Odkrywanie reprezentacji generatorowej wzorców częstych z wykorzystaniem struktur listowych

Marzena Kryszkiewicz i Piotr Pielasa
Instytut Informatyki, Politechnika Warszawska
Nowowiejska 15/19, 00-665 Warszawa, Polska
mkr@ii.pw.edu.pl, P.Pielasa@elka.pw.edu.pl

Abstrakt. Jednym z głównym zadań eksploracji danych jest odkrywanie częstych wzorców. Zwykle liczba wzorców częstych jest olbrzymia. Stąd wynika potrzeba stosowania zwięzłych reprezentacji takich wzorców. Najwcześniej zaproponowanymi, a obecnie najczęściej wykorzystywanymi reprezentacjami wzorców częstych, są: reprezentacja oparta na częstych zbiorach zamkniętych i reprezentacja generatorowa. W celu odkrywania częstych zbiorów zamkniętych zaproponowano w literaturze szereg algorytmów wydobywających je bezpośrednio z bazy danych oraz algorytmów wykorzystujących struktury listowe stworzone na podstawie bazy. Dostępne w literaturze algorytmy odkrywają reprezentację generatorową bezpośrednio z bazy danych. W niniejszym raporcie proponujemy algorytm *GRM* odkrywania reprezentacji generatorowej z wykorzystaniem struktur listowych.

1 Wprowadzenie

Jednym z głównych zadań eksploracji danych jest odkrywanie wzorców często występujących w danych [1]. Zbiory częste są zwykle wykorzystywane do pozyskiwania silnych reguł asocjacyjnych. Stosuje się je także do budowy klasyfikatorów oraz w grupowaniu danych. Zwykle liczba zbiorów częstych jest olbrzymia, co uniemożliwia ich efektywne wykorzystanie. Stąd wynikła potrzeba opracowania zwięzłych reprezentacji takich wzorców, które umożliwiają wyprowadzanie wszystkich zbiorów częstych. Najwcześniej zaproponowanymi reprezentacjami wzorców częstych są: reprezentacja oparta na częstych zbiorach zamkniętych [8, 17] i reprezentacja generatorowa [10]. Pomimo opracowania także znacznie zwięźlejszych reprezentacji dysjunkcyjnych [5, 6-7, 13-16], częste zbiory zamknięte i reprezentacja generatorowa są najpowszechniej wykorzystywane, m. in. do budowy reprezentacji silnych reguł asocjacyjnych [4, 9, 11-13, 19].

W celu odkrywania częstych zbiorów zamkniętych zaproponowano w literaturze szereg algorytmów wydobywających je bezpośrednio z bazy danych (np. [2-3]) oraz algorytmów wykorzystujących struktury listowe stworzone na podstawie bazy (np. [20]). Dostępne są także algorytmy odkrywania częstych zamkniętych zbiorów zarówno z bazy danych (np. [17]), jak i wykorzystujące struktury listowe (np. [21]). Jednakże dostępne w literaturze algorytmy odkrywają reprezentację generatorową wyłącznie bezpośrednio z bazy danych [13]. W niniejszym raporcie oferujemy alternatywny algorytm *GRM* odkrywania reprezentacji generatorowej z wykorzystaniem struktur listowych.

Raport ma następujący układ: W rozdziale 2 przedstawiamy podstawowe pojęcia oraz podajemy ich własności: definiujemy zbiory częste i rzadkie, generatory, reprezentację generatorową i sposób wyprowadzania wsparć zbiorów przy jej użyciu oraz określanie wsparć zbiorów na podstawie struktur listowych. W rozdziale 3 definiujemy drzewo zbiorów i omawiamy jego własności. W rozdziale 4 omawiamy związek pomiędzy reprezentacją generatorową a drzewem zbiorów. Wnioski wyprowadzone w rodziałach 3 i 4 wykorzystamy do skonstruowania algorytmu *GRM* służącego odkrywaniu reprezentacji generatorowej. Algorytm ten zostanie zaprezentowany i zilustrowany przykładem w rozdziale 5. Omówimy tu także możliwość zastosowania w algorytmie różnicowych struktur listowych [20]. W rozdziale 6 prezentujemy rezultaty przeprowadzonych eksperymentów. Uzyskane wyniki podsumujemy w rozdziale 7.

2 Pojęcia podstawowe

2.1 Częste i rzadkie zbiory pozycji

Niech $I = \{i_1, i_2, ..., i_m\}$, $I \neq \emptyset$, będzie zbiorem różnych literałów, które będziemy nazywać *pozycjami* (ang. *items*). W przypadku transakcyjnej bazy danych pojęcie pozycji odpowiada sprzedawanemu produktowi, a w przypadku relacyjnej bazy danych - parze (*atrybut*, *wartość*).

Niech $\mathcal{D}$ będzie zbiorem transakcji (lub krotek), gdzie każda transakcja (krotka) jest podzbiorem zbioru I. Bez utraty ogólności ograniczymy dalsze rozważania do baz transakcyjnych. *Wsparcie* (ang. *support*) zbioru X (oznaczane przez $sup(X)$) definiujemy jako liczbę (lub procent) transakcji w $\mathcal{D}$, które zawierają X. Notacja $X_{[n]}$ będzie oznaczać, że X jest wspierany przez n transakcji. X będziemy nazywać *zbiorem częstym* (ang. *frequent*), jeśli jego wsparcie jest większe od pewnej, zadanej przez użytkownika, wartości progowej oznaczanej przez *minSup*. W przeciwnym przypadku, zbiór będzie nazywany *rzadkim*. Rodzina wszystkich zbiorów częstych będzie oznaczana przez $\mathcal{F}$:

$$\mathcal{F} = \{X \subseteq I | \; sup(X) > minSup\}.$$

Własność 2.1.1 [2-3]. Wsparcie zbioru pozycji nie jest większe od wsparć jego podzbiorów.

Konsekwencją tej własności jest kolejna własność zbiorów częstych i rzadkich.

Własność 2.1.2 [2-3].
a) Wszystkie podzbiory zbioru częstego są zbiorami częstymi.
b) Wszystkie nadzbiory zbioru rzadkiego są zbiorami rzadkimi.

2.2 Generatory

Zbiór X definiujemy jako *generator* wtedy i tylko wtedy, gdy wsparcia wszystkich jego podzbiorów właściwych są różne (czyli większe) od wsparcia zbioru X.[1] Zbiór wszystkich generatorów będziemy oznaczać przez $\mathcal{G}$.

Własność 2.2.1 (generatorów) [10, 13].
a) $\emptyset$ jest generatorem.
b) Zbiór X jest generatorem wtedy i tylko wtedy, gdy wsparcia wszystkich jego podzbiorów krótszych od X o 1 pozycję są różne od wsparcia zbioru X.
c) Wszystkie podzbiory generatora są generatorami.
d) Wszystkie nadzbiory zbioru nie będącego generatorem nie są generatorami.

Niech X będzie zbiorem nie będącym generatorem. Zatem X ma pewien podzbiór właściwy o wsparciu równym wsparciu zbioru X. Ponieważ wsparcie zbioru X nie może przekraczać wsparcia żadnego z jego podzbiorów, a jednocześnie jest równe wsparciu pewnego z jego podzbiorów właściwych wnioskujemy, że $sup(X) = \min(\{sup(Y) | \; Y \subseteq X\})$. Podobnie możemy wnioskować o wsparciach wszystkich podzbiorów właściwych zbioru X, które nie są generatorami. Minimalne z takich podzbiorów będą miały wszystkie podzbiory właściwe w $\mathcal{G}$. Jeśli będą znane wsparcia generatorów, to będziemy w stanie wyznaczyć wsparcia takich minimalnych podzbiorów zbioru X. Wiedza o wyprowadzonych wsparciach tych zbiorów i wsparciach generatorów będzie wystarczająca do wyprowadzenia wsparć kolejnych podzbiorów zbioru X, które nie są generatorami itd. Ostatecznie, po pewnej liczbie iteracji uda nam się wyprowadzić wsparcia wszystkich podzbiorów właściwych zbioru X, jak również samego zbioru X. Informację o wsparciu dowolnego zbioru pozycji można jednakże wyprowadzić bezpośrednio ze wsparć generatorów bez wyprowadzania wsparć zbiorów pośrednich.

[1] Oryginalną definicję generatora można znaleźć np. w [13]. Tam również wykazano, że użyta w niniejszym raporcie definicja generatora jest równoważna oryginalnej.

Własność 2.2.2 (reprezentatywność generatorów) [10, 13]. Niech $X \subseteq I$.

$$sup(X) = \min(\{sup(Y) | Y \in \mathcal{G} \wedge Y \subseteq X\}).$$

2.3 Reprezentacja generatorowa zbiorów częstych

Rodzina wszystkich generatorów jest zwykle bardzo liczna. Nie zawsze też żądamy informacji o wsparciach zbiorów rzadkich. W przypadkach, kiedy chcemy jedynie rozróżniać pomiędzy zbiorami częstymi a rzadkimi oraz wyprowadzać wsparcia tylko dla zbiorów częstych, można zrezygnować z przechowywania informacji o wsparciach wszystkich generatorów. *Reprezentacja generatorowa* zaproponowana w pracy [10] stanowi podzbiór $\mathcal{G}$ spełniający te cele.

- *Reprezentacja generatorowa* (GR) jest definiowana jako zbiór dwóch komponentów:
- *komponentu głównego* składającego się z wszystkich częstych generatorów (*FG*) i przechowującego infomację o ich wsparciach;
- *negatywnej granicy* (GBd^-) zawierającej wszystkie rzadkie generatory, których wszystkie podzbiory właściwe są zawarte w *FG*.

Biorąc pod uwagę własności zbiorów częstych i generatorów, można wysnuć wniosek, że każdy element reprezentacji GR ma wszystkie podzbiory właściwe w *FG* oraz GBd^- jest rodziną minimalnych rzadkich generatorów [13].

Własność 2.3.1 (wyprowadzanie wsparć z użyciem GR) [10, 13].
Niech $X \subseteq I$ będzie zbiorem, którego wsparcie chcemy ustalić. Wówczas:

- jeśli w GBd^- istnieje podzbiór zbioru *X*, to *X* jest rzadki;
- w przeciwnym przypadku *X* jest zbiorem częstym o wsparciu równym najmniejszemu ze wsparć podzbiorów zbioru *X*, które są częstymi generatorami:

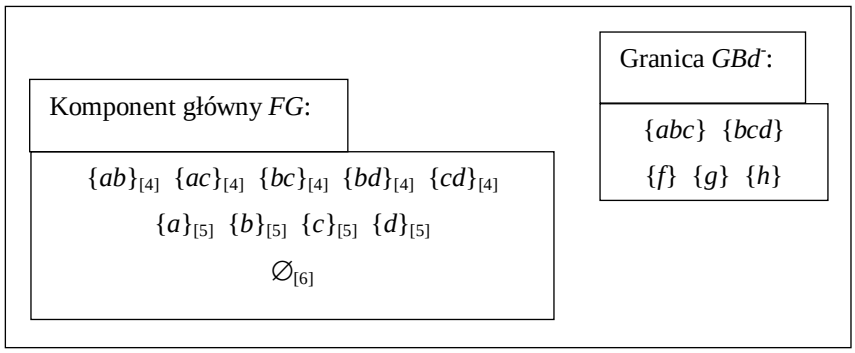
$$sup(X) = \min(\{sup(Y) | Y \in FG \wedge Y \subseteq X\}).$$

Przykład 2.3.1. (wyznaczania wsparcia zbiorów na podstawie reprezentacji generatorowej)
Tabela 1 prezentuje przykładową bazę danych, do której będziemy się odwoływać w raporcie. Rysunek 1 przedstawia reprezentację generatorową GR otrzymaną z danych w tabeli 1 dla progu *minSup* =3. Liczby w nawiasach kwadratowych podają wsparcia elementów komponentu głównego. Pokażemy teraz sposób wykorzystania tej reprezentacji do wyznaczania wsparć zbiorów. Załóżmy, że zbiory, których wsparcia chcemy wyznaczyć, to {*af*} i {*ade*}. Zbiór {*af*} ma podzbiór {*f*} w negatywnej granicy. Ponieważ wszystkie elementy granicy są rzadkie, to {*f*} i jego nadzbiór {*af*} także są rzadkie. Dla odmiany zbiór {*ade*} nie ma podzbiorów w granicy, więc jest zbiorem częstym. Jego wsparcie jest wyznaczane jako minimum ze wsparć podzbiorów zbioru {*ade*} w komponencie głównym, czyli jako $min\{sup(\{a\}), sup(\{d\}), sup(\emptyset)\} = 5$.

□

Identyfikator transakcji	Elementy transakcji
1	*abcdeg*
2	*abcdef*
3	*abcdeh*
4	*abde*
5	*acdeh*
6	*bce*

Tabela 1: Przykładowa baza danych



Rysunek 1: Reprezentacja generatorowa GR zbiorów częstych otrzymana z tabeli 1 dla *minSup* = 3

2.4 Określanie wsparć zbiorów z wykorzystaniem list identyfikatorów transakcji

Dla dowolnego zbioru X *listę identyfikatorów transakcji* zawierających X będziemy oznaczać przez $T(X)$. Oczywiście wsparcie zbioru X jest równe liczbie elementów w $T(X)$.

Znajomość list identyfikatorów transakcji dla wszystkich zbiorów 1-elementowych umożliwia wyznaczenie list identyfikatorów transakcji wszystkich innych zbiorów (a tym samym ich wsparć) w sposób następujący:

$$T(X) = \bigcap_{a \in X} T(\{a\}).$$

Zależność ta implikuje dalej dla dowolnych zbiorów X i Y:

$$T(X \cup Y) = T(X) \cap T(Y).$$

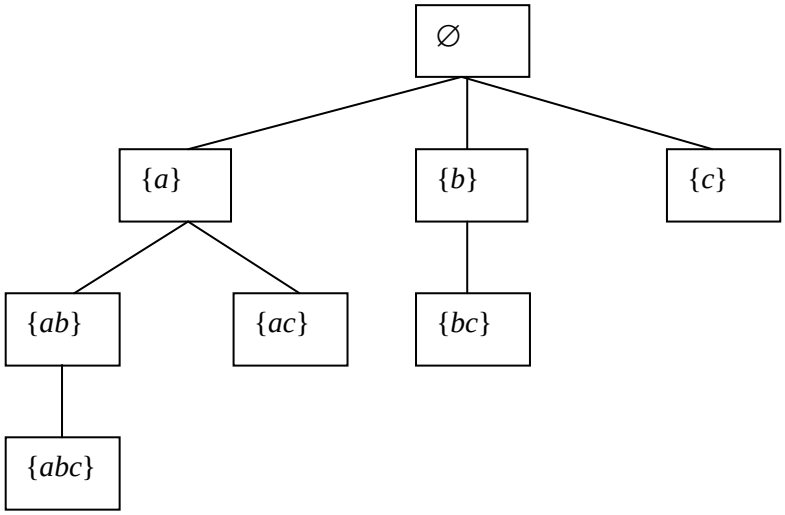
Wyznaczanie wsparć za pomocą list identyfikatorów transakcji wykorzystano w wielu algorytmach eksploracji danych, między innymi w algorytmie CHARM [21] odkrywającym częste zbiory zamknięte. My także wykorzystamy ten sposób wyznaczania wsparć w proponowanym przez nas algorytmie odkrywania reprezentacji generatorowej.

3 Drzewo zbiorów

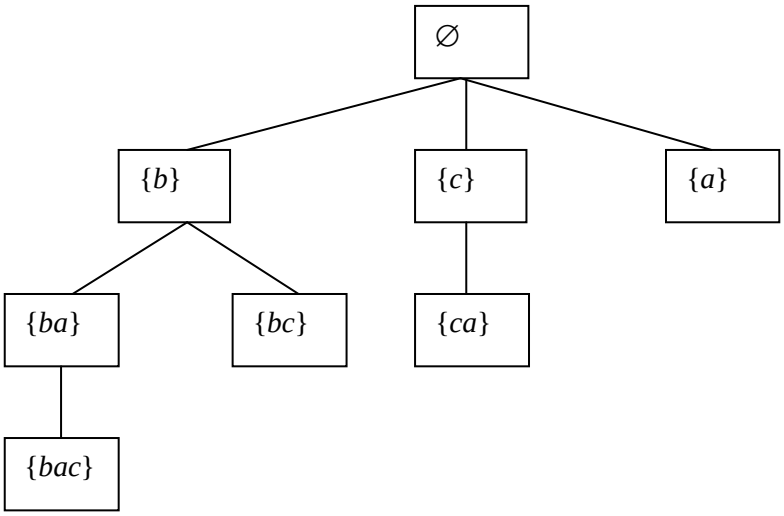
Wszystkie podzbiory zbioru I można przedstawić za pomocą *drzewa zbiorów* (*DZ*) zdefiniowanego następująco:

- korzeniem drzewa jest zbiór pusty;
- potomkami zbioru pustego w drzewie są wszystkie 1-elementowe zbiory w I;
- potomkami każdego niepustego zbioru X w drzewie są wszystkie jego nadzbiory $X \cup Y$, gdzie Y jest bratem X zamieszczonym w drzewie *DZ* po prawej stronie X.

Drzewo *DZ* nie jest zdefiniowane jednoznacznie. Rysunki 2 i 3 prezentują przykładowe drzewa *DZ* dla $I = \{abc\}$. Rysunek 2 przedstawia najbardziej powszechne rozwiązanie uporządkowania wszystkich podzbiorów I w postaci drzewa. W rozwiązaniu tym zakłada się, że wszyscy synowie każdego zbioru w drzewie są uporządkowani leksykograficznie względem siebie. W literaturze poświęconej eksploracji danych często zakłada się, że wszyscy synowie każdego zbioru w drzewie są uporządkowani względem siebie według wsparcia [18, 21]. Nasza definicja drzewa *DZ* nie narzuca żadnego uporządkowania braci w drzewie.

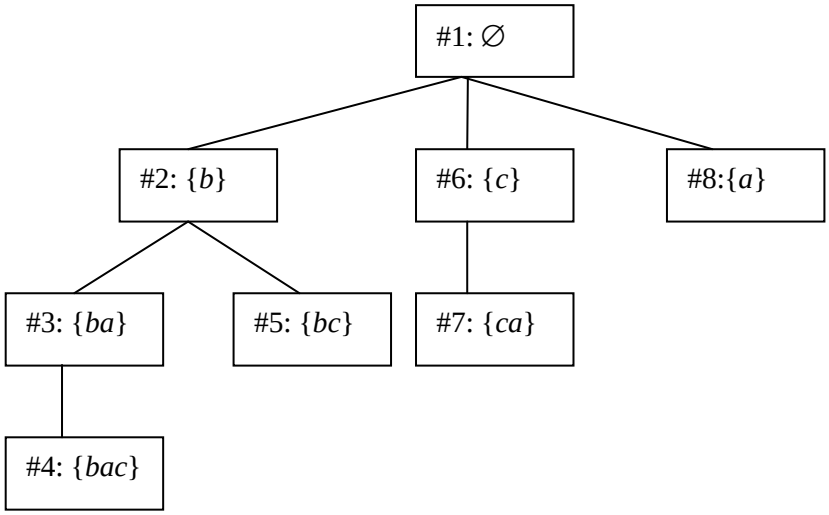


Rysunek 2: Drzewo zbiorów uporządkowane leksykograficznie



Rysunek 3: Inne drzewo zbiorów

Strategia przeszukiwania drzewa *DZ* w głąb lub wszerz zapewnia, że każdy węzeł zostanie jednokrotnie przeanalizowany. W niniejszej pracy przyjmiemy strategię przeszukiwania drzewa w głąb, a w drugiej kolejności, z lewa na prawo. Rysunek 4 prezentuje przyjętą strategię przeszukiwania zbiorów drzewie. Będziemy mówić, że *zbiór X poprzedza zbiór Y* (lub *Y występuje po X*) w drzewie, jeśli *X* jest analizowany wcześniej niż *Y* według tej strategii. Innymi słowy, każdy zbiór *X* w drzewie *DZ* jest poprzedzany wyłącznie przez wszystkich swoich przodków w drzewie, ich lewostronnych braci oraz wszystkich potomków tych braci.



Rysunek 4: Poprzedzanie zbiorów w drzewie *DZ*

Poniżej przedstawiamy własności drzewa *DZ* niezależne od sposobu uszeregowania braci.

Własność 3.1 (drzewa *DZ*).
a) *DZ* zawiera wszystkie podzbiory *I*.
b) Dowolne dwa zbiory będące braćmi w drzewie *DZ*, są postaci $X \cup \{a\}$ oraz $X \cup \{b\}$, gdzie $X \subseteq I$ jest rodzicem obydwu tych braci oraz $a,b \in I \setminus X$ i $a \neq b$.
c) Wszystkie zbiory na tym samym poziomie w drzewie *DZ* są tej samej długości.
d) Niech *X* będzie prawym bratem zbioru *Y* w drzewie *DZ*. Żaden zbiór w poddrzewie drzewa *DZ* o korzeniu *X* nie jest nadzbiorem żadnego zbioru w poddrzewie drzewa *DZ* o korzeniu *Y*.
e) Żaden zbiór *X* w drzewie *DZ* nie jest nadzbiorem jakichkolwiek zbiorów poprzedzających *X*, za wyjątkiem przodków zbioru *X*.

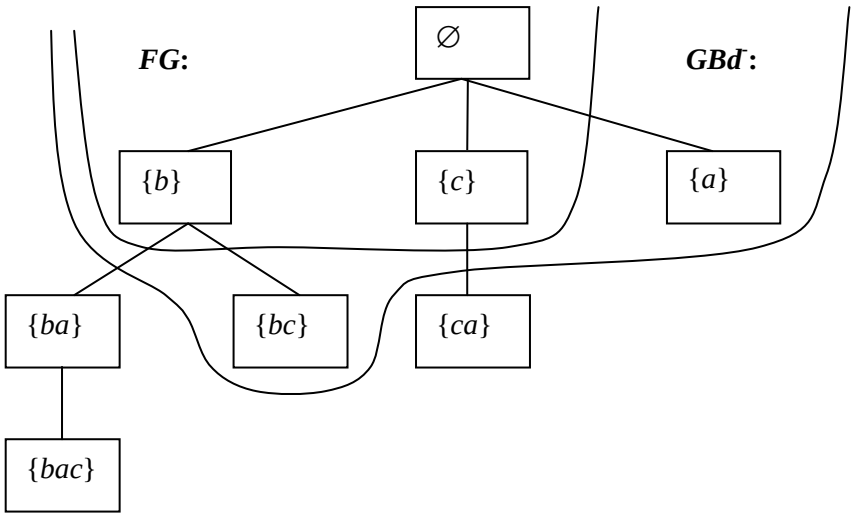
Dowód: Ad d) Ponieważ *Y* i *X* są braćmi, to na mocy Własności 3.1b są odpowiednio postaci $Z \cup \{a\}$ oraz $Z \cup \{b\}$, gdzie $Z \subseteq I$ jest rodzicem obydwu tych braci oraz $a,b \in I \setminus Z$ i $a \neq b$. Ponieważ $Z \cup \{b\}$ jest prawym bratem zbioru $Z \cup \{a\}$, to $Z \cup \{b\}$ oraz wszyscy jego potomkowie w drzewie *DZ* nie będą zawierać elementu *a*. W rezultacie, $X = Z \cup \{b\}$ oraz wszyscy jego potomkowie w drzewie *DZ* nie będą nadzbiorami, ani zbioru $Y = Z \cup \{a\}$, ani jego potomków w drzewie *DZ*.
Ad. e) Niech *X* będzie zbiorem w drzewie *DZ*. Wszystkie zbiory poprzedzające *X* w drzewie *DZ*, to jego przodkowie oraz ich bracia lewostronni wraz ze wszystkimi potomkami tych braci (*). Na mocy Własności 3.1d *X* nie jest nadzbiorem żadnego swojego lewostronnego brata, ani jego potomków w drzewie (**). Na mocy tej samej własności żaden przodek zbioru *X* nie jest nadzbiorem żadnego swojego lewostronnego brata, ani jego potomków w drzewie. Ponieważ *X* należy do podrzew wyznaczanych przez swoich przodków, to na mocy Własności 3.1d *X* także nie jest nadzbiorem żadnego lewostronnego brata swoich przodków, ani potomków tych braci w drzewie *DZ* (***). Własność 3.1.e wynika z (*), (**) i (***).

□

4 Reprezentacja generatorowa a drzewo zbiorów

Naszym zadaniem jest opracowanie algorytmu wyznaczania wszystkich częstych generatorów *FG* i wszystkich minimalnych rzadkich zbiorów *GBd*. Cechą charakterystyczną szukanych generatorów jest to, że ich wszystkie właściwe podzbiory są częstymi generatorami. Zauważamy przy tym, że ∅ bezwarunkowo należy do szukanych zbiorów, ponieważ jest generatorem i jeśli jest częsty, zostanie umieszczony w *FG*, a jeśli jest rzadki, to jest minimalnym rzadkim generatorem, w związku

z czym zostanie umieszczony w *GBd*$^{-}$. Powyższe własności reprezentacji generatorowej oznaczają, że można ją przedstawić w postaci poddrzewa drzewa *DZ* o korzeniu będącym ∅. Rysunek 5 ilustruje przykładową reprezentację generatorową w kontekście drzewa *DZ*.



Rysunek 5: Przykładowa reprezentacja generatorowa w drzewie *DZ*

Stawiamy sobie za zadanie ograniczenie budowy drzewa *DZ* tylko do jego poddrzewa, które zawiera wyłącznie zbiory, o których nie da się wywnioskować a'priori, że nie należą do reprezentacji. Aby ograniczyć rozbudowę drzewa *DZ* o fragmenty nieprzydatne przy wyszukiwaniu reprezentacji GR, będziemy przycinać drzewo korzystając z własności elementów reprezentacji.

Własność 4.1 (zbiorów rzadkich).
Jeśli zbiór *X* jest rzadki, wszystkie jego nadzbiory właściwe są także rzadkie, ale nie są minimalnymi rzadkimi generatorami. A zatem, wszystkie właściwe nadzbiory zbioru rzadkiego nie są elementami reprezentacji GR i dlatego poddrzewo drzewa *DZ*, którego korzeniem jest *X*, powinno być odcięte.

Własność 4.2 (generatorów względem reprezentacji bazy danych – *GRM*-Property).
Niech *X* i *Y* będą niepustymi podzbiorami *I*.

a) Jeśli zbiory *X* i *Y* występują w tych samych transakcjach, to:
   - $X \cup Y$ występuje w tych samych transakcjach co *X* i *Y*. Wynika stąd, że ani $X \cup Y$, ani jego nadzbiory nie są generatorami. Zatem podrzewo drzewa *DZ* o korzeniu $X \cup Y$ nie powinno być tworzone podczas wyszukiwania GR.
   - dla dowolnego zbioru *Z*, $X \cup Z$ występuje w tych samych transakcjach co $Y \cup Z$. Wynika stąd, że wyznaczanie transakcji wspierających nadzbiory zbioru *Y* jest sprowadzalne do wyznaczania transakcji wspierających nadzbiory zbioru *X*.

b) Jeśli zbiór *X* występuje we właściwym podzbiorze zbioru transakcji wspierających zbiór *Y*, to $X \cup Y$ występuje w tych samych transakcjach co *X*. Tym samym $X \cup Y$ nie jest generatorem. W związku z powyższym, fragment drzewa *DZ* składający się z nadzbiorów $X \cup Y$ nie zawiera żadnych elementów reprezentacji GR, więc powinien być odcięty.

c) Jeśli zbiory transakcji wspierających *X* i *Y* są różne i żaden z tych zbiorów transakcji nie jest podzbiorem właściwym drugiego, to zbiór transakcji wspierających $X \cup Y$ jest podzbiorem właściwym zarówno zbioru transakcji wspierających *X*, jak i zbioru transakcji wspierających *Y*. Tym samym, wsparcie zbioru $X \cup Y$ jest mniejsze od wsparcia zbioru *X* i wsparcia zbioru *Y*. Jest więc możliwe, że $X \cup Y$ jest generatorem.

Zauważmy, że przechodząc drzewo *DZ* zgodnie z przyjętą strategią, aktualnie analizowany częsty zbiór *X* może okazać się podzbiorem właściwym pewnych zbiorów przeanalizowanych wcześniej (np. na rysunku 4 zbiór {*ca*} jest podzbiorem właściwym poprzedzającego go zbioru {*bac*}) i mających to samo wsparcie co *X*. Zbiory takie należy usunąć, gdyż nie są generatorami. Skądinąd na mocy Własności 3.1e, *X* nie jest nadzbiorem żadnego z wcześniej odwiedzonych zbiorów za wyjątkiem swoich przodków.

Podobne rozważania można przeprowadzić dla aktualnie analizowanego zbioru *Y*, który jest rzadki. Wśród wcześniej przeanalizowanych rzadkich zbiorów mogą się znajdować nadzbiory właściwe zbioru *Y*. Oczywiście takie nadzbiory nie są minimalnymi rzadkimi generatorami i powinny być odrzucone. Jednakże na mocy Własności 3.1e, *Y* nie jest nadzbiorem żadnego z wcześniej przeanalizowanych zbiorów za wyjątkiem swoich przodków.

W szczególnych przypadkach, zbiory wcześniej przeanalizowane nie zostaną wyeliminowane przez żadne podzbiory później występujące w drzewie *DZ*. Własność 4.3 wymienia te przypadki.

Własność 4.3 (generatorów 0-, 1- i 2-elementowych).

a) ∅ należy do reprezentacji GR.
b) Jeśli ∅ jest częsty, to każdy zbiór 1-elementowy, którego wsparcie jest mniejsze od liczby transakcji, należy do reprezentacji GR.
c) Każdy zbiór 2-elementowy, powstały ze sklejenia dwóch częstych 1-elementowych generatorów wspieranych przez zbiory transakcji, między którymi nie zachodzi relacja zawierania, należy do reprezentacji GR.

Dowód: Ad b) Jedynym podzbiorem właściwym zbioru 1-elementowego jest ∅. Wsparcie ∅ jest równe liczbie transakcji. Zatem każdy zbiór 1-elementowy, którego wsparcie jest mniejsze od liczby transakcji, jest generatorem częstym lub minimalnym rzadkim (ponieważ jedyny podzbiór właściwy zbioru 1-elementowego - ∅ - jest częstym generatorem).
Ad c) Zbiór 2-elementowy, powstały ze sklejenia dwóch częstych 1-elementowych generatorów *X* i *Y* wspieranych przez zbiory transakcji, między którymi nie zachodzi relacja zawierania, jest wspierany przez zbiór transakcji stanowiący podzbiór właściwy zarówno transakcji wspierających *X*, jak i transakcji wspierających *Y*. A zatem, wsparcie takiego 2-elementowego zbioru jest mniejsze od wsparć zbiorów, z których został wytworzony, i wsparć ich podzbiorów. Ponieważ zbiór wynikowy ma tylko 2 podzbiory właściwe krótsze o 1 element (mianowicie *X* i *Y*) i obydwa te podzbiory są częste oraz mają wsparcia różne od wsparć zbioru wynikowego, zbiór wynikowy jest generatorem częstym lub minimalnym rzadkim.

□

5 Odkrywanie reprezentacji generatorowej

5.1 Algorytm GRM

Zaprezentujemy obecnie algorytm *GRM* stanowiący naszą propozycję wyznaczania reprezentacji generatorowej jako poddrzewa drzewa *DZ*. Do określania wsparć wartościowanych zbiorów będziemy używać list identyfikatorów transakcji skojarzonych z odpowiednimi zbiorami.

Ujmując zagadnienie w sposób techniczny, z każdym węzłem w drzewie *DZ* związana będzie rodzina potencjalnych generatorów, lista identyfikatorów transakcji wspierających te generatory oraz wskazania na węzły potomne.

Oznaczenia:

root, N	- węzły w drzewie
N.children	- węzły potomne węzła *N*
$N.\mathcal{G}$	- rodzina potencjalnych generatorów związana z węzłem *N*
N.T	- lista identyfikatorów transakcji, w których występują $N.\mathcal{G}$

```
Algorithm GRM (𝒟, minSup):
      FG = GBd- = {}
      if (|𝒟| > minSup) then
            root.𝒢 = {∅}
            dodaj {{a}: a ∈ I ∧ {a}.sup ≤ minSup} do GBd-
            forall a ∈ I such that minSup < {a}.sup < |𝒟| do
                  N.𝒢 = {{a}}
                  N.T = <identyfikatory transakcji, w których występuje a>
                  dodaj N do root.children
            endfor
            GARM (root, FG, GBd-, minSup)
            dodaj ∅ do FG
      else
            dodaj ∅ do GBd-
      endif
      return <FG, GBd>   //generatory częste i negatywna granica

procedure GARM (węzeł N, FG, GBd-, minSup):
      {* LN – węzeł potomny węzła N; RN – prawy brat węzła LN *}
      forall LN in N.children do
            forall RN in N.children taki, że RN jest na prawo od LN
                  GARM-Property(N, LN, RN)
            endfor
            LN.𝒢 = {X ∪ Y: X ∈ LN.𝒢 ∧ Y ∈ N.𝒢}
            //generatory rodzica
            forall X ∈ LN.𝒢 do begin
                  usuń z FG wszystkie nadzbiory X, których wsparcie wynosi X.sup
                  dodaj X do FG
            endfor
            forall węzły N' in LN.children do
                  if (|N'.T| ≤ minSup) then
                        N'.G = {X ∪ Y: X ∈ N'.𝒢 ∧ Y ∈ LN.𝒢}   // |N'.𝒢| = 1
                        forall X ∈ N'.𝒢 do begin
                              usuń z GBd- wszystkie nadzbiory X
                              dodaj X do GBd-
                        endfor
                        usuń N' z LN.children
                  endif
            endfor
            GARM (LN, FG, GBd-, minSup)
      endfor

procedure GARM-Property(N, LN, RN):
      if LN.T = RN.T then
            Usuń RN z N.children
            LN.𝒢 = LN.𝒢 ∪ RN.𝒢
      else if LN.T ⊂ RN.T or LN.T ⊃ RN.T then
            {* żaden nadzbiór zbioru {XYZ}: Z∈N.𝒢, X∈LN.𝒢, Y∈RN.𝒢, nie jest
      generatorem *}
      else if LN.T ≠ RN.T then
            N' = kopia RN
            N'.T = LN.T ∩ RN.T
            LN.children = LN.children ∪ {N'}
      endif
```

Algorytm rozpoczyna działanie od sprawdzenia, czy próg *minSup* żądanego wsparcia przekracza rozmiar bazy danych. Jeśli tak, to zbiór pusty jest zamieszczony w negatywnej granicy i działanie algorytmu kończy się. Jeśli *minSup* nie przekracza rozmiaru bazy danych, to zbiór pusty jest zamieszczony w korzeniu drzewa. Zbiory, których wsparcie jest równe liczbie transakcji w bazie danych, nie są generatorami, gdyż posiadają przynajmniej jeden podzbiór o tym samym wsparciu w bazie danych – zbiór pusty. Wszystkie rzadkie jednoelementowe zbiory są generatorami i zapisywane są do negatywnej granicy. W kolejnym kroku tworzone są węzły potomne korzenia z

jednoelementowych zbiorów częstych, których wsparcie jest mniejsze od rozmiaru bazy danych. Na rzecz korzenia wywoływana jest procedura *GARM*.

Procedura *GARM* przegląda przestrzeń węzłów potomnych węzła *N*, na rzecz którego została wywołana. Dla każdego węzła *LN* wykonywane są następujące kroki:

- dla wszystkich węzłów *RN* będących potomkami *N*, które są prawymi braćmi węzła *LN*, wywoływana jest procedura *GARM-Property*, która w zależności od relacji zachodzącej między listami *T* węzłów *LN* i *RN* odpowiednio modyfikuje poddrzewo z korzeniem w węźle *N*;
- rodzina potencjalnych generatorów węzła *LN* budowana jest jako iloczyn kartezjański zbioru początkowych pozycji pamiętanych w polu generatorów węzła *LN* i zbioru potencjalnych generatorów rodzicielskiego węzła *N*;
- wszystkie potencjalne generatory z węzła *LN* dodawane są do wynikowej rodziny częstych generatorów *FG*. Dodanie każdego z nich do rodziny wynikowej poprzedzone jest usunięciem z tej rodziny wszystkich nadzbiorów zbioru dodawanego o tym samym wsparciu;
- wszystkie węzły potomne węzła *LN* (mogły zostać dodane w wyniku działania procedury *GARM-Property*) przeglądane są w celu sprawdzenia czy wsparcie przechowywanych potencjalnych generatorów przekracza próg *minSup*. Dla każdego węzła potomnego *N'*, którego pole *T* spełnia warunek $|N'.T| \leq minSup$, jego rodzina potencjalnych generatorów tworzona jest jako iloczyn kartezjański początkowego zbioru generatorów węzła *N'* i zbioru potencjalnych generatorów węzła rodzicielskiego *LN*. Następnie ta rozszerzona rodzina dodawana jest do granicy, a cały węzeł *N'* usuwany jest ze zbioru potomków węzła *LN*. Dodanie każdego potencjalnego generatora rzadkiego do granicy poprzedzone jest usunięciem z niej wszystkich nadzbiorów zbioru dodawanego;
- rekurencyjnie wywoływana jest procedura *GARM* na rzecz węzła *LN*.

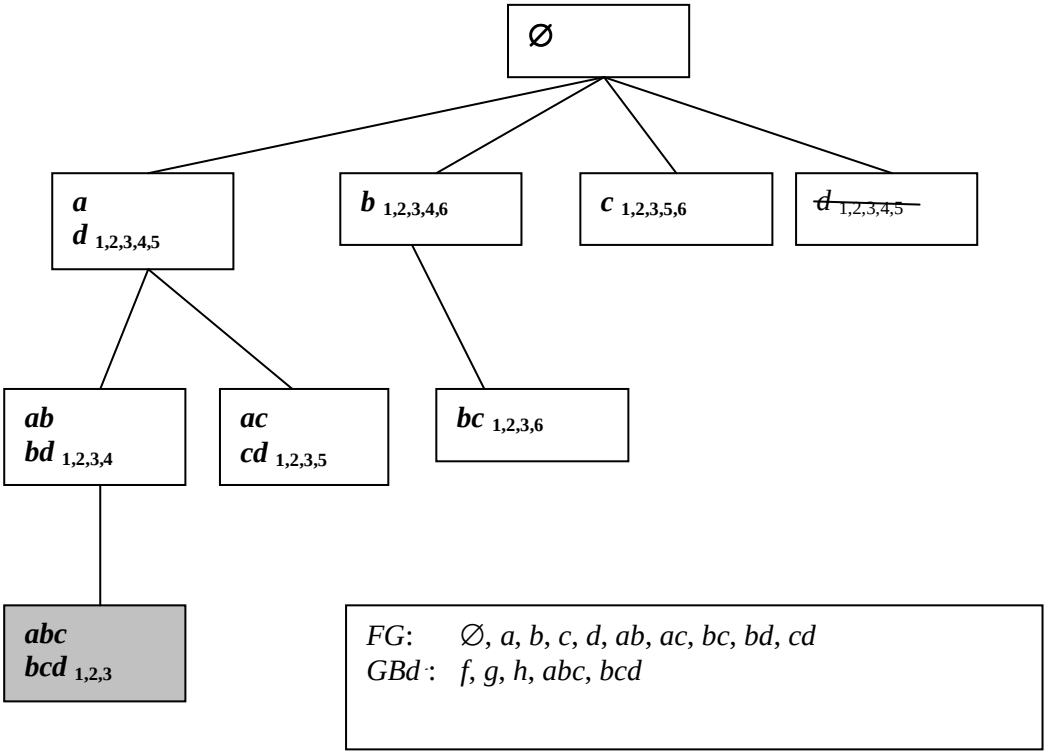
Procedura *GARM-Property* otrzymuje jako argumenty węzeł *N* oraz dwa spośród jego węzłów potomnych – węzły *LN* i *RN*. Relacja zachodząca pomiędzy listami identyfikatorów transakcji węzłów *LN* i *RN* decyduje o tym, która, spośród trzech możliwych sytuacji ma miejsce:

- zbiory identyfikatorów transakcji węzłów *LN* i *RN* są równe – oznacza to, że zbiory z rodzin generatorów obu węzłów występują w tych samych transakcjach. Nie ma sensu przechowywanie dwu takich węzłów. Potencjalne generatory węzła *RN* dodawane są zatem do rodziny potencjalnych generatorów węzła *LN*, a sam węzeł *RN* zostaje usunięty spośród potomków węzła *N*;
- Lista identyfikatorów transakcji jednego z węzłów jest podzbiorem właściwym zbioru identyfikatorów drugiego węzła – oznacza to, że potencjalne generatory jednego z węzłów występują częściej w bazie danych i na pewno są we wszystkich transakcjach, w których występują potencjalne generatory drugiego węzła. W tej sytuacji, nie ma sensu ani tworzyć nowy węzeł (gdyż żaden nadzbiór zbioru $\{XYZ\}$: $Z \in N.\mathcal{G}$, $X \in LN.\mathcal{G}$, $Y \in RN.\mathcal{G}$ nie jest generatorem), ani usuwać którykolwiek z węzłów (gdyż nadzbiory generatorów każdego z węzłów z osobna, mogą być generatorami). Procedura *GARM-Property* nie dokonuje tutaj żadnych zmian w poddrzewie węzła *N*;
- zbiory identyfikatorów transakcji węzła *LN* i węzła *RN* są różne (lecz nie zachodzi sytuacja opisana powyżej) – oznacza to, że zbiory utworzone jako iloczyn kartezjański rodzin potencjalnych generatorów obu węzłów pojawiają się w transakcjach, których lista identyfikatorów jest podzbiorem właściwym zarówno zbioru *LN.T* jak i *RN.T*, a co za tym idzie, zbiory będące wynikiem iloczynu $LN.\mathcal{G} \times RN.\mathcal{G}$ mają szansę być generatorami. Utworzony zostaje wobec tego węzeł *N'* będący kopią węzła *RN*. Zbiór *N'.T* obliczany zostaje jako iloczyn $LN.T \cap RN.T$, a węzeł *N'* dodany do potomków węzła *LN*.

W algorytmie głównym, po powrocie z procedury *GARM*, następuje dołączenie zbioru pustego do reprezentacji generatorowej.

5.2 Przykład działania algorytmu GRM

W przykładzie działania posłużymy się bazą danych przedstawioną w tabeli 1 ustalając *minSup* = 3.



Rysunek 6: Poddrzewo drzewa *DZ* (z listami identyfikatorów transakcji) zbudowane w wyniku działania algorytmu *GRM* na danych z tabeli 1 przy *minSup* = 3

Korzeń drzewa budowanego przez *GRM* jest inicjalizowany zbiorem pustym o wsparciu 6. Jako węzły potomne korzenia dodane zostały (tu: w porządku leksykograficznym)[2] jednoelementowe zbiory częste wraz z listami identyfikatorów transakcji: {*a*, 12345}, {*b*, 12346}, {*c*, 12356}, {*d*, 12345}, których wsparcie jest mniejsze od 6. Wszystkie jednoelementowe zbiory rzadkie są generatorami – dodajemy je zatem do granicy: {*f*, 2}, {*g*, 1}, {*h*, 35}. Następnie na rzecz korzenia wywołana zostaje procedura *GARM*.

Ponieważ całe ciało procedury GARM zawiera się w pętli przebiegającej po potomkach węzła, na rzecz którego została wywołana, w dalszej części tego opisu węzeł *LN* będący iteratorem tej pętli nazywać będziemy węzłem bieżącym. Pierwszym węzłem bieżącym tego wywołania *GARM* jest węzeł {*a*, 12345}. Najpierw wykonywana jest procedura *GARM-Property* ustalająca relację zachodzącą pomiędzy {*a*, 12345} i {*b*, 12346}. Ponieważ zachodzi nierówność list identyfikatorów transakcji, przy czym nie zachodzi relacja zawierania, do potomków węzła {*a*, 12345} dodany zostaje nowy węzeł {*b*, 1234}, którego lista identyfikatorów transakcji obliczona została jako iloczyn list identyfikatorów transakcji węzłów rodzicielskich. Podobnie wywołanie *GARM-Property* dla węzłów {*a*, 12345} i {*c*, 12356} daje w wyniku nowego potomka węzła {*a*, 12345}: {*c*, 1235}. Natomiast wywołanie *GARM-Property* dla węzłów {*a*, 12345} i {*d*, 12345}, wobec równości ich list identyfikatorów transakcji spowoduje dodanie nowego potencjalnego generatora *d* do węzła {*a*, 12345} i usunięcie węzła {*d*, 12345}. Od tej chwili węzeł, który aktualnie przetwarzamy, posiada 2 potencjalne generatory i zapisywać go będziemy następująco: {*a*, *d*, 12345}. Wywołujemy procedurę *GARM-Property* dla węzłów {*a*, *d*, 12345} i {*e*, 123456} i odkrywamy, że zachodzi relacja zawierania pomiędzy listami identyfikatorów transakcji. Wobec powyższego nie podejmujemy żadnej akcji modyfikującej drzewo. W ten sposób zakończyliśmy pętlę modyfikującą drzewo na podstawie relacji zachodzącej pomiędzy listą identyfikatorów transakcji przetwarzanego węzła a listami jego prawych braci. Etap uzupełniania (przez iloczyn kartezjański) potencjalnych generatorów bieżącego węzła o generatory węzła rodzicielskiego nie wprowadza żadnych zmian, gdyż jedynym generatorem

[2] Można zastosować dowolny porządek, np. według wsparcia.

węzła rodzicielskiego jest zbiór pusty. Do wynikowej rodziny *FG* dodane zostają wszystkie potencjalne generatory bieżącego węzła, a więc: {*a*} o wsparciu 5 i {*d*} o wsparciu 5. Wszystkie węzły potomne bieżącego węzła zawierają zbiory częste, więc sekcja tworząca generatory granicy nie jest wykonywana. Wywołana zostaje rekurencyjnie procedura *GARM* na rzecz aktualnie przetwarzanego węzła {*a*, *d*, 12345}.

W tym wywołaniu jako pierwszy przetwarzany będzie węzeł {*b*, 1234}. Wywołanie procedury *GARM-Property* porównującej listę identyfikatorów węzła {*b*, 1234} z listą identyfikatorów jego jedynego prawego brata: {*c*, 1235}, powoduje powstanie nowego węzła potomnego {*c*, 123}.

Uzupełnianie potencjalnych generatorów bieżącego węzła {*b*, 1234} o potencjalne generatory węzła rodzicielskiego {*a*, *d*, 12345} dokonuje się przez iloczyn kartezjański obu zbiorów potencjalnych generatorów. W ten sposób węzeł {*b*, 1234} zamienia się w węzeł {*ab*, *bd*, 1234}. Potencjalne generatory {*ab*} i {*bd*} o wsparciu 4 zostają dodane do *FG*. Jedyny potomek bieżącego węzła - węzeł {*c*, 123} zawiera potencjalny rzadki generator o wsparciu 3. Elementy tego potencjalnego generatora są również uzupełniane o potencjalne generatory rodzica – węzeł {*c*, 123} zamienia się zatem w {*abc*, *bcd*, 123}. Dwa rzadkie potencjalne generatory {*abc*} i {*bcd*} zostają dodane do wynikowej granicy GBd^-, a węzeł {*abc*, *bcd*, 123} usunięty z drzewa. Wywołanie procedury *GARM* na rzecz węzła {*ab*, *bd*, 1234} kończy się natychmiastowym powrotem, gdyż ten węzeł nie posiada już potomków. Bieżącym węzłem staje się następnie węzeł {*c*, 1235}. Ponieważ nie posiada on już prawostronnych braci w drzewie, procedura *GARM-Property* nie zostanie wywołana. Uzupełnienie potencjalnych generatorów z wykorzystaniem potencjalnych generatorów węzła rodzicielskiego {*a*, *d*, 12345} zamienia węzeł {*c*, 1235} w węzeł {*ac*, *cd*, 1235}. Wywołanie procedury *GARM* na rzecz węzła {*ac*, *cd*, 1235} kończy się natychmiastowym powrotem, gdyż ten węzeł nie posiada potomków. Następuje powrót do przetwarzania kolejnych węzłów potomnych korzenia – węzłem bieżącym staje się {*b*, 12346}. Wywołanie procedury *GARM-Property* na rzecz węzła bieżącego i węzła {*c*, 12356} powoduje dodanie węzła {*c*, 1236} do potomków węzła {*b*, 12346}. *GARM-Property* wywołana na rzecz węzłów: bieżącego i {*e*, 123456}, wobec istnienia relacji zawierania pomiędzy listami identyfikatorów w tych węzłach, nie modyfikuje drzewa. Potencjalny częsty generator {*b*} o wsparciu 5 zostaje dodany do *FG*. Bieżący węzeł nie posiada potomków z potencjalnymi rzadkimi generatorami. Wywołana zostaje procedura *GARM* na rzecz węzła {*b*, 12346}. Ponieważ jego jedyny potomek: węzeł {*c*, 1236} nie ma ani prawostronnych braci, ani potomków, wykonujemy jedynie wyznaczenie prawdziwych potencjalnych generatorów węzła przez iloczyn kartezjański z potencjalnymi generatorami węzła rodzicielskiego (uzyskujemy węzeł {*bc*, 1236}) i dodanie generatora {*bc*} ze wsparciem 4 do *FG*. Wywołanie *GARM* na rzecz bezpotomnego węzła {*bc*, 1236} kończy się natychmiastowym powrotem z procedury. Kolejnym potomkiem korzenia, który staje się węzłem bieżącym jest {*c*, 12356}. Ponieważ nie posiada on prawostronnych braci, *GARM-Property* nie będzie wywoływana. Zbiór {*c*} o wsparciu 5 zostaje dodany do *FG*. Wywołanie *GARM* na rzecz bezpotomnego węzła {*c*, 12356} kończy się natychmiastowym powrotem z procedury. Na tym również kończy się wywołanie *GARM* na rzecz korzenia całego drzewa.

Każde dodanie zbioru do *FG* poprzedzone jest usunięciem z tego komponentu ewentualnych nadzbiorów zbioru dodawanego o tym samym wsparciu. Każde dodanie zbioru do GBd^- poprzedzone jest usunięciem z granicy ewentualnych nadzbiorów zbioru dodawanego.

Ostatnią akcją, jaką zostanie wykonana, jest dodanie zbioru pustego do *FG*.

5.3 Uwagi implementacyjne

Algorytm *GRM* weryfikuje potencjalne częste generatory poprzez wyszukiwanie nadzbiorów danego zbioru *X* o tym samym wsparciu. Jakkolwiek wyszukiwanie nadzbiorów jest operacją czasochłonną, to wyszukiwanie nadzbiorów o tym samym wsparciu może być wykonywane bardzo efektywnie przy użyciu funkcji mieszającej stosowanej do list identyfikatorów transakcji [21]. Rozwiązanie to zastosujemy do weryfikowania częstych generatorów. Jak wykażą eksperymenty, jest ono bardzo efektywne.

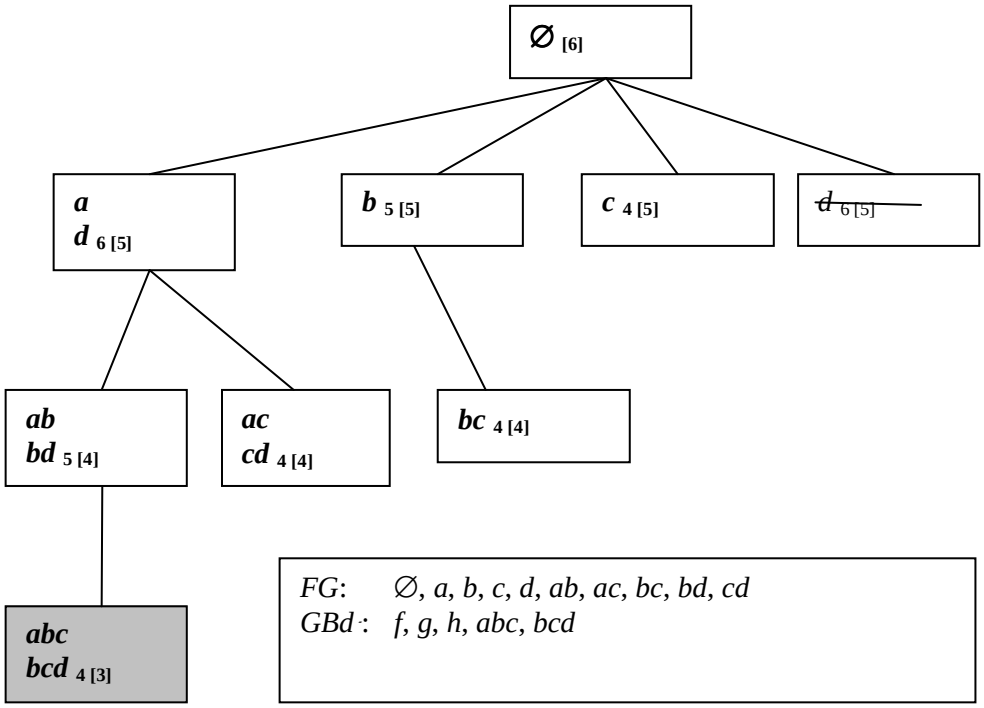
W przeciwieństwie do potencjalnych częstych generatorów, potencjalne generatory granicy weryfikuje się poprzez wyszukiwanie nadzbiorów danego zbioru bez względu na ich wsparcie. Zamiast wykonywać tę czasochłonną operację, proponujemy wykonywać inną skuteczną weryfikację potencjalnych generatorów granicy. Otóż proponujemy, aby zamiast jak najszybciej eliminować wszystkie fałszywe potencjalne minimalne rzadkie generatory, przeprowadzić ich weryfikację po zakończeniu budowy drzewa, kiedy znane już będą wszystkie częste generatory. Wtedy spośród znalezionych potencjalnych generatorów granicy będziemy eliminować te, które nie mają wszystkich częstych podzbiorów krótszych o jedną pozycję wśród częstych generatorów. Aby szybko wyszukiwać podzbiory elementów wartościowanych, częste generatory warto przechowywać w strukturze drzewa mieszającego [2] lub w tablicy mieszającej. Omawianą operację weryfikacji elementów granicznych możemy opcjonalnie poprzedzić usuwaniem na bieżąco tych potencjalnych generatorów granicy, które są nadzbiorami ostatnio znalezionego potencjalnego minimalnego rzadkiego generatora i mają to samo wsparcie. Poniżej zamieszczamy pseudokod przedstawiający algorytm *GRM* uwzględniający wyżej omówiony sposób weryfikacji elementów granicy. Wersja ta została zaimplementowana i przebadana eksperymentalnie.

```
Algorithm GRM (𝒟, minSup):
    FG = GBd- = {}
    if (|𝒟| > minSup) then
        root.𝒢 = {∅}
        dodaj {{a}: a ∈ ℐ ∧ {a}.sup ≤ minSup} do GBd-
        forall a ∈ ℐ such that minSup < {a}.sup < |𝒟| do
            N.𝒢 = {{a}}
            N.T = <identyfikatory transakcji, w których występuje a>
            dodaj N do root.children
        endfor
        GARM (root, FG, GBd-, minSup)
        dodaj ∅ do FG
        usuń z GBd- wszystkie zbiory, których o 1 pozycję
                mniej liczne podzbiory nie należą do FG
    else
        dodaj ∅ do GBd-
    endif
    return <FG, GBd>   //generatory częste i negatywna granica

procedure GARM (węzeł N, FG, GBd-, minSup):
    {* LN – węzeł potomny węzła N; RN – prawy brat węzła LN *}
    forall LN in N.children do
        forall RN in N.children taki, że RN jest na prawo od LN
            GARM-Property(N, LN, RN)
        endfor
        LN.𝒢 = {X ∪ Y: X ∈ LN.𝒢 ∧ Y ∈ N.𝒢}
        //generatory rodzica
        forall X ∈ LN.𝒢 do begin
            usuń z FG wszystkie nadzbiory X, których wsparcie wynosi X.sup
            dodaj X do FG
        endfor
        forall węzły N' in LN.children do
            if (|N'.T| ≤ minSup) then
                N'.G = {X ∪ Y: X ∈ N'.𝒢 ∧ Y ∈ LN.𝒢}      // |N'.𝒢| = 1
                forall X ∈ N'.𝒢 do begin
                    usuń z GBd- wszystkie nadzbiory X,  // opcjonalnie
                        których wsparcie wynosi X.sup
                    dodaj X do GBd-
                endfor
                usuń N' z LN.children
            endif
        endfor
        GARM (LN, FG, GBd-, minSup)
    endfor
```

5.4 Zastosowanie list różnicowych

W [21] zaproponowano *listy różnicowe* (oznaczane przez *D*) jako alternatywny do list identyfikatorów sposób przechowywania informacji o transakcjach, w których występują zbiory. Mianowicie, zamiast przechowywać informację, w których transakcjach występuje zbiór, przechowuje się informację o tym, gdzie zbiór nie występuje. Informacja ta jest jednak kontekstowa. Jeśli *X* jest rodzicem *Y* w drzewie, to *D*(*Y*) zawiera jedynie identyfikatory transakcji, w których nie występuje *Y*, wyłączywszy te identyfikatory transakcji, w których nie występuje rodzic *X*. Rysunek 7 pokazuje *D*(*X*) dla każdego zbioru w drzewie z przykładu omawianego w rozdz. 5.2.



Rysunek 7: Poddrzewo drzewa *DZ* (z listami różnicowymi) zbudowane w wyniku działania algorytmu *GRM* na danych z tabeli 1 przy *minSup* = 3

Listę różnicową dla potomka węzła w drzewie wyznacza się w sposób następujący: Jeśli bieżący węzeł zawiera zbiór *X*, a jego prawostronny brat zawiera zbiór *Y*, to lista różnicowa potomnego węzła $X \cup Y$ jest różnicą teoriomnogościową listy różnicowej zbioru *Y* i listy różnicowej zbioru *X* [21]:

$$D(X \cup Y) = D(Y) \setminus D(X).$$

Wsparcia list różnicowych węzłów potomnych są wyznaczane efektywnie, jeśli w węzłach przechowuje się informację o wsparciach rodziców. Mianowicie, jeśli *X* jest rodzicem *Z*, to:

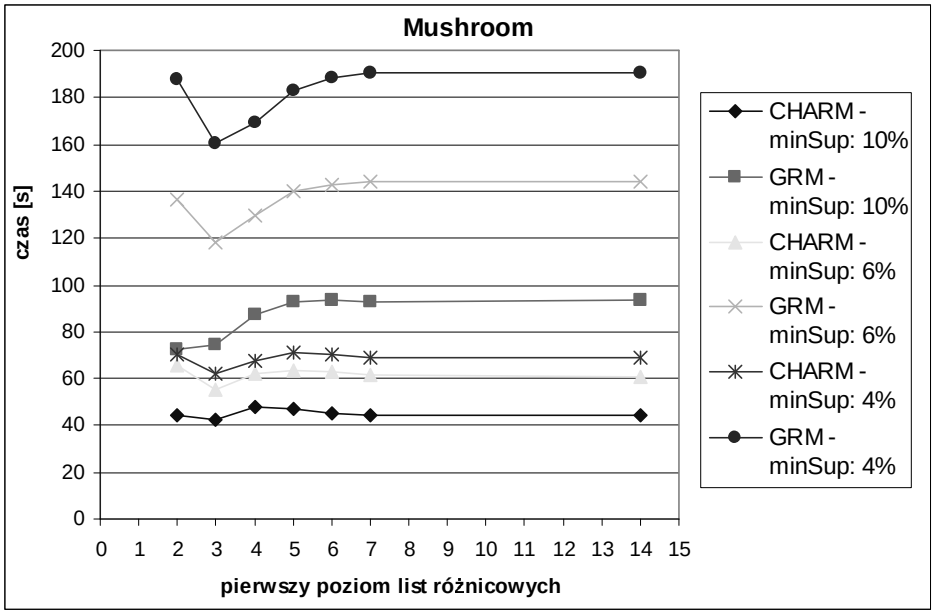
$$sup(Z) = sup(X) - |D(Z)|.$$

W naszych eksperymentach przeprowadziliśmy badanie algorytmu *GRM* zarówno z wykorzystaniem list identyfikatorów transakcji, jak i z wykorzystaniem list różnicowych. W przeciwieństwie do innych prac wykorzystujących struktury listowe, zbadaliśmy także efektywność *GRM* stosując listy identyfikatorów transakcji do zadanego poziomu w drzewie, a listy różnicowe dla poziomów dalszych. Przechodząc ze struktury list identyfikatorów transakcji na listy różnicowe, korzystaliśmy z następującej zależności [21]:

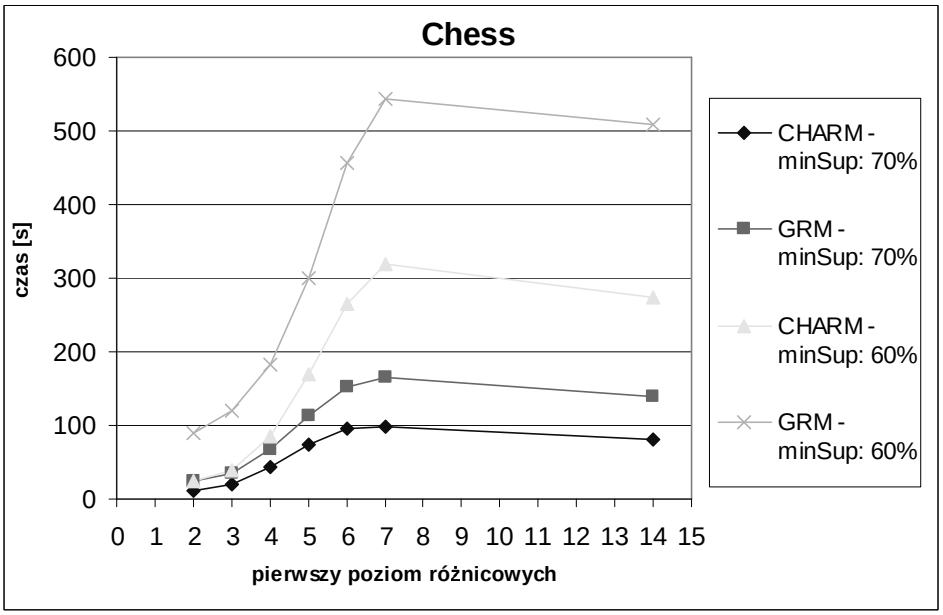
$$D(X \cup Y) = T(X) \setminus T(Y).$$

6 Eksperymenty

Poniżej przedstawiamy wyniki eskperymentów przeprowadzonych na powszechnie testowanych zbiorach danych *mushroom*, *connect-4* i *chess* [22]. Wykresy 1 i 2 pokazują rezultaty badań, od którego poziomu warto stosować listy różnicowe przy zadanym progu *minSup* dla zbiorów danych odpowiednio *mushroom* i *chess*. W badaniach przyjęliśmy, że korzeń drzewa jest na poziomie 0. Poziom 14 odpowiada obliczeniom z wykorzystaniem wyłącznie list identyfikatorów transakcji. W przypadku zbioru *mushroom* najlepszą efektywność algorytmu *GRM* uzyskuje się przy przełączeniu na listy różnicowe na poziomie 3 drzewa. W przypadku *chess*, *GRM* jest najefektywniejszy przy wykorzystaniu list różnicowych od poziomu 2.

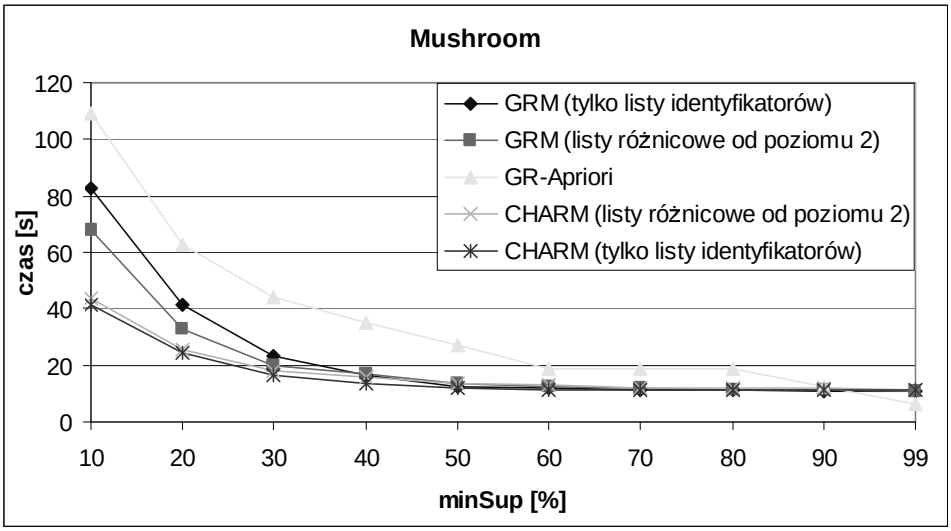


Wykres 1: Czas wykonania *GRM* na danych *mushroom* w zależności od poziomu przełączenia na listy różnicowe

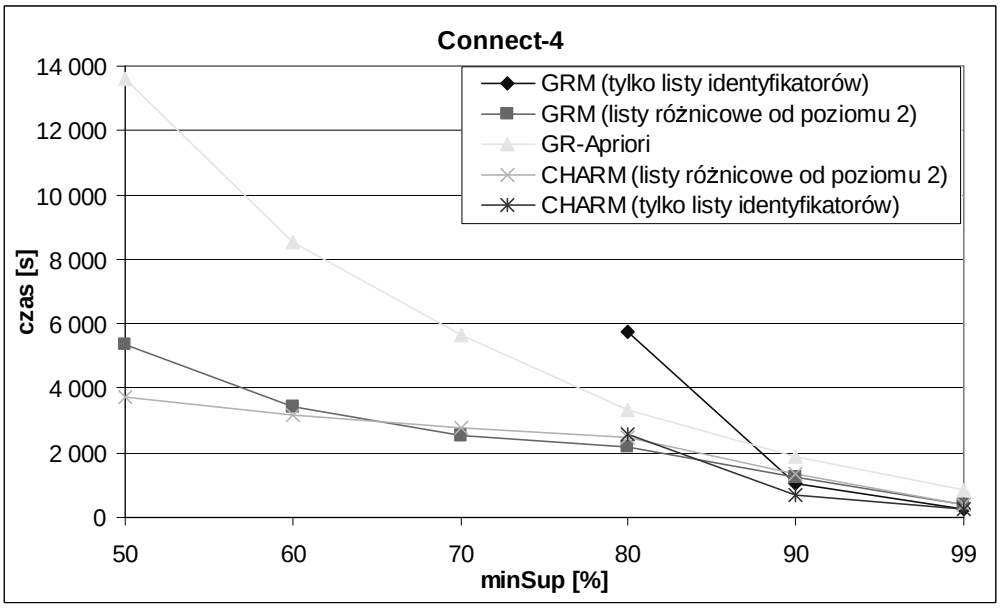


Wykres 2: Czas wykonania *GRM* na danych *chess* w zależności od poziomu przełączenia na listy różnicowe

Wykresy 3 i 4 (p. także tabele 2 i 3) prezentują czasy wykonania algorytmów odkrywających reprezentację GR: *GRM* i *GR-Apriori* [13] oraz czas wykonania algorytmu *CHARM* odkrywającego częste zbiory zamknięte ze zbiorów danych, odpowiednio *mushroom* i *connect-4*. *GR-Apriori*, w przeciwieństwie do pozostałych dwóch algorytmów, nie wykorzystuje struktur listowych do wartościowania elementów kandydujących. Dla algorytmów *GRM* i *CHARM* relacjonujemy zarówno wyniki uzyskane z wykorzystaniem wyłącznie list identyfikatorów transakcji, jak i wyniki uzyskane przy przełączaniu się na listy różnicowe od poziomu 2 drzewa. Eksperymenty wskazują, że algorytmy *GRM* i *CHARM* wykonują się szybciej niż *GR-Apriori* dla średnich i małych wartości progowych. Tabele 2 i 3 zawierają ponadto informacje o licznościach odkrytych reprezentacji, liczbie elementów odrzucanych na etapie weryfikacji w *GRM* i *CHARM* oraz o czasochłonnosci tego etapu. Okazuje sie, że weryfikacja jest przeprowadzana bardzo sprawnie i stanowi znikomy fragment całkowitego czasu wykonania tych algorytmów.



Wykres 3: Czas wykonania algorytmów *GRM*, *GR-Apriori* i *CHARM* na danych *mushroom* w zależności od wartości progowej wsparcia



Wykres 4: Czas wykonania algorytmów *GRM*, *GR-Apriori* i *CHARM* na danych *conect-4* w zależności od wartości progowej wsparcia

Mushroom	GRM (tylko listy identyfikatorów)								GRM (listy różnicowe od poziomu 2)							
			FG (weryfikacja bież.)		GBd (weryfikacja bież.)		GBd (weryfikacja końc.)				FG (weryfikacja bież.)		GBd (weryfikacja bież.)		GBd (weryfikacja końc.)	
minSup[%]	czas[s]	zbiorow	czas us.[s]	zb. us.	czas us.[s]	zb. us.	czas us.[s]	zb. us.	czas[s]	zbiorow	czas us.[s]	zb. us.	czas us.[s]	zb. us.	czas us.[s]	zb. us.
10	82,9	10 704	0,6	1 744	0,6	2 243	0,3	2 805	67,8	10 704	0,6	1 744	0,8	2 243	0,3	2 805
20	41,2	2 754	0,1	235	0,1	237	0,1	625	33,0	2 754	0,1	235	0,0	237	0,1	625
30	22,9	973	0,0	16	0,0	25	0,0	183	19,9	973	0,0	16	0,0	25	0,0	183
40	16,5	429	0,0	3	0,0	0	0,0	63	16,7	429	0,0	3	0,0	0	0,0	63
50	12,5	206	0,0	2	0,0	0	0,0	3	13,6	206	0,0	2	0,0	0	0,0	3
60	12,2	144	0,0	1	0,0	0	0,0	4	12,3	144	0,0	1	0,0	0	0,0	4
70	11,3	127	0,0	1	0,0	0	0,0	0	11,8	127	0,0	1	0,0	0	0,0	0
80	11,4	125	0,0	1	0,0	0	0,0	1	11,4	125	0,0	1	0,0	0	0,0	1
90	10,9	122	0,0	0	0,0	0	0,0	0	11,3	122	0,0	0	0,0	0	0,0	0
99	10,7	119	0,0	0	0,0	0	0,0	0	10,8	119	0,0	0	0,0	0	0,0	0

Mushroom	GR-Apriori		CHARM (listy różnicowe od poziomu 2)				CHARM (tylko listy identyfikatorów)			
minSup[%]	czas[s]	zbiorow	czas[s]	zbiorow	czas us.[s]	zb. us.	czas[s]	zbiorow	czas us.[s]	zb. us.
10	109,0	10 704	43,4	4 885	0,6	1 398	41,2	4 885	2,7	1 398
20	63,0	2 754	25,4	1 197	0,4	198	24,5	1 197	1,5	198
30	44,0	973	18,2	427	0,2	25	16,7	427	0,6	25
40	35,0	429	15,7	140	0,1	2	13,7	140	0,3	2
50	27,0	206	13,6	45	0,1	1	11,8	45	0,2	1
60	18,5	144	12,8	19	0,0	1	11,5	19	0,1	1
70	18,8	127	12,0	12	0,0	1	11,5	12	0,1	1
80	18,6	125	12,1	10	0,1	1	11,3	10	0,1	1
90	12,2	122	11,9	5	0,0	1	11,3	5	0,0	1
99	6,1	119	11,2	1	0,0	1	11,5	1	0,0	1

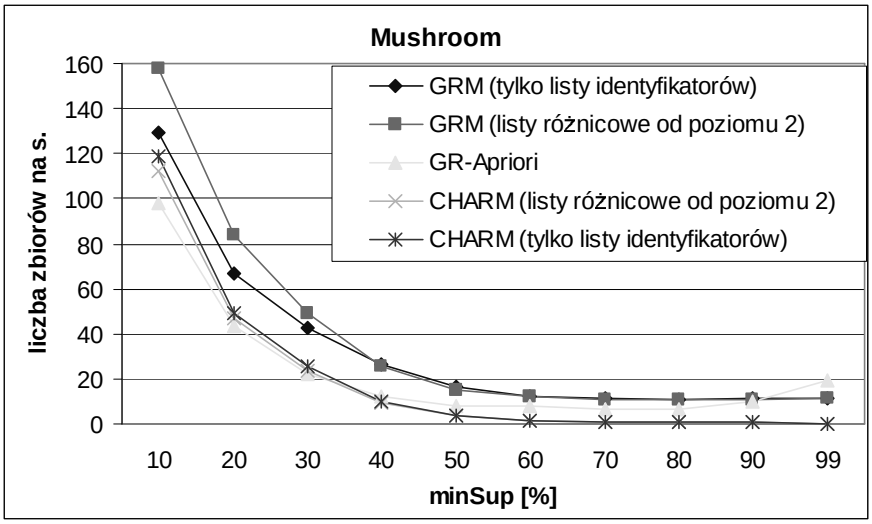
Tabela 2: Wyniki eksperymentalne dla zbioru danych *mushroom* dla algorytmów *GRM*, *GR-Apriori* i *CHARM*

Connect-4	GRM (tylko listy identyfikatorów)								GRM (listy różnicowe od poziomu 2)							
			FG (weryfikacja bież.)		GBd (weryfikacja bież.)		GBd (weryfikacja końc.)				FG (weryfikacja bież.)		GBd (weryfikacja bież.)		GBd (weryfikacja końc.)	
minSup[%]	czas[s]	zbiorow	czas us.[s]	zb. us.	czas us.[s]	zb. us.	czas us.[s]	zb. us.	czas[s]	zbiorow	czas us.[s]	zb. us.	czas us.[s]	zb. us.	czas us.[s]	zb. us.
50									5 338,1	135 150	44,5	0	3,4	0	82,0	12 005
60									3 443,5	71 284	9,6	0	0,5	0	5,1	6 556
70									2 523,2	37 550	3,6	0	0,3	0	1,7	3 367
80	5 770,8	16 084	1,8	0	0,9	0	5,9	1 514	2 186,4	16 084	0,9	0	0,1	0	0,5	1 514
90	1 063,6	3 882	0,1	0	0,0	0	0,2	360	1 246,7	3 882	0,1	0	0,0	0	0,1	360
99	262,2	171	0,0	0	0,0	0	0,7	0	414,6	171	0,0	0	0,0	0	0,0	0

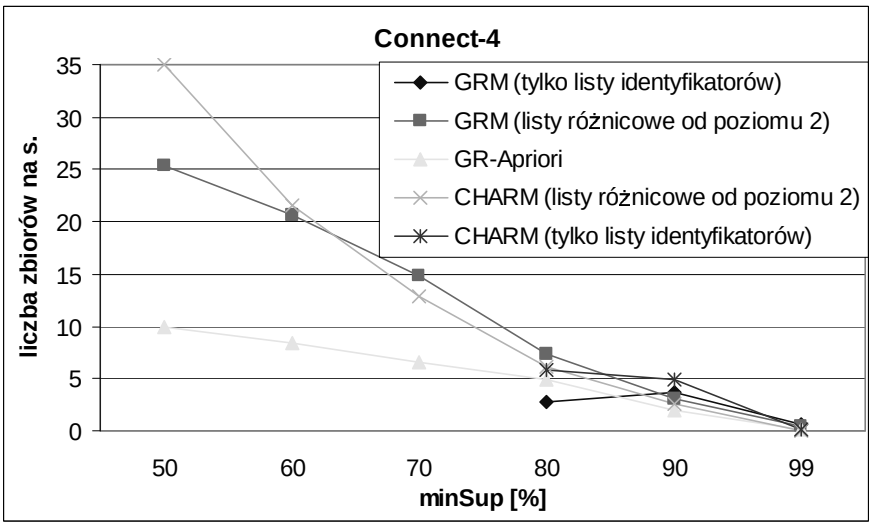
Connect-4	GR-Apriori		CHARM (listy różnicowe od poziomu 2)				CHARM (tylko listy identyfikatorów)			
minSup[%]	czas[s]	zbiorow	czas[s]	zbiorow	czas us.[s]	zb. us.	czas[s]	zbiorow	czas us.[s]	zb. us.
50	13 602,7	135 150	3 725,1	130 102	32,8	0				
60	8 534,3	71 284	3 180,1	68 344	15,9	0				
70	5 679,3	37 550	2 795,2	35 876	7,7	0				
80	3 306,0	16 084	2 475,1	15 108	3,8	0	2 585,1	15 108	506,4	0
90	1 908,0	3 882	1 338,6	3 487	1,7	0	704,0	3 487	110,0	0
99	831,6	171	421,3	29	0,6	0	260,8	29	1,1	0

Tabela 3: Wyniki eksperymentalne dla zbioru danych *connect-4* dla algorytmów *GRM*, *GR-Apriori* i *CHARM*

Udowodniono, że reprezentacja generatorowa nie może być liczniejsza niż rodzina częstych zbiorów zamkniętych [13], dlatego zamiast porównywać czasy trwania algorytmów *GRM* i *CHARM*, które odkrywają różne reprezentacje, sensowniejsze jest porównanie szybkości z jaką algorytmy te odkrywają właściwe dla nich reprezentacje. Wykresy 5 i 6 prezentują średnią liczbę elementów reprezentacji generatorowej znajdowanych przez *GRM* (i *GR-Apriori*) w przeciągu sekundy oraz średnią liczbę częstych zbiorów zamkniętych znajdowanych przez *CHARM* w przeciągu sekundy z danych *mushroom* i *connect-4*. Jak wynika z wykresu 5, *GRM* wykrywa najwięcej elementów reprezentacji na jednostkę czasu ze zbioru danych *mushroom*. Wykres 6 wskazuje, że *GRM* i *CHARM* wykrywają porównywalną liczbę elementów właściwych dla nich reprezentacji z danych *connect-4* dla progów wsparcia nie przekraczających *minSup* = 60%; dla *minSup* = 50% *CHARM* wykrywa 1,38 raza szybciej elementy reprezentacji niż *GRM*.



Wykres 5: Szybkości odkrywania reprezentacji generatorowej przez *GRM* i *GR-Apriori* oraz częstych zbiorów zamkniętych przez *CHARM* na danych *mushroom* w zależności od wartości *minSup*



Wykres 6: Szybkości odkrywania reprezentacji generatorowej przez *GRM* i *GR-Apriori* oraz częstych zbiorów zamkniętych przez *CHARM* na danych *connect-4* w zależności od wartości *minSup*

7 Podsumowanie

W raporcie zaproponowalismy algorytm *GRM* odkrywania reprezentacji generatorowej z wykorzystaniem struktur listowych. Przeprowadzone eksperymenty wskazują, że wraz ze zmniejszaniem wartości progu *minSup* stosowanie list różnicowych od początkowych poziomów drzewa coraz efektywniej skraca czas wykonania algorytmu *GRM*. *GRM* stosujący listy różnicowe szybciej niż *GR-Apriori* odkrywa reprezentację generatorową. Liczba elementów reprezentacji generatorowej znajdowanych w jednostce czasu przez *GRM* bywa większa, mniejsza lub porównywalna z liczbą częstych zbiorów zamkniętych znajdowanych w jednostce czasu przez *CHARM* w zależności od eksplorowanego zbioru danych i wartości progowej.

Literatura

[1] Agrawal R., Imielinski T., Swami A.: Mining Associations Rules between Sets of Items in Large Databases. In: Proc. of the ACM SIGMOD Conference on Management of Data, Washington, USA (1993) 207–216

[2] Agrawal R., Mannila H., Srikant R., Toivonen H., Verkamo A.I.: Fast Discovery of Association Rules. In: Fayyad U.M., Piatetsky-Shapiro G., Smyth P., Uthurusamy R. (eds.): Advances in Knowledge Discovery and Data Mining. AAAI, CA (1996) 307–328

[3] Agrawal R., Srikant R.: Fast Algorithms for Mining Association Rules. In: Proc. of the 20th International Conference on Very Large Databases Conference (VLDB), Santiago, Chile, 1994. Morgan Kaufmann (1994) 487–499

[4] Bastide Y., Pasquier N., Taouil R., Stumme G., Lakhal L.: Mining Minimal Non-redundant Association Rules Using Frequent Closed Itemsets. Computational Logic (2000) 972–986

[5] Bykowski A., Rigotti C.: A Condensed Representation to Find Frequent Patterns. In: Proc. of the 12th ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems (PODS), Santa Barbara, California, USA. ACM (2001) 267–273

[6] Calders, T., and Goethals, B.: Mining All Non-Derivable Frequent Itemsets. In Proceedings, PKDD-02 European Conference on Principles and Practice of Knowledge Discovery in Databases, Helsinki, Finland, August 2002, LNCS 2431, Springer (2002) 74-85

[7] Calders, T., and Goethals, B., 2003, Minimal k-free representations. In Proceedings, PKDD-03 European Conference on Principles and Practice of Knowledge Discovery in Databases, 2003, Cavtat-Dubrovnik, Croatia, pp. 71-82

[8] Ganter B., Wille R.: Formal Concept Analysis, Mathematical Foundations. Springer (1999)

[9] Kryszkiewicz M.: Closed Set Based Discovery of Representative Association Rules. In: Proc. of The Fourth International Symposium on Intelligent Data Analysis (IDA), Lisbon, Portugal, Springer, 2001. Lecture Notes in Computer Science Series, Vol. 2189. Springer (2001) 350–359

[10] Kryszkiewicz M.: Concise Representation of Frequent Patterns based on Disjunction-Free Generators. In: Proc. of the 2001 IEEE International Conference on Data Mining (ICDM), San Jose, California, USA, 2001. IEEE Computer Society (2001) 305–312

[11] Kryszkiewicz M.: Closed Set Based Discovery of Maximal Covering Rules. In: Proc. of Ninth International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU), Annecy, France (2002) 299–306

[12] Kryszkiewicz M.: Concise Representations of Association Rules. In: Proc. of Exploratory Workshop on Pattern Detection and Discovery in Data Mining (ESF), London, UK, 2002. Lecture Notes in Artificial Intelligence, Vol. 2447. Springer-Verlag (2002) 92–109

[13] Kryszkiewicz M.: Concise Representations of Frequent Patterns and Association Rules. Prace Naukowe, Elektronika, Oficyna Wydawnicza Politechniki Warszawskiej, z. 142 (2002)

[14] Kryszkiewicz M.: Reducing Borders of k-Disjunction Free Representations of Frequent Patterns. In: Proc. of ACM Symposium on Applied Computing (SAC) Nikosia, Cyprus, 2004. ACM 1-58113-812-1/3, (2004) 559-563

[15] Kryszkiewicz M. Upper Bound on the Length of Generalized Disjunction Free Patterns. In: Proc. of 16th International Conference on Scientific and Statistical Databases Management (SSDBM) Santorini, Greece, 2004. IEEE Computer Society 1099-3371/04 (2004) 31-40

[16] Kryszkiewicz M., Gajek M.: Concise Representation of Frequent Patterns based on Generalized Disjunction-Free Generators. In: Advances in Knowledge Discovery and Data Mining. Proc. of 6th Pacific-Asia Conference (PAKDD). Taipei, Taiwan, May 2002. Lecture Notes in Computer Science, Vol. 2336. Springer (2002) 159–171

[17] Pasquier N., Bastide Y., Taouil R., Lakhal L.: Discovering Frequent Closed Itemsets for Association Rules. In: Proc. of Database Theory – ICDT '99. Proc. of 7th International Conference (ICDT), Jerusalem, Israel, 1999. Lecture Notes in Computer Science, Vol. 1540. Springer (1999) 398–416

[18] Pei J., Han J., Mao R.: CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. In: Proc. 2000 ACM–SIGMOD DMKD '00, Dallas, TX, USA, 2000. ACM (2000) 21–30

[19] Saquer J., Deogun, JS.: Using Closed Itemsets for Discovering Representative Association Rules. In: Foundations of Intelligent Systems. Proc. of 12th International Symposium (ISMIS), Charlotte, USA, 2000. Lecture Notes in Artificial Intelligence, Vol. 1932. Springer-Verlag (2000) 495–504

[20] Zaki M.J., Gouda K.: Fast Vertical Mining Using Diffsets. Fast vertical mining using diffsets. In: Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, USA, August, 2003, ACM (2003) 326-335

[21] Zaki M.J., Hsiao C.J.: CHARM: An Efficient Algorithm for Closed Itemset Mining. In: Proc. of 2nd SIAM International Conference on Data Mining, Arlington (2002)

[22] Przykładowe zbiory danych: http://fimi.cs.helsinki.fi/data/