproponowano obliczanie wsparcia zbiorów częstych w partycjach bazy danych. Koncepcja wykorzystywała to, że zbiór może być częsty wtedy i tylko wtedy, gdy jest częsty w co najmniej jednej partycji. Działanie algorytmu polega na wczytywaniu do pamięci operacyjnej partycji bazy danych, odkryciu w nich zbiorów częstych i na ich podstawie wyznaczaniu zbiorów częstych w całej bazie danych.

W raporcie dokonano identyfikacji najbardziej kosztownych pod względem czasu przetwarzania elementów algorytmu *Apriori* i zaproponowano modyfikacje algorytmu, które umożliwiają wykorzystanie wcześniejszych wyników i znacząco redukują czas odkrywania wzorców w dużych wolumenach danych.

# 2. Pojęcia i definicje

## 2.1. Podstawowe pojęcia z zakresu baz danych

**Perspektywa zwykła** (ang. *view*) jest wywiedzioną tabelą, zdefiniowaną w oparciu o tabele bazowe [5]. Perspektywa definiuje funkcję ze zbioru tabel bazowych do tabeli wywiedzionej (w postaci zapytania SQL). Funkcja ta jest zazwyczaj obliczana przy każdym odwołaniu do perspektywy. Perspektywa jest swego rodzaju „oknem", przez które użytkownik odczytuje lub modyfikuje dane w zbiorze tabel, na których została zdefiniowana. Perspektywa charakteryzuje się następującymi cechami:

- ogranicza zakres dostępnych danych do atrybutów i krotek określonych w definicji perspektywy,
- jest definiowana na bazie co najmniej jednej tabeli bazowej lub innej perspektywy,
- jest pamiętana w systemie wyłącznie w postaci swojej definicji.

Podstawową wadą perspektywy zwykłej jest to, że zapytanie które ją definiuje jest wykonywane przy każdym odwołaniu do perspektywy.

**Perspektywa materializowana** (ang. *materialized view*) to perspektywa, której zawartość jest zmaterializowana w bazie danych. Na materializowanej perspektywie można zakładać indeksy, przez co dostęp do danych zawartych w perspektywie jest dużo szybszy niż ponowne wyliczenie perspektywy.
Podstawową wadą perspektywy materializowanej jest to, że jej zawartość staje się nieaktualna po modyfikacji tabel bazowych definiujących tę perspektywę. Proces modyfikowania perspektywy materializowanej w wyniku zmian zachodzących w tabeli bazowej nazywa się **pielęgnacją** perspektywy. Ze względu na rodzaj dokonywanych zmian można wyróżnić pielęgnację pełną (gdy po dowolnej modyfikacji tabeli bazowej cała perspektywa materializowana jest przeliczana od nowa) oraz przyrostową (gdy po modyfikacji tabeli bazowej uaktualniona zostaje tylko część perspektywy wywiedziona ze zmodyfikowanych danych).

## 2.2. Podstawowe pojęcia z zakresu eksploracji danych

**Zbiory częste**: Niech $L=\{l_1,l_2,...,l_n\}$ będzie zbiorem literałów zwanych *elementami*. Niech D będzie kolekcją transakcji, gdzie każda transakcja T jest dowolnej długości i $\forall T \in D\ T \subseteq L$. Transakcja T *wspiera* element x jeśli $x \in T$. Transakcja T *wspiera* zbiór X, jeśli T wspiera każdy element $x \in X$. **Wsparciem** (ang. *support*) zbioru X nazywamy stosunek liczby transakcji wspierających X do liczby wszystkich transakcji w bazie danych.

$$\sup(X, D) \frac{\left|\{T \in D : T\, wspiera\, X\}\right|}{|D|}$$

Problem odkrywania zbiorów częstych polega na znalezieniu w danej bazie danych D wszystkich zbiorów, których wsparcie jest wyższe niż zdefiniowany przez użytkownika próg minimalnego wsparcia (ang. *minsup*). Zbiór, którego wsparcie jest wyższe niż *minsup* nazywamy **zbiorem częstym** (ang. *frequent itemset*).
**Reguła asocjacyjna** to implikacja postaci $X \rightarrow Y$, gdzie $X \subset L$, $Y \subset L$ i $X \cap Y = \varnothing$. Zbiór X nazywa się *ciałem* reguły a zbiór Y *głową* reguły. Z każdą regułą asocjacyjną związane są dwie miary wyznaczające siłę i statystyczne znaczenie reguły. **Wsparcie** (ang. *support*) reguły $X \rightarrow Y$ w bazie danych D to stosunek liczby transakcji wspierających regułę do liczby wszystkich transakcji. Innymi słowy, reguła $X \rightarrow Y$ ma w bazie danych D wsparcie *s*, jeśli *s*% transakcji w bazie danych wspiera $X \cup Y$.

$$\sup(X \rightarrow Y, D) \frac{\left|\{T \in D : T\, wspiera\, X \cup Y\}\right|}{|D|}$$

W praktyce wsparcie określa siłę reguły asocjacyjnej, czyli ilość transakcji zawierających zbiory częste wchodzące w skład reguły.
**Ufność** (ang. *confidence*) reguły $X \rightarrow Y$ w bazie danych D to stosunek liczby transakcji wspierających regułę do liczby transakcji wspierających ciało reguły. Innymi słowy, reguła $X \rightarrow Y$ ma w bazie danych D ufność *c*, jeśli *c*% transakcji wspierających X wspiera również Y.

$$conf(X \rightarrow Y, D) \frac{\left|\{T \in D : T\, wspiera\, X \cup Y\}\right|}{\left|\{T \in D : T\, wspiera\, X\}\right|}$$

Praktycznie miara ufności określa statystyczne znaczenie reguły, czyli siłę korelacji między zbiorami częstymi tworzącymi regułę.
Problem odkrywania reguł asocjacyjnych polega na znalezieniu w danej bazie danych D wszystkich reguł asocjacyjnych, których wsparcie i ufność są wyższe od zdefiniowanych przez użytkownika progów minimalnego wsparcia (ang. *minsup*) i minimalnej ufności (ang. *minconf*).

## 2.3. Zapytania eksploracyjne

Znajdowanie zbiorów częstych lub reguł asocjacyjnych wykonywane jest w procesie eksploracji na podstawie zapytania eksploracyjnego. W tym raporcie do wyrażania zapytań eksploracyjnych posłużono się językiem MineSQL [9]. MineSQL to deklaratywny język eksploracji danych oparty na języku SQL, którego składnia przypomina składnię języka SQL.

Obecnie język ten pozwala na wyrażanie poleceń służących do odkrywania zbiorów częstych, reguł asocjacyjnych i wzorców sekwencyjnych.
Przykładowe zapytanie eksploracyjne, które w tabeli *Zakupy* znajduje zbiory częste o wsparciu powyżej 30% i zawierające produkty *chleb* i *masło*, ma następującą postać:

```
MINE ITEMSET, SUPPORT(ITEMSET)
FOR PRODUKTY FROM (
      SELECT SET(PRODUKT) AS PRODUKTY
      FROM ZAKUPY
      GROUP BY ID_KLIENTA )
WHERE SUPPORT(ITEMSET) > 0.3
AND ITEMSET CONTAINS TO_SET('CHLEB')
AND ITEMSET CONTAINS TO_SET('MASLO');
```

**Relacje między wynikami zapytań eksploracyjnych**
Między wynikami dwóch zapytań eksploracyjnych $Q_1$ i $Q_2$ mogą zachodzić następujące relacje:

- **równoważność** - zapytania eksploracyjne są równoważne, jeśli dla każdego zbioru danych zwracają ten sam zbiór odkrytych wzorców i dla każdej pary wzorców wartości współczynników statystycznych (np. wsparcia i ufności) są identyczne.
- **zawieranie** - zapytanie eksploracyjne $Q_2$ zawiera zapytanie $Q_1$ jeżeli dla każdego zbioru danych każdy wzorzec odkryty przez zapytanie $Q_1$ jest też odkryty przez $Q_2$ i wartości współczynników statystycznych dla tych samych wzorców są identyczne w obu przypadkach.
- **dominacja** - zapytanie eksploracyjne $Q_2$ dominuje zapytanie $Q_1$ jeżeli dla każdego zbioru danych każdy wzorzec odkryty przez zapytanie $Q_1$ jest też odkryty przez $Q_2$ i wartości współczynników statystycznych wyznaczone przez $Q_2$ są nie mniejsze niż wartości współczynników wyznaczone przez $Q_1$.

Równoważność zapytań eksploracyjnych jest szczególnym przypadkiem relacji zawierania, zaś relacja zawierania jest szczególnym przypadkiem relacji dominacji. Przedstawione relacje mają ogólny charakter i można je stosować do wielu typów wzorców (zbiorów częstych, reguł asocjacyjnych) oraz wielu modeli ograniczeń.
Na podstawie oceny relacji między wynikami dwóch zapytań można udzielić odpowiedzi na zapytanie $Q_1$ korzystając z wyników zapytania $Q_2$. Możliwe są trzy przypadki:

- użytkownik podaje zapytanie eksploracyjne $Q_1$, a istnieją zmaterializowane wyniki zapytania równoważnego $Q_2$. Wykonanie zapytania $Q_1$ jest niepotrzebne, ponieważ oba zapytania mają ten sam wynik.
- użytkownik podaje zapytanie eksploracyjne $Q_1$, a istnieją zmaterializowane wyniki zapytania $Q_2$ które zawiera $Q_1$. Aby udzielić odpowiedzi na $Q_1$ wystarczy odczytać zmaterializowany wynik zapytania $Q_2$ i odrzucić wzorce nie spełniające ograniczeń nałożonych na $Q_1$
- użytkownik podaje zapytanie eksploracyjne $Q_1$, a istnieją zmaterializowane wyniki zapytania $Q_2$ które dominuje $Q_1$. Aby udzielić odpowiedzi na $Q_1$ należy odczytać zmaterializowany wynik zapytania $Q_2$ i odrzucić wzorce nie spełniające ograniczeń nałożonych na $Q_1$, a następnie dokonać pełnego odczytu bazy danych aby określić rzeczywiste wartości współczynników statystycznych wzorców zgodnie z ograniczeniami nałożonymi na $Q_1$.

**Perspektywa eksploracyjna** (ang. *data mining view*) jest definiowana przez zapytanie eksploracyjne i przechowuje wzorce odkrywane w wyniku zapytania eksploracyjnego. Przy każdym odwołaniu do perspektywy eksploracyjnej zapytanie eksploracyjne definiujące perspektywę jest wykonywane od nowa, co przy dużych wolumenach danych jest czasochłonne.
**Materializowana perspektywa eksploracyjna** (ang. *materialized data mining view*) poza definiującym ją zapytaniem eksploracyjnym zawiera też wzorce odkryte przez zapytanie. Wzorce te są składowane w bazie danych. Materializowane perspektywy eksploracyjne są

wykorzystywane podczas eksploracji w celu przyśpieszenia odpowiedzi na zapytanie eksploracyjne do bazy danych.
Definicja materializowanej perspektywy eksploracyjnej składa się z dwóch rodzajów ograniczeń. Ograniczenia bazodanowe są zawarte w klauzuli `WHERE` zapytania `SELECT` i służą do wyboru zbioru eksplorowanych danych. Ograniczenia eksploracyjne są zawarte w klauzuli `WHERE` zapytania `MINE` i służą do zawężenia wyniku eksploracji tylko do wybranych wzorców.
Zbiory częste przechowywane w perspektywie materializowanej mają znacznik czasowy, określający moment ich odkrycia oraz ważności. Z perspektywą związany jest również przedział czasowy **`REFRESH`**, określający czas, po którym następuje automatyczne odświeżenie zawartości perspektywy. Odświeżanie perspektywy może również odbywać się na żądanie użytkownika. Perspektywę eksploracyjną można odświeżać w sposób pełny (wykonanie kosztownego algorytmu odkrywania wzorców) lub przyrostowo (korzystając z jednego z efektywnych algorytmów eksploracji przyrostowej [3,4,11]).

**Optymalizacja zapytań eksploracyjnych za pomocą perspektyw eksploracyjnych**

W wielu przypadkach zawartość materializowanej perspektywy eksploracyjnej może posłużyć do odpowiedzi na zapytanie eksploracyjne, które jest podobne do zapytania definiującego perspektywę. Może się tak stać w przypadku, gdy zapytanie definiujące perspektywę $Q_v$ dominuje lub zawiera dane zapytanie eksploracyjne Q. Wtedy do udzielenia odpowiedzi na zapytanie Q wystarczy odczytać zawartość perspektywy i odfiltrować te wzorce, które nie spełniają warunków sformułowanych w zapytaniu Q.
Analizując relacje, które zachodzą między zbiorami ograniczeń w dwóch zapytaniach, oznaczonych odpowiednio: $Q_v$ (zapytanie definiujące perspektywę eksploracyjną) oraz Q (zapytanie eksploracyjne użytkownika), można wyróżnić cztery sytuacje:

- zapytanie Q **rozszerza ograniczenia bazodanowe** zapytania Qv jeżeli dodaje do ograniczeń bazodanowych zapytania $Q_v$ dodatkowe klauzule `WHERE` lub `HAVING`, dodaje koniunkcję nowego warunku do warunków bazodanowych w klauzulach `WHERE` lub `HAVING` lub usuwa warunek z alternatywy warunków bazodanowych w klauzulach `WHERE` lub `HAVING`.
- zapytanie Q **redukuje ograniczenia bazodanowe** zapytania $Q_v$ jeżeli usuwa z ograniczeń bazodanowych zapytania $Q_v$ klauzule `WHERE` lub `HAVING`, usuwa warunek z koniunkcji warunków bazodanowych w klauzulach `WHERE` lub `HAVING` lub dodaje alternatywę nowego warunku do warunków bazodanowych w klauzulach `WHERE` lub `HAVING`.
- zapytanie Q **rozszerza ograniczenia eksploracyjne** zapytania $Q_v$ jeżeli dodaje do ograniczeń eksploracyjnych zapytania $Q_v$ dodatkowe klauzule `WHERE` lub `HAVING`, dodaje koniunkcję nowego warunku do warunków eksploracyjnych w klauzulach `WHERE` lub `HAVING`, usuwa warunek z alternatywy warunków eksploracyjnych w klauzulach `WHERE` lub `HAVING`, lub zastępuje ograniczenie eksploracyjne występujące w $Q_v$ ograniczeniem bardziej restryktywnym (np. wyższy próg minimalnego wsparcia) .
- zapytanie Q **redukuje ograniczenia eksploracyjne** zapytania $Q_v$ jeżeli redukuje ograniczenia eksploracyjne zapytania $Q_v$ o klauzule `WHERE` lub `HAVING`, usuwa warunek z koniunkcji warunków eksploracyjnych w klauzulach `WHERE` lub `HAVING`, dodaje alternatywę nowego warunku do warunków eksploracyjnych w klauzulach `WHERE` lub `HAVING`, lub zastępuje ograniczenie eksploracyjne występujące w $Q_v$ ograniczeniem mniej restryktywnym (np. niższy próg minimalnego wsparcia).

**Rodzaje eksploracji**

W zależności od relacji między zapytaniem $Q_v$ definiującym perspektywę a zapytaniem Q można wykorzystać jeden z czterech sposobów eksploracji:

1. **Eksploracja pełna** (ang. *full mining*), polega na wykonaniu całego algorytmu odkrywania wzorców, bez wykorzystania zawartości perspektywy. Sytuacja taka występuje wówczas, gdy zapytanie Q rozszerza ograniczenia bazodanowe zapytania $Q_v$.

2. **Eksploracja przyrostowa** (ang. *incremental mining*), polega na wykonaniu algorytmu przyrostowego odkrywania wzorców w rozszerzonym zbiorze danych. Sytuacja taka występuje wówczas, gdy zapytanie Q redukuje ograniczenia bazodanowe zapytania $Q_v$.
3. **Eksploracja uzupełniająca** (ang. *complementary mining*), polega na odkrywaniu wzorców na podstawie wcześniej znalezionych wzorców. Sytuacja taka występuje wówczas, gdy zapytanie Q redukuje ograniczenia eksploracyjne zapytania $Q_v$.
4. **Eksploracja weryfikująca** (ang. *verifying mining*), polega na odfiltrowywaniu tych wzorców przechowywanych w perspektywie, które nie spełniają rozszerzonych ograniczeń eksploracyjnych. Sytuacja taka występuje wówczas, gdy zapytanie Q rozszerza ograniczenia eksploracyjne zapytania $Q_v$.

# 3. Algorytmy eksploracji danych

## 3.1. Klasyczny algorytm *Apriori*

Algorytm *Apriori* [2] jest podstawowym algorytmem odkrywania zbiorów częstych. Zasada działania algorytmu *Apriori* opiera się na tym, że dowolny zbiór jest częsty wtedy i tylko wtedy, gdy wszystkie jego podzbiory są częste. *Apriori* odkrywa zbiory częste na podstawie wcześniej odkrytych zbiorów o mniejszych rozmiarach, redukując znacząco przestrzeń możliwych zbiorów częstych. Niech D oznacza zbiór danych w których następuje eksploracja. $L_k$ oznacza zbiór k-elementowych zbiorów częstych (l-zbiorów, ang. *large-itemsets*) czyli zbiorów posiadających wsparcie wyższe niż *minsup*. $C_k$ oznacza zbiór kandydatów (potencjalnych zbiorów częstych) o rozmiarze k. Algorytm *Apriori* ma następującą postać:

```
1   L1 = {large 1-itemsets};
2   for(k=2; Lk-1≠∅; k++) do
3   Begin
4       Ck = APRIORI_GEN(Lk-1);
5       for all transactions t∈D do
6       Begin
7           Ct = SUBSET(Ck, t);
8           for all candidates c∈Ct do c.sup++;
9       End
10      Lk={c∈Ct | c.sup ≥ minsup}
11  End
12  ANSWER = ∪kLk
```

W pierwszym kroku tworzony jest zbiór $L_1$ zawierający wszystkie jednoelementowe zbiory częste. Zbiór ten jest wyznaczany podczas jednego pełnego odczytu bazy danych. Następnie algorytm iteracyjnie wyznacza zbiory częste. Funkcja **APRIORI_GEN** służy do zbudowania zbiorów kandydujących o rozmiarze k na podstawie odkrytych w poprzedniej iteracji zbiorów częstych o rozmiarze k-1. W każdej iteracji po zbudowaniu zbioru kandydatów $C_k$ następuje weryfikacja tych kandydatów podczas pełnego odczytu bazy danych (funkcja **SUBSET**). Pod koniec każdej iteracji do zbioru $L_k$ wpisywane są te zbiory kandydujące, które podczas weryfikacji uzyskały wsparcie większe niż *minsup*. Algorytm kończy działanie wtedy, gdy w pewnej iteracji wsparcie żadnego zbioru z $C_k$ nie przekracza już progu minimalnego wsparcia lub gdy na podstawie $L_{k-1}$ nie można wygenerować już żadnych zbiorów kandydujących.
Funkcja **APRIORI_GEN** generuje zbiór kandydatów $C_k$ i składa się z dwóch następujących etapów: W etapie łączenia (ang. *join step*) następuje połączenie zbiorów p i $q \in L_{k-1}$, które mają takie same k-2 pierwsze elementy. Suma tych zbiorów zostaje włączona do zbioru $C_k$. Wykorzystuje się tu założenie upraszczające mówiące o tym, że przechowywane dane są posortowane rosnąco. W etapie filtrowania (ang. *prune step*) następuje usunięcie tych zbiorów z $C_k$, których jakikolwiek podzbiór k-1 elementowy nie należy do $L_{k-1}$ (nie jest zbiorem częstym).

Funkcja **SUBSET** wyznacza zbiór tych zbiorów z $C_k$, których wszystkie elementy są zawarte w rozpatrywanej transakcji i zwiększa tym zbiorom wartość wsparcia.

## 3.2. Negatywna granica zbioru

Z algorytmem *Apriori* związane jest pojęcie negatywnej granicy zbioru częstego. Zbiór należy do negatywnej granicy, jeżeli jest zbiorem kandydującym, ale nie spełnia warunku minimalnego wsparcia. Formalnie można to zapisać w następujący sposób:

$$NBd(L_k) = C_k - L_k$$

$$L_k \cup NBd(L_k) = C_k$$

Aby algorytm *Apriori* wyznaczał granicę negatywną należy go zmodyfikować w sposób przedstawiony poniżej.

```
1    L1 = {large 1-itemsets};
2    for(k=2; Lk-1≠∅; k++) do
3    Begin
4         Ck = APRIORI_GEN(Lk-1);
5         for all transactions t∈D do
6         Begin
7              Ct = SUBSET(Ck, t);
8              for all candidates c∈Ct do c.sup++;
9         End
10        Lk={c∈Ct | c.sup ≥ minsup}
10a       NBd(Lk)={c∈Ct | c.sup < minsup}
11   End
12   ANSWER = ∪kLk
12a  NBd(L) = ∪kNBd(Lk)
```

W linii 10a zbiory, które nie spełniają warunku minimalnego wsparcia, czyli zbiory, których wartość wsparcia jest mniejsza niż wartość progu *minsup*, są włączane do granicy negatywnej zbioru $L_k$. W linii 12a konstruowany jest zbiór NBd(L), zawierający wszystkie nieczęste zbiory kandydujące.

## 3.3. Propozycja nowego rozwiązania

Podstawową wadą algorytmów eksploracji danych jest długi czas odpowiedzi na zapytanie eksploracyjne. W przypadku algorytmu Apriori jest to związane z tym, że Apriori potrzebuje k+1 pełnych odczytów bazy danych do wyznaczenia wszystkich k-elementowych zbiorów częstych. Dla dużych wolumenów danych czas działania algorytmu jest nie do zaakceptowania z punktu widzenia użytkownika. Z tego powodu wiele prac poświęcono optymalizacji algorytmów eksploracji danych, które korzystają z wyników wcześniejszych zapytań (materializowanych perspektyw).

Aby zaproponować modyfikacje algorytmu Apriori zaimplementowano go w wersji klasycznej i na podstawie eksperymentów obliczeniowych stwierdzono, że najbardziej czasochłonną funkcją jest funkcja SUBSET, gdyż następuje w niej pełen odczyt bazy danych. Czas przetwarzania rośnie liniowo wraz ze wzrostem rozmiaru wczytywanych danych i w przypadku dużych wolumenów danych staje się zbyt długi, aby proces eksploracji mógł spełniać warunek interaktywności. Czas odpowiedzi na zapytanie można wydatnie skrócić poprzez zastosowanie materializacji wyników. Zaproponowane modyfikacje obejmują:

- zapytania eksploracyjne wydawane dla zbioru danych zmniejszonego względem zawartości perspektywy eksploracyjnej,
- zapytania eksploracyjne wydawane dla zbioru danych rozszerzonego względem zawartości perspektywy eksploracyjnej.

# 4. Opis zaproponowanego rozwiązania

## 4.1. Modyfikacja algorytmu *Apriori* dla zapytania rozszerzającego ograniczenia bazodanowe

Jeżeli użytkownik wprowadza nowe zapytanie, które odkrywa zbiory częste w pewnym fragmencie bazy danych, a zmaterializowano wynik wykonania podobnego zapytania na nadzbiorze tych danych, to użytkownik nie musi wykonywać klasycznego algorytmu *Apriori*, ale może użyć zmodyfikowanej wersji algorytmu, która korzysta z wyników poprzedniej eksploracji (jest to przypadek zapytania, które rozszerza ograniczenia bazodanowe względem zapytania definiującego perspektywę eksploracyjną). Zmodyfikowana wersja algorytmu jest znacznie szybsza, niż wersja podstawowa. Niech **D** oznacza zbiór danych na którym działa algorytm klasyczny, a **D'** zbiór danych, na którym działa zmodyfikowany algorytm *Apriori*. **t** oznacza transakcję należącą do zbioru D, **t'** transakcję należącą do zbioru D' a **Δt** oznacza różnicę między transakcjami (zbiór elementów, które są w transakcji **t**, ale nie ma ich w transakcji **t'**. $\mathbf{L_k}$ to zbiór k-elementowych zbiorów częstych wyznaczonych w wyniku działania algorytmu klasycznego. **L** oznacza zbiór wszystkich zbiorów częstych $L_k$. W końcu $\mathbf{L_k'}$ oznacza zbiór k-elementowych zbiorów częstych wyznaczonych w wyniku działania algorytmu zmodyfikowanego.

Zmodyfikowany algorytm *Apriori* dokonuje jednego odczytu bazy danych, w którym dla każdego zbioru częstego wyznaczonego w wyniku działania algorytmu klasycznego, następuje sprawdzenie, czy tworzące go elementy są zawarte w różnicy transakcji. Zbiory częste zawarte w **Δt** mają wsparcie zmniejszane o 1.

```
1   for all transactions t∈D or t'∈D' do
2   Begin
3       Δt = t - t';
4       for all Lk∈L do
5       Begin
6           for all l∈Lk do
7           Begin
8               if ∃e: e∈l and e∈Δt then l.sup--;
9           End
10      End
11  End
12  Lk' = {l∈Lk | l.sup ≥ minsup}
13  ANSWER = ∪kLk'
```

**Zalety algorytmu zmodyfikowanego.**

Podstawową zaletą zmodyfikowanego algorytmu *Apriori* dla zapytania rozszerzającego ograniczenia bazodanowe jest znaczne skrócenie czasu działania. Algorytm zmodyfikowany potrzebuje tylko jednego odczytu bazy danych do wyznaczenia k-elementowych zbiorów częstych, podczas gdy algorytm klasyczny potrzebuje k odczytów. Zysk na czasie wykonywania zapytania eksploracyjnego jest tym bardziej znaczący, im większy jest rozmiar danych, na których dokonuje się eksploracji.

## 4.2. Modyfikacja algorytmu *Apriori* dla zapytania redukującego ograniczenia bazodanowe

Może się zdarzyć, że użytkownik wprowadzi zapytanie, które odkrywa wzorce w większej części bazy danych niż zapytanie definiujące perspektywę materializowaną. W takiej sytuacji użytkownik nie musi wykonywać klasycznego algorytmu *Apriori*, ale może użyć zmodyfikowanej wersji algorytmu, która korzysta z wyników poprzedniej eksploracji (jest to przypadek zapytania, które redukuje ograniczenia bazodanowe względem zapytania

definiującego perspektywę eksploracyjną). Zmodyfikowana wersja algorytmu jest szybsza, niż wersja podstawowa, ale czas jej działania zależy od różnicy między zapytaniami. Niech **D** oznacza zbiór danych na którym działa algorytm klasyczny, **D'** oznacza zbiór danych, na którym działa zmodyfikowany algorytm *Apriori*. **t** oznacza transakcję należącą do zbioru D, **t'** oznacza transakcję należącą do zbioru D' a **Δt** niech oznacza różnicę między transakcjami (**Δt = t'–t**). $\mathbf{L_k}$ oznacza zbiór k-elementowych zbiorów częstych wyznaczonych w wyniku działania algorytmu klasycznego. **L** oznacza zbiór wszystkich zbiorów częstych $L_k$. Wreszcie $\mathbf{L_k'}$ oznacza zbiór k-elementowych zbiorów częstych wyznaczonych w wyniku działania algorytmu zmodyfikowanego. $\mathbf{NB_k}$ to zbiór k-elementowych zbiorów należących do granicy negatywnej a **NB** to suma wszystkich zbiorów $NB_k$. $\mathbf{LNB_k}$ oznacza zbiór k-elementowych zbiorów należących do $NB_k$, które w wyniku działania algorytmu stały się częste, **LNB** oznacza zbiór wszystkich elementów $LNB_k$. $\mathbf{GL_k}$ to zbiór k-elementowych zbiorów częstych, które zostały wygenerowane na podstawie zbiorów $L_1$ oraz $LNB_{k-1}$. **GL** to zbiór wszystkich zbiorów $GL_k$.

Algorytm korzysta ze zbiorów zawartych w granicy negatywnej NB oraz z wygenerowanych w wyniku działania klasycznego algorytmu *Apriori* zbiorów częstych należących do L. Ponieważ zapytania różnią się miedzy sobą (np. dochodzą elementy pominięte w poprzednim zapytaniu) to niektóre zbiory przechodzą z granicy negatywnej do zbioru LNB, innymi słowy stają się częste. Są to zbiory, które zawierają elementy zawarte w różnicy transakcji wyznaczanych przez oba zapytania. Następnie na podstawie zbioru LNB i zbioru $L_1$ generowany jest zbiór GL, który powstaje z rozszerzenia zbiorów należących do LNB elementami ze zbioru $L_1$. Dla tak wygenerowanych zbiorów należy dokonać dodatkowego odczytu bazy danych i wyznaczyć wartość ich wsparcia. Te zbiory, które spełniają warunek minimalnego wsparcia stają się częste. Wszystkie zbiory częste wyznaczone przez algorytm należą do zbiorów L, LNB oraz GL.

```
1   for all transactions t∈D or t'∈D' do
2   Begin
3         Δt = t' - t;
4         for all Lk∈L do
5         Begin
6               for all l∈Lk do
7               Begin
8                if ∃e: e∈l and e∈Δt and l⊆t' then l.sup++;
9               End
10        End
11        for all NBk∈NB do
12        Begin
13              for all n∈NBk do
14              Begin
15               if ∃e: e∉n and e∈Δt then e.sup++; NB1+={e};
16               if ∃e: e∈n and e∈Δt and n⊆t' then n.sup++;
17              End
18              LNBk = {n∈NBk | n.sup ≥ minsup}
19        End
20  End
21  LNB = ∪kLNBk
22  GL  = GENERATE(LNB, L1);
23  GL += GENERATE(L, LNB1);
24  GL  = SUBSET_NEW(GL);
25  ANSWER = L∪LNB∪GL
```

Funkcja **SUBSET_NEW** wyznacza wartości wsparcia dla potencjalnie częstych zbiorów ze zbioru GL. Jej działanie polega na sprawdzeniu, czy aktualnie rozpatrywany, potencjalnie częsty zbiór jest zawarty w aktualnie rozpatrywanej transakcji (wyznaczonej przez zapytanie redukujące ograniczenia bazodanowe względem poprzednio wykonanego zapytania eksploracyjnego).

```
1    for all transactions t'∈D' do
2    Begin
3         for all GL_k∈GL do
4         Begin
5              for all g∈GL_k do
6              Begin
7                if g⊆t' then g.sup++;
8              End
9              GL_k' = {g∈GL_k | g.sup ≥ minsup}
10        End
11   ANSWER ∪_k GL_k'
```

**Zalety algorytmu zmodyfikowanego.**
Podstawową zaletą zmodyfikowanego algorytmu *Apriori* dla zapytania redukującego ograniczenia bazodanowe jest skrócenie czasu działania. Algorytm zmodyfikowany potrzebuje dwóch odczytów bazy danych do wyznaczenia k-elementowych zbiorów częstych, podczas gdy algorytm klasyczny potrzebuje k odczytów. Zysk na czasie wykonywania zapytania eksploracyjnego jest tym większy, im mniejsza jest różnica miedzy zapytaniami. Jeżeli ta różnica jest niewielka, to niewiele zbiorów z granicy negatywnej stanie się zbiorami częstymi i w związku z tym niewiele zbiorów potencjalnie częstych zostanie wygenerowanych w wyniku działania funkcji GENERATE. W konsekwencji czas działania algorytmu jest krótszy.

# 5. Przeprowadzone testy

## 5.1. Środowisko testowe

Wszystkie testy zostały przeprowadzone na komputerze PC z procesorem Pentium Celeron 333 MHz, wyposażonym w 256 MB pamięci operacyjnej. Dane były przechowywane w bazie danych Oracle w wersji 8.1.7, działającej pod kontrolą systemu operacyjnego Windows 2000 Professional PL.

## 5.2. Test 1 – identyfikacja kosztownych części algorytmu klasycznego

Celem przeprowadzenia Testu 1 jest wskazanie najbardziej kosztownych czasowo elementów klasycznego algorytmu *Apriori*. Test 1 przeprowadzono dla 1000 transakcji i minimalnego wsparcia na poziomie 15 transakcji (*minsup*=15). Czas trwania algorytmu klasycznego wynosił 337,5 sekundy.

| | JOIN [s] | PRUNE [s] | SUBSET [s] | DELETE [s] |
|---|---|---|---|---|
| Budowanie $L_1$ | brak | brak | 8,20 | brak |
| Budowanie $L_2$ | 0,02 | brak | 57,51 | 0,01 |
| Budowanie $L_3$ | 2,73 | 4,93 | 226,87 | 0,01 |
| Budowanie $L_4$ | 0,68 | 0,13 | 27,27 | 0,01 |
| Budowanie $L_5$ | 0,01 | 0,01 | 6,72 | 0,01 |
| Budowanie $L_6$ | 0,01 | 0,01 | 2,35 | 0,01 |
| **Czas sumaryczny** | **3,45** | **5,08** | **328,92** | **0,05** |
| **Procent** | **1,02%** | **1,51%** | **97,46%** | **0,01%** |

**Tabela 1.** Wyniki działania algorytmu klasycznego dla 1000 transakcji i *minsup*=15.

Ten sam test przeprowadzono dla 10000 transakcji i minimalnego wsparcia na poziomie 200 transakcji (*minsup*=200). Czas trwania algorytmu klasycznego wynosił 1640,34 sekundy.

| | JOIN [s] | PRUNE [s] | SUBSET [s] | DELETE [s] |
|---|---|---|---|---|
| Budowanie $L_1$ | brak | brak | 20,42 | brak |
| Budowanie $L_2$ | 0,03 | brak | 465,05 | 0,01 |
| Budowanie $L_3$ | 0,74 | 4,07 | 944,84 | 0,08 |
| Budowanie $L_4$ | 0,55 | 2,92 | 177,11 | 0,01 |
| Budowanie $L_5$ | 0,01 | 3,78 | 20,77 | 0,05 |
| **Czas sumaryczny** | **1,33** | **10,77** | **1628,19** | **0,15** |
| **Procent** | **0,08%** | **0,66%** | **99,26%** | **0,01%** |

**Tabela 2.** Wyniki działania algorytmu klasycznego dla 10000 transakcji i *minsup*=200.

W Tabelach 1 i 2 przedstawiono czasy działania poszczególnych funkcji klasycznego algorytmu Apriori (JOIN, PRUNE, SUBSET, DELETE) dla dwóch instancji:

**$I_1$:** 1000 transakcji i wartości wsparcia minimalnego *minsup* = 15,

**$I_2$:** 10000 transakcji i wartości wsparcia minimalnego *minsup* = 200.

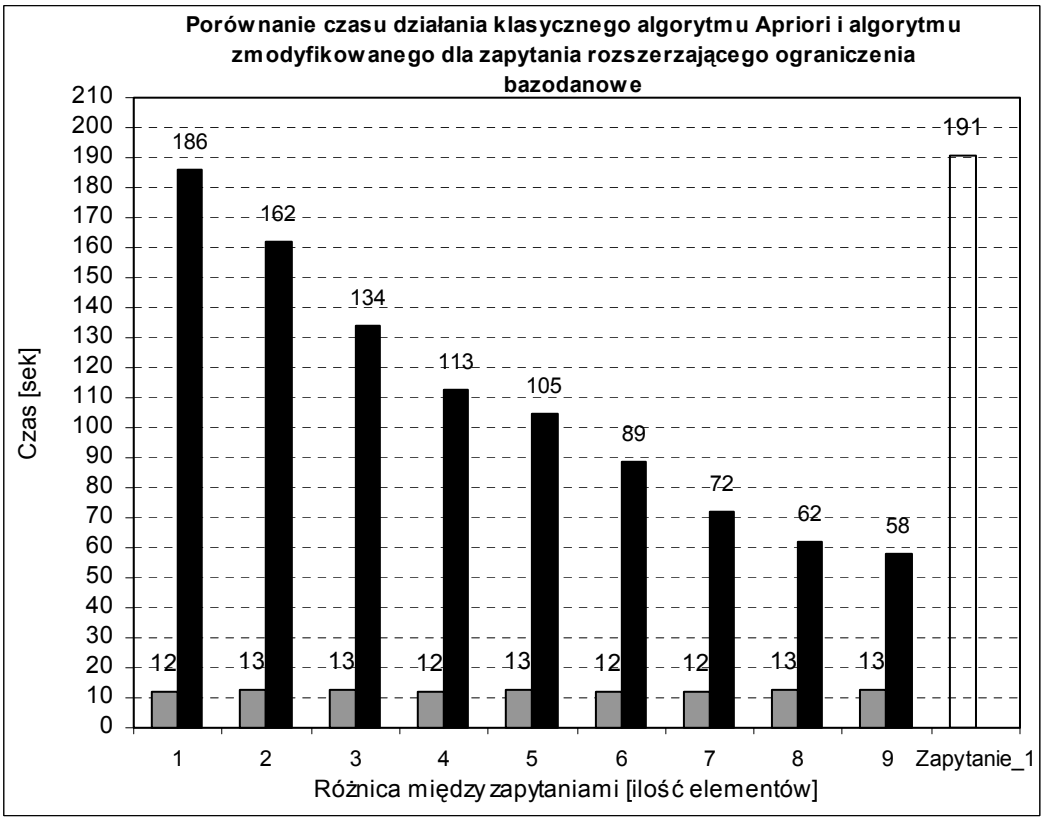
W przypadku instancji $\mathbf{I_1}$ największy zbiór, jaki odkrył algorytm klasyczny to $L_6$, podczas gdy w przypadku instancji $\mathbf{I_2}$ największy zbiór to $L_4$ (wszyscy kandydaci na $L_5$ zostali odrzuceni). Po przeprowadzeniu testu stwierdzono, że najbardziej kosztowną pod względem czasu działania jest funkcja SUBSET, która zajmuje większość czasu działania całego algorytmu.

## 5.3. Test 2 – algorytm zmodyfikowany dla zapytania rozszerzającego ograniczenia bazodanowe

Celem przeprowadzenia Testu 2 jest pokazanie zysku czasowego jaki można osiągnąć stosując algorytm zmodyfikowany. Test 2 pokazuje przyśpieszenie uzyskane podczas zastosowania algorytmu wykorzystującego wyniki wcześniejszej eksploracji dla zapytania rozszerzającego ograniczenia bazodanowe. Test 2 przeprowadzono dla instancji $\mathbf{I_1}$ zwierającej 1000 transakcji i dla wartości minimalnego wsparcia *minsup*=20.

| | Czas działania algorytmu zmodyfikowanego c1 [s] | Czas działania algorytmu klasycznego c2 [s] | Zysk czasowy c2/c1 [%] |
|---|---|---|---|
| ZAPYTANIE 1 | - | 191 | - |
| ZAPYTANIE 2 różnica 1 elementu | 12 | 186 | 1550 |
| ZAPYTANIE 2 różnica 2 elementów | 13 | 162 | 1246 |
| ZAPYTANIE 2 różnica 3 elementów | 13 | 134 | 1031 |
| ZAPYTANIE 2 różnica 4 elementów | 12 | 113 | 942 |
| ZAPYTANIE 2 różnica 5 elementów | 13 | 105 | 808 |
| ZAPYTANIE 2 różnica 6 elementów | 12 | 89 | 742 |
| ZAPYTANIE 2 różnica 7 elementów | 12 | 72 | 600 |
| ZAPYTANIE 2 różnica 8 elementów | 13 | 62 | 477 |
| ZAPYTANIE 2 różnica 9 elementów | 13 | 58 | 446 |

**Tabela 3.** Wyniki działania algorytmu zmodyfikowanego i klasycznego dla 1000 transakcji i *minsup*=20.




**Wykres 1.** Różnica czasu działania algorytmu zmodyfikowanego i klasycznego dla kilku różnic między zapytaniem Z1 i zapytaniem Z2 (1000 transakcji, *minsup*=20).

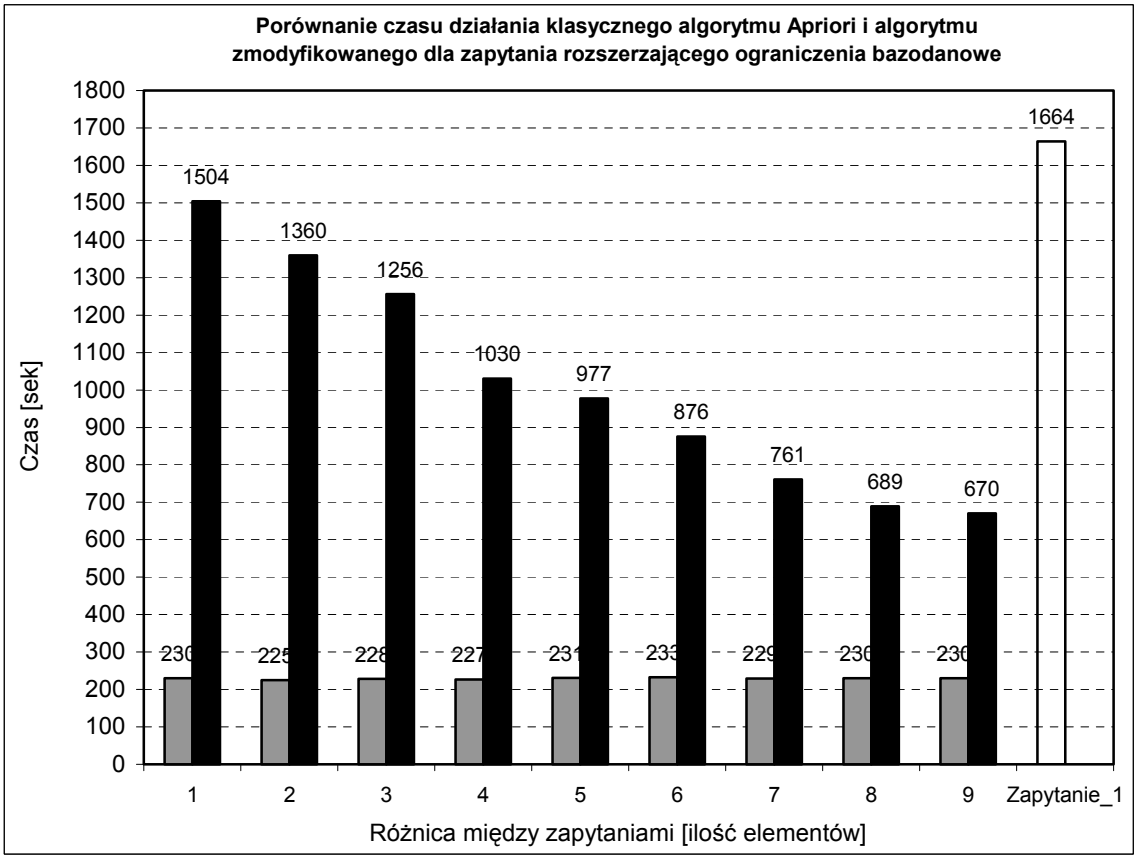
Na Wykresie 1 przedstawiono czasy działania zmodyfikowanego i klasycznego algorytmu Apriori. Oś X wyraża różnicę między Z1 i Z2 podaną jako liczbę elementów, którymi różnią się zbiory danych wyznaczane przez te zapytania. Na wykresie przyjęto następujące oznaczenia:

- **Kolor biały** – oznacza czas wykonywania algorytmu klasycznego dla Z1,
- **Kolor szary** – oznacza czas wykonywania algorytmu zmodyfikowanego dla Z2,
- **Kolor czarny** – oznacza czas wykonywania algorytmu klasycznego dla Z2.

Test 2 przeprowadzono także dla instancji $\mathbf{I_2}$ zawierającej 10000 transakcji i wartości minimalnego wsparcia *minsup*=200.

| | Czas działania algorytmu zmodyfikowanego c1 [s] | Czas działania algorytmu klasycznego c2 [s] | Zysk czasowy c2/c1 [%] |
|---|---|---|---|
| ZAPYTANIE 1 | - | 1664 | - |
| ZAPYTANIE 2 różnica 1 elementu | 230 | 1504 | 654 |
| ZAPYTANIE 2 różnica 2 elementów | 225 | 1360 | 604 |
| ZAPYTANIE 2 różnica 3 elementów | 228 | 1256 | 551 |
| ZAPYTANIE 2 różnica 4 elementów | 227 | 1030 | 454 |
| ZAPYTANIE 2 różnica 5 elementów | 231 | 977 | 423 |
| ZAPYTANIE 2 różnica 6 elementów | 233 | 876 | 376 |
| ZAPYTANIE 2 różnica 7 elementów | 229 | 761 | 332 |
| ZAPYTANIE 2 różnica 8 elementów | 230 | 689 | 300 |
| ZAPYTANIE 2 różnica 9 elementów | 230 | 670 | 291 |

**Tabela 4.** Wyniki działania algorytmu zmodyfikowanego i klasycznego dla 10000 transakcji i *minsup*=200.




**Wykres 2.** Różnica czasu działania algorytmu zmodyfikowanego i klasycznego dla kilku różnic między zapytaniem Z1 i zapytaniem Z2 (10000 transakcji, *minsup*=200).

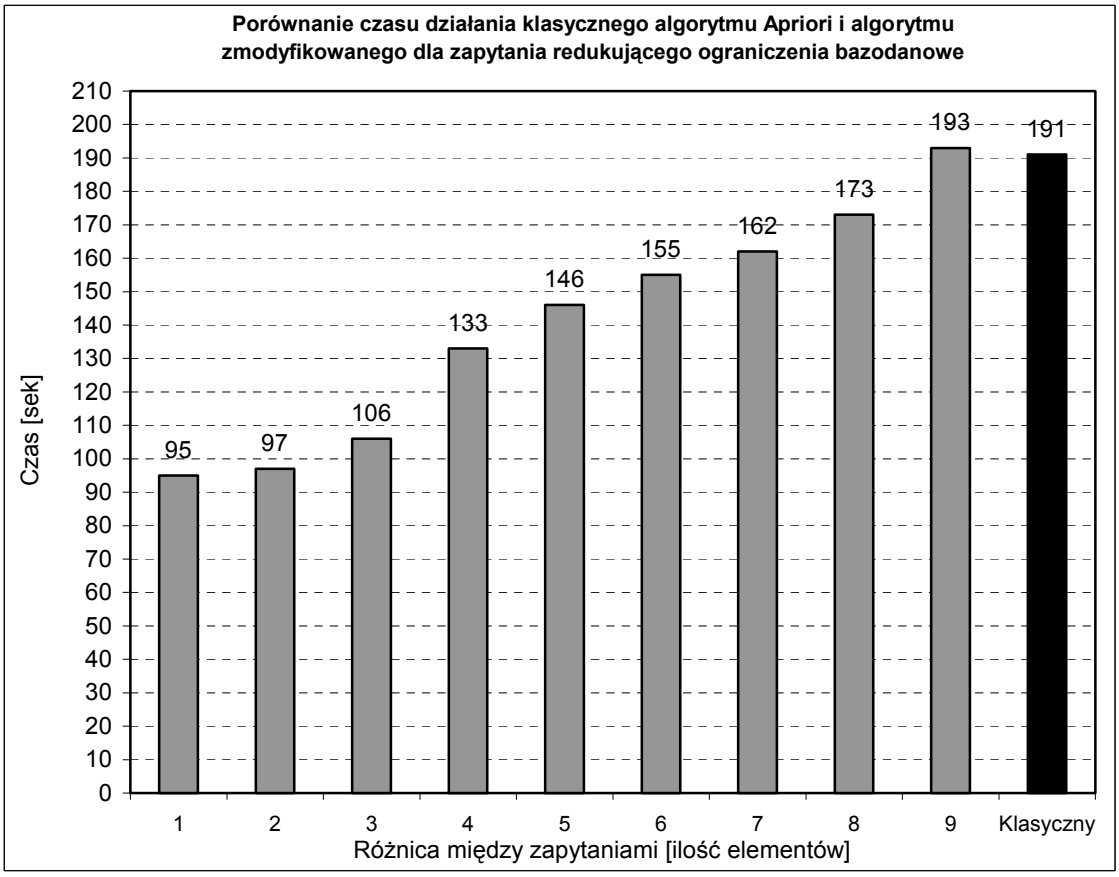
Wykres 2 prezentuje graficzne przedstawienie Tabeli 4, czyli zestawienie czasu działania algorytmu zmodyfikowanego i algorytmu klasycznego. Oznaczenia dotyczące kolorów przyjęto takie same jak w przypadku Wykresu 1.

## 5.4. Test 3 – algorytm zmodyfikowany dla zapytania redukującego ograniczenia bazodanowe

Dokonano również drugiej modyfikacji algorytmu, która korzysta z wyników poprzedniej eksploracji. Jest to modyfikacja dla zapytania, które redukuje ograniczenia bazodanowe poprzedniego zapytania, czyli rozszerza zbiór danych, na którym dokonywana jest eksploracja. Celem przeprowadzenia Testu 3 jest pokazanie zysku czasowego jaki można osiągnąć stosując algorytm zmodyfikowany. Użytkownik zadaje zapytanie Z1, w wyniku czego wykonywany jest klasyczny algorytm *Apriori* i otrzymywane są wyniki w postaci zbiorów częstych. Następnie użytkownik wykonuje zapytanie Z2 redukujące ograniczenia bazodanowe zapytania pierwszego w wyniku czego wykonywany jest zmodyfikowany algorytm *Apriori* dla tego przypadku. Test przeprowadzono dla następującej sytuacji: w wyniku zapytania Z1 (zapytanie to wyznacza zbiór danych pomniejszony o jeden element od zbioru danych wyznaczanego przez zapytanie Z2) wykonany został klasyczny algorytm *Apriori*. Na podstawie zbiorów częstych otrzymanych w wyniku jego działania algorytm zmodyfikowany dokonuje eksploracji na zbiorze danych wyznaczanym przez Z2. Wykonywany jest również klasyczny algorytm *Apriori* na zbiorze danych wyznaczanym przez Z2, w celu porównania czasów działania.
Test ten jest powtarzany dla zapytania Z1 wyznaczającego zbiór danych pomniejszony również o: 2, 3, 4, 5, 6, 7, 8, 9 elementów w stosunku do zbioru wyznaczanego przez Z2 (które jest stałe dla całego testu).
Test 3 przeprowadzono dla instancji $\mathbf{I_1}$ zawierającej 1000 transakcji i dla wartości minimalnego wsparcia *minsup*=20.

| | Czas działania algorytmu zmodyfikowanego korzystającego z wyników ZAPYTANIA 1 c1 [s] | Czas działania algorytmu klasycznego dla ZAPYTANIA 2 c2 [s] | Zysk czasowy c2/c1 [%] |
|---|---|---|---|
| ZAPYTANIE 1 różnica 1 elementu | 95 | 191 | 201 |
| ZAPYTANIE 1 różnica 2 elementów | 97 | | 197 |
| ZAPYTANIE 1 różnica 3 elementów | 106 | | 180 |
| ZAPYTANIE 1 różnica 4 elementów | 133 | | 144 |
| ZAPYTANIE 1 różnica 5 elementów | 146 | | 131 |
| ZAPYTANIE 1 różnica 6 elementów | 155 | | 123 |
| ZAPYTANIE 1 różnica 7 elementów | 162 | | 118 |
| ZAPYTANIE 1 różnica 8 elementów | 173 | | 110 |
| ZAPYTANIE 1 różnica 9 elementów | 193 | | 99 |

**Tabela 5.** Wyniki działania algorytmu zmodyfikowanego i klasycznego dla 1000 transakcji i *minsup*=20.




**Wykres 3.** Różnica czasu działania algorytmu zmodyfikowanego i klasycznego dla kilku różnic między zapytaniem Z1 i zapytaniem Z2 (1000 transakcji, *minsup*=20).

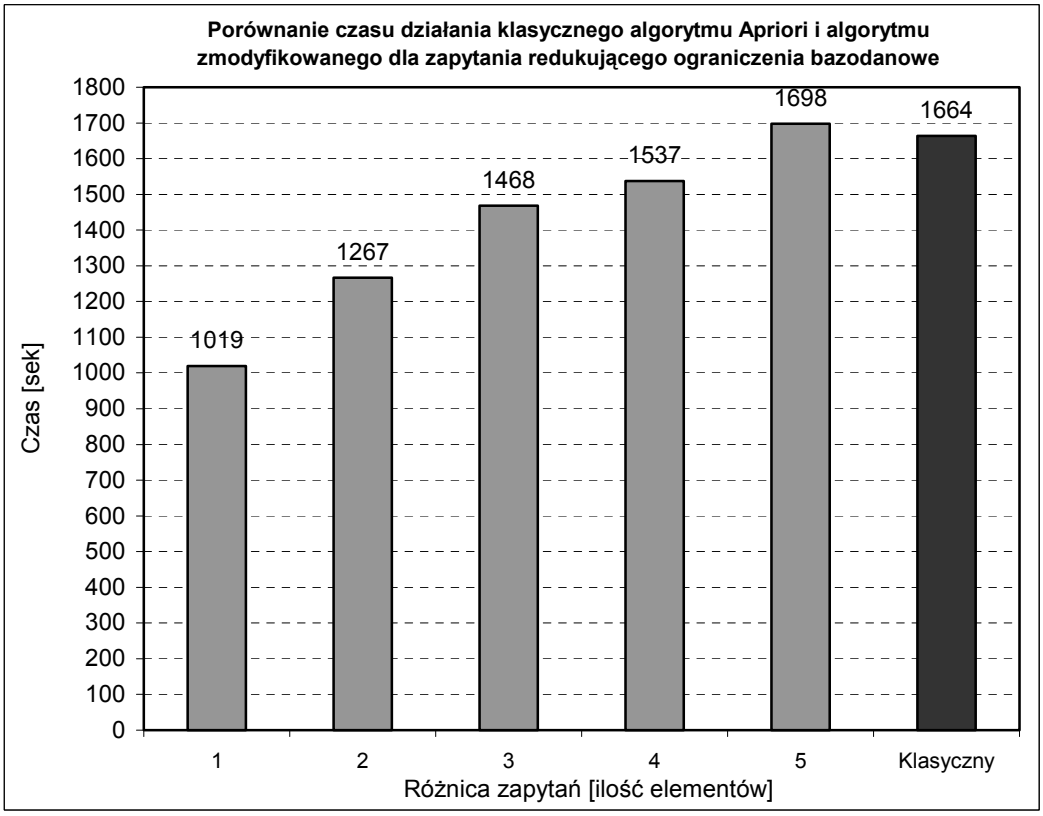
Na Wykresie 3 podano czasy działania zmodyfikowanego i klasycznego algorytmu Apriori. Oś X wyraża różnicę między zapytaniem Z1 i zapytaniem Z2 podaną jako liczbę elementów, którymi różnią się zbiory danych wyznaczanych przez te zapytania. Na wykresie przyjęto następujące oznaczenia:

- **Kolor szary** – oznacza czas wykonywania algorytmu zmodyfikowanego dla Z2, korzystając z wyników Z1,
- **Kolor czarny** – oznacza czas wykonywania algorytmu klasycznego dla Z2.

Test 3 przeprowadzono także dla instancji $\mathbf{I_2}$ zawierającej 10000 transakcji i wartości minimalnego wsparcia *minsup*=200.

| | Czas działania algorytmu zmodyfikowanego korzystającego z wyników ZAPYTANIA 1 c1 [s] | Czas działania algorytmu klasycznego dla ZAPYTANIA 2 c2 [s] | Zysk czasowy c2/c1 [%] |
|---|---|---|---|
| ZAPYTANIE 1 różnica 1 elementu | 1019 | 1664 | 163 |
| ZAPYTANIE 1 różnica 2 elementów | 1267 | | 131 |
| ZAPYTANIE 1 różnica 3 elementów | 1468 | | 113 |
| ZAPYTANIE 1 różnica 4 elementów | 1537 | | 108 |
| ZAPYTANIE 1 różnica 5 elementów | 1698 | | 98 |

**Tabela 6.** Wyniki działania algorytmu zmodyfikowanego i klasycznego dla 10000 transakcji i *minsup*=200.




**Wykres 4.** Różnica czasu działania algorytmu zmodyfikowanego i klasycznego dla kilku różnic między zapytaniem Z1 i zapytaniem Z2 (10000 transakcji, *minsup*=200).

Na Wykresie 4 podano czasy działania zmodyfikowanego i klasycznego algorytmu *Apriori*. Oś X wyraża różnicę między zapytaniem Z1 i zapytaniem Z2 podaną jako liczbę elementów, którymi różnią się zbiory danych wyznaczanych przez te zapytania. Na wykresie przyjęto oznaczenia kolorów takie same, jak w przypadku Wykresu 3.

# 6. Wnioski i spostrzeżenia

## 6.1. Test 1

Test 1 przeprowadzono dla dwóch instancji zbiorów danych, dla których zbadano czasy działania poszczególnych funkcji algorytmu *Apriori*.

Spostrzeżenia:

- dla instancji $\mathbf{I_1}$ zawierającej 1000 transakcji przyjęto poziom wsparcia *minsup*=15. Algorytm dokonał eksploracji i wyznaczył zbiory częste (największy 6-elementowy). Algorytm potrzebował 6 odczytów bazy danych w celu określenia wsparcia zbiorów kandydujących. Łączny czas przeznaczony na działanie funkcji SUBSET stanowi ponad 97% czasu przeznaczonego na działanie całego algorytmu, wynoszącego 337,5 sekundy.
- dla instancji $\mathbf{I_2}$ zawierającej 10000 transakcji przyjęto natomiast poziom wsparcia *minsup*=200. W wyniku wykonania algorytmu *Apriori* znaleziono zbiory częste (największy 4-elementowy). Algorytm potrzebował 5 odczytów bazy danych; w piątym odczycie bazy danych odrzucono wszystkie zbiory kandydujące. Łączny czas działania funkcji SUBSET stanowi 99% czasu przeznaczonego na działanie algorytmu, który w tym przypadku wynosi 1640,34 sekundy.

Wnioski:

- funkcja SUBSET jest najbardziej kosztowną czasowo częścią algorytmu *Apriori*.
- wraz ze wzrostem ilości transakcji rośnie czas działania algorytmu. Jest to spowodowane tym, że algorytm musi wyznaczyć wartość wsparcia na podstawie większej ilości transakcji. W przypadku 100-krotnego wzrostu ilości transakcji czas działania algorytmu wzrósł 50-krotnie.
- czas działania algorytmu zależy również od przyjętej wartości parametru *minsup*. Im mniejsza wartość tego parametru, tym dłuższy czas działania. Powodem takiego zachowania algorytmu jest to, że przyjęcie mniejszej wartości minimalnego wsparcia powoduje, że należy sprawdzić większą liczbę kandydatów w funkcji SUBSET, więcej kandydatów staje się zbiorami częstymi i w wyniku powstają zbiory częste o większej liczności (większej liczbie elementów). To natomiast powoduje, że funkcja SUBSET jest wywoływana większą ilość razy.
- propozycją modyfikacji klasycznego algorytmu *Apriori* jest ograniczenie liczby wywołań funkcji SUBSET. Warunkiem wykorzystania algorytmu zmodyfikowanego jest istnienie wyników poprzedniej eksploracji (zbiorów częstych i zbiorów należących do granicy negatywnej).

## 6.2. Test 2

Na podstawie obserwacji dokonanych w Teście 1 uznano, że modyfikacja algorytmu *Apriori* powinna polegać na zminimalizowaniu wywołań funkcji SUBSET. Zaimplementowano więc modyfikację algorytmu dla przypadku, w którym użytkownik dokonał eksploracji (zapytanie Z1) i chce ponownie dokonać eksploracji, na mniejszym zbiorze danych (zapytanie Z2). Może on wówczas skorzystać ze zmodyfikowanego algorytmu, który w jednym odczycie bazy danych wyznacza wartość wsparcia zbiorów, zawierających elementy będące częścią różnicy wyznaczonej przez oba zapytania.

Spostrzeżenia:

- dla instancji $\mathbf{I_1}$ zawierającej 1000 transakcji przyjęto poziom wsparcia *minsup*=20. Algorytm klasyczny, na danych określonych przez Z1, dokonał eksploracji i odkrył zbiory częste (największy 4-elementowy). Algorytm potrzebował 4 odczytów bazy danych w celu określenia wsparcia zbiorów kandydujących, a czas jego trwania wyniósł 191 sekund.

Następnie uruchomiono algorytm zmodyfikowany, który znajdował zbiory częste na podstawie wyników poprzedniej eksploracji. Algorytm zmodyfikowany potrzebował tylko jednego odczytu bazy danych w celu wyznaczenia zbiorów częstych na danych wyznaczanych przez Z2, a czas jego trwania wyniósł 12 sekund. Na końcu wykonano pomiar czasu działania klasycznego algorytmu Apriori na danych określonych przez Z2. Czas ten wyniósł 186 sekund.

- dla instancji $\mathbf{I_2}$ zawierającej 10000 transakcji przyjęto poziom wsparcia *minsup*=200. Wykonano ponownie Test 2 i otrzymano następujące wyniki:
  - czas algorytmu klasycznego na danych wyznaczanych przez Z1 to 1664 sekundy.
  - czas algorytmu klasycznego na danych wyznaczanych przez Z2 to 1504 sekundy.
  - czas algorytmu zmodyfikowanego na danych wyznaczanych przez Z2 to 230 sekund.
- wraz ze wzrostem różnicy między danymi wyznaczanymi przez Z1 oraz Z2 czas działania klasycznego algorytmu (na danych wyznaczanych przez Z2) malał. Jest to związane z tym, że rozmiar danych, na których dokonywano eksploracji zmniejszał się, co powodowało, że algorytm wyznaczał mniej zbiorów częstych (funkcja SUBSET sprawdzała mniej kandydatów).
- z Wykresów 1 i 2 wynika, że różnica między danymi wyznaczanymi przez Z1 i danymi wyznaczanymi przez Z2 nie ma wpływu na czas działania algorytmu zmodyfikowanego, który w każdym przypadku był bliski wartości 12 sekund. Jest to związane z tym, że algorytm zawsze potrzebuje tylko jednego odczytu bazy danych w celu wyznaczenia zbiorów częstych.

Wnioski:

- jeżeli zapytanie eksploracyjne rozszerza ograniczenia bazodanowe poprzedniego zapytania eksploracyjnego, to należy użyć zmodyfikowanego algorytmu *Apriori* (dla tego przypadku). Użycie algorytmu zmodyfikowanego wielokrotnie przyspiesza czas odpowiedzi na zapytanie eksploracyjne. W przypadku, gdy zapytania różnią się tylko 1 elementem zysk czasowy jest bardzo duży. Algorytm zmodyfikowany jest 15-razy szybszy od klasycznego
- wraz ze wzrostem różnicy między zapytaniami maleje czas wykonywania algorytmu klasycznego, natomiast czas działania algorytmu zmodyfikowanego pozostaje stały. Czasy działania algorytmu klasycznego i zmodyfikowanego zrównają się, gdy algorytm klasyczny będzie potrzebował tylko jednego odczytu bazy danych w celu wyznaczenia zbiorów częstych (gdy w danych będzie istniał tylko jeden 1-elementowy zbiór częsty). W praktyce, takie dane są przypadkiem bardzo rzadkim.
- różnica między czasem wykonania algorytmu klasycznego i algorytmu zmodyfikowanego jest tym większa, im więcej zbiorów o dużej liczności istnieje w danych, w których algorytmy te dokonują eksploracji. Jeżeli w danych można przy ustalonym poziomie wsparcia wyznaczyć zbiory aż do $L_8$, to zysk czasowy płynący z wykorzystania algorytmu zmodyfikowanego jest znaczny.

## 6.3. Test 3

Po przeprowadzeniu Testu 1 stwierdzono, że należy minimalizować liczbę wywołań funkcji SUBSET. Zaimplementowano modyfikację algorytmu dla przypadku, w którym użytkownik dokonał eksploracji (zapytanie Z1) i chce ponownie dokonać eksploracji, na większym zbiorze danych (zapytanie Z2). Użytkownik może wówczas skorzystać z algorytmu zmodyfikowanego, który wyznacza nowe zbiory częste w dwóch odczytach bazy danych.

Spostrzeżenia:

- dla instancji $\mathbf{I_1}$ zawierającej 1000 transakcji przyjęto poziom wsparcia *minsup*=20. Algorytm klasyczny, na danych wyznaczonych przez Z1, dokonał eksploracji i wyznaczył zbiory częste, z których największe były 4-elementowe. Czas trwania algorytmu wyniósł 191 sekund. Następnie uruchomiono algorytm zmodyfikowany, który znajdował zbiory częste na podstawie wyników poprzedniej eksploracji. Algorytm zmodyfikowany potrzebował dwóch odczytów bazy danych w celu znalezienia zbiorów częstych w danych wyznaczanych przez Z2 (różniące się od Z1 tylko jednym elementem). Czas trwania

algorytmu wyniósł 95 sekund. Test ten powtórzono dla różnych wersji Z2, które różniły się od Z1 kolejno: 2, 3, 4, 5, 6, 7, 8, 9 elementami. Wraz ze wzrostem różnicy wzrastał także czas działania algorytmu zmodyfikowanego.

- dla instancji $\mathbf{I_2}$ zawierającej 10000 transakcji przyjęto poziom wsparcia *minsup*=200. Wykonano ponownie Test 3 i otrzymano następujące wyniki:
  - czas algorytmu klasycznego na danych wyznaczanych przez Z1 to 1664 sekundy.
  - czas algorytmu zmodyfikowanego na danych z Z2 to 1019 sekund.
- wraz ze wzrostem różnicy między danymi wyznaczanymi przez Z1 oraz Z2 czas działania (na danych wyznaczanych przez Z2) algorytmu zmodyfikowanego wzrasta. Jest to spowodowane tym, że w wyniku zwiększania się różnicy między zapytaniami algorytm zmodyfikowany musi wygenerować większą liczbę nowych zbiorów potencjalnie częstych, którym następnie w funkcji SUBSET_NEW algorytm oblicza wartości wsparcia. Im więcej nowych, potencjalnie częstych kandydatów, tym dłużej trwa wykonywanie algorytmu.
- różnica między zapytaniami może stać się na tyle duża (w przypadku instancji $\mathbf{I_1}$ jest to różnica 9 elementów, a w przypadku instancji $\mathbf{I_2}$ różnica 5 elementów), że używanie algorytmu zmodyfikowanego staje się nieopłacalne – czas działania algorytmu zmodyfikowanego przewyższa czas działania algorytmu klasycznego.

Wnioski:

- jeżeli zapytanie eksploracyjne redukuje ograniczenia bazodanowe poprzedniego zapytania eksploracyjnego, to należy użyć zmodyfikowanego algorytmu *Apriori* dla tego przypadku. Użycie algorytmu zmodyfikowanego zmniejsza czas odpowiedzi na zapytanie eksploracyjne.
- wraz ze wzrostem różnicy między zapytaniami wzrasta czas działania algorytmu zmodyfikowanego. Czasy działania algorytmu klasycznego i zmodyfikowanego zrównają się w dwóch przypadkach.
  **Przypadek 1:** Gdy algorytm klasyczny będzie potrzebował dwóch odczytów bazy danych w celu wyznaczenia zbiorów częstych (gdy w danych będą istniały tylko 1-elementowe i 2-elementowe zbiory częste). W praktyce taki przypadek jest rzadki.
  **Przypadek 2:** Gdy różnica między zapytaniami stanie się na tyle duża, że zysk czasowy wynikający z mniejszej liczby wywołań funkcji SUBSET zostanie zrównoważony ilością zbiorów potencjalnie częstych, wygenerowanych przez algorytm zmodyfikowany, które funkcja SUBSET musi sprawdzić. Jak pokazał Test 3 w przypadku instancji $\mathbf{I_1}$ jest to różnica 9 elementów, w przypadku instancji $\mathbf{I_2}$ różnica 5 elementów. Ważne jest zatem, aby następujące po sobie zapytania eksploracyjne niewiele się od siebie różniły.
- różnica między czasem wykonania algorytmu klasycznego i algorytmu zmodyfikowanego jest większa wtedy, gdy w danych podlegających eksploracji istnieje większa liczba zbiorów o dużej liczności. Test 3 wykazał, że jeżeli w danych istnieją zbiory 4-elementowe i zapytania różnią się tylko jednym elementem to czas działania algorytmu zmodyfikowanego jest ponad dwukrotnie krótszy niż czas działania algorytmu klasycznego.

## 6.4. Podsumowanie i kierunek przyszłych badań

Czas procesu eksploracji dla dużych wolumenów danych jest bardzo długi. W związku z tym wymaganie dotyczące interaktywności i iteracyjności tego procesu nie jest spełnione. Czas eksploracji może zostać wydatnie skrócony poprzez zastosowanie w kolejnych eksploracjach zmaterializowanych wyników poprzednich eksploracji. Klasyczny algorytm *Apriori*, służący do znajdowania zbiorów częstych potrzebuje k odczytów bazy danych do wyznaczenia k-elementowych zbiorów częstych. Można to zmienić stosując zmodyfikowane algorytmy *Apriori* dla dwóch sytuacji:

- eksploracja dokonuje się na zbiorze danych zmniejszonym w stosunku do zbioru danych wykorzystywanego w poprzedniej eksploracji (zapytanie rozszerza ograniczenia bazodanowe). Algorytm zmodyfikowany potrzebuje tylko jednego odczytu bazy danych.

- eksploracja dokonuje się na zbiorze danych zwiększonym w stosunku do zbioru danych wykorzystywanego w poprzedniej eksploracji (zapytanie redukuje ograniczenia bazodanowe). Algorytm zmodyfikowany potrzebuje tylko dwóch odczytów bazy danych.

Zastosowanie algorytmów zmodyfikowanych znacznie skraca czas przetwarzania zapytania eksploracyjnego.

W przyszłości należałoby rozwinąć przeprowadzone w pracy badania o następujące elementy:

- propozycję funkcji kosztu określającej podobieństwo następujących po sobie zapytań eksploracyjnych oraz implementację systemu, który parze zapytań przypisuje wartość funkcji kosztu,
- propozycję technik pielęgnacji materializowanych perspektyw eksploracyjnych oraz implementację realizującego to zadanie oprogramowania.

W przyszłości należałoby również przeprowadzić eksperymenty naukowe dla dwóch kolejnych przypadków relacji między zapytaniami:

- w wyniku eksploracji użytkownik uzyskuje zbiór rozwiązań zmniejszony w stosunku do zbioru rozwiązań otrzymanego w poprzedniej eksploracji (zapytanie rozszerza ograniczenia eksploracyjne),
- w wyniku eksploracji użytkownik uzyskuje zbiór rozwiązań zwiększony w stosunku do zbioru rozwiązań otrzymanego w poprzedniej eksploracji (zapytanie redukuje ograniczenia eksploracyjne).

Skonstruowanie algorytmów zmodyfikowanych dla tych przypadków może również wydatnie skrócić czas procesu eksploracji.

# 7. Bibliografia


1. Agrawal, R., Imielinski, T., Swami, A., Mining association rules between sets of items in large databases. In Proc. of the ACM SIGMOD International Conference on Management of Data, Washington, USA, May 1993
2. Agrawal, R., Srikant, R., Fast Algorithms for Mining Association Rules. In Proc. of the 20th International Conference on Very Large Data Bases (VLDB'94), Santiago, Chile, 1994.
3. Cheung, D.W., Han, J., Ng, V., Wong, C.Y., Maintenance of discovered association rules in large databases: An incremental updating technique. In Proc. of the 12th International Conference on Data Engineering (ICDE'96), New Orleans, USA, February 1996
4. Cheung, D. W., Lee, S. D. and Kao, B., A General Incremental Technique for Maintaining Discovered Association Rules In Proc. of the 5th International Conference on Database Systems for Advanced Applications (DASFAA'97), Melbourne, Australia, April 1997
5. Gupta, A., Mumick, I.S., Maintenance of Materialized Views: Problems, Techniques, and Applications. IEEE Data Engineering Bulletin, Special Issue on Materilaized Views and Data Warehousing, 18(2), June 1995
6. Gupta, A., Mumick, I.S., Materialized Views: Techniques, Implementations, and Applications, The MIT Press, 1999
7. Imielinski, T., Mannila, H., A Database Perspective on Knowledge Discovery. Communications of the ACM, Vol.39, No.11, 1996
8. Morzy, M., Wojciechowski, M., Mechanizm perspektyw materializowanych w eksploracji danych, XIV Górska Szkoła Polskiego Towarzystwa Informatycznego Szczyrk 2002, pp.93-108 (Vol.1), WNT
9. Morzy, T., Zakrzewicz, M., SQL-like Language for Database Mining. In Proc. of the 1st East European Symposium on Advances in Databases and Information Systems (ADBIS'97), St-Petersburg, Russia, September 1997
10. Nag, B., Deshpande, P., DeWitt, D.J., Using a Knowledge Cache for Interactive Discovery of Association Rules. In Proc. of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99), San Diego, USA, August 1999
11. Thomas, S., Bodagala, S., Alsabti, K., Ranka, S., An Ecient Algorithm for the Incremental Updation of Association Rules in Large Databases. In Proc. of the 3rd ACM SIGKDD

International Conference on Knowledge Discovery and Data Mining (KDD'97), Newport Beach, USA, August 1997

12. Toivonen, H., Sampling large databases for association rules. In Proc. of the 22th International Conference on Very Large Data Bases (VLDB'96), Bombay, India, September 1996
13. Wojciechowski, M., Zakrzewicz, M., Itemset Materializing for Fast Mining of Association Rules. In Proc. of the 2nd East European Conference on Advances in Databases and Information Systems (ADBIS'98), Poznań, Poland, September 1998