Wykrywanie reguł związków w plikach Web Logów

Mariusz Kujawiak[1], **Mieczysław A. Kłopotek**[1]

Streszczenie: Powszechnie istniejące systemy do śledzenia poruszania się użytkowników po serwerze WWW, wykorzystują różne techniki Data Maining. Najczęściej stosowane modele to algorytmy szukające reguł związków (reguły asocjacyjne). Wykrywanie reguł związków jest jedną z najczęściej używanych nieukierunkowanych metod odkrywania wiedzy. W tej pracy proponuje się nowy algorytm wydobywający reguły związków, który porównano z najczęściej stosowanymi algorytmami, między innymi Apriori, AprioriTID, AprioriHybrid i FP-Tree. Przeprowadzone eksperymenty wskazują, że algorytm jest znacznie wydajniejszy od wskazanych algorytmów.

Słowa kluczowe: reguły związków, reguły asocjacyjne, data maining, web log, Apriori, AprioriTID, AprioriHybrid, FP-Tree.

1. Wprowadzenie

Systemy analizy Web Logów są bardzo często wykorzystywane do określenia profilu użytkownika. Najczęściej twórcy albo administratorzy chcą się dowiedzieć poprze analizę Web Logów o zainteresowaniach użytkowników. Daje to możliwość wprowadzania zmian na stronie WWW, w celu zainteresowania użytkownika dodatkowymi informacjami. Przykładem takiego mechanizmu może być sklep internetowy z rekomendacją produktów na podstawie koszyka. Pozwala to użytkownikowi zapoznać się z produktami, które często są kupowane wspólnie z wybranym przez niego przedmiotem. W tym celu z Web Logów wydobywa się reguły związków (reguły asocjacyjne) zawierające informacje na temat często odwiedzanych stronach. Pierwszy algorytm odkrywania reguł związków przedstawiono w roku 1993. W tym samym roku, przedstawiono też algorytm SETM, który w procesie odkrywania reguł związków wykorzystuje operatory relacyjne. W roku 1994 pojawiła się fundamentalna praca Agrawala i Srikanta (Agrawala, Srikanta, 1994), w której przedstawiono dwa nowe algorytmy odkrywania silnych binarnych reguł związków: Apriori i AprioriTID. Algorytmy te stały się, w późniejszym czasie, podstawą wielu nowych algorytmów odkrywania reguł związków. Cechą wspólną wszystkich algorytmów odkrywania reguł związków jest identyczny ogólny schemat działania algorytmu. W pracy przedstawiono algorytmy Apriori, AprioriTID, AprioriHybrid, FP-Tree oraz nowy model znacznie szybszy w artykule nazywany CFP-SFP (Creating Frequent Patterns with Set from Frequent Patterns).

[1] Instytut Informatyki, Akademia Podlaska, Sienkiewicza 51, 08-110 Siedlce
e-mail: {mariusz,klopotek}@ii3.ap.siedlce.pl

2. Czym są reguły związków

Wykrywanie reguł związków jest jedną z najczęściej używanych nieukierunkowanych metod odkrywania wiedzy w Web Logach. Jest to także metoda, której wyniki są jednymi z najłatwiejszych do interpretacji i obrazują to, co większość osób wyobraża sobie jako odkrywanie wiedzy. Proces ten polega na znajdowaniu związków pomiędzy występowaniem grup elementów (atrybutów czy też wartości) w zbiorach danych. Szukane związki mają postać: występowanie określonego wzorca implikuje wystąpienie innego wzorca. Dla znalezionych reguł obliczane są wartości współczynników określających jak silna jest dana reguła i jak wysokie jest prawdopodobieństwo, że wystąpi ona ponownie.

Reguła związku ma postać: $A \Rightarrow B$. Wzorzec A nazywany jest poprzednikiem reguły, a wzorzec B następnikiem reguły. Zarówno lewa, jak i prawa strona składają się z logicznych (prawdziwych lub fałszywych) stwierdzeń lub zdań. Reguły związków określane są przez dwa współczynniki:

- wsparcie (ang. support) - jest to stosunek liczby transakcji z bazy D, które wspierają dany wzorzec do liczby wszystkich transakcji T w bazie D. Formalna definicja wsparcie wygląda następująco:

$$wsparcie(A) = |\{T \in D \mid A \subseteq T\}|/|D| \tag{1}$$

  Przy założeniu, że A i B są wzorcami z bazy danych D, można określić następująca własność wsparcia:

$$A \subseteq B \Rightarrow wsparcie(A) \geq wsparcie(B) \tag{2}$$

  Dla reguły wsparcie określa, w jakiej części transakcji z bazy danych występuje dana zależność i obliczane jest w następujący sposób:

$$wsparcie(A \Rightarrow B) = wsparcie(A \cup B) \tag{3}$$

- zaufanie (ang. confidence) - określa z jakim prawdopodobieństwem występowanie w transakcji poprzednika implikuje wystąpienie następnika. Zaufanie obliczane jest przy użyciu wzoru:

$$zaufanie(A \Rightarrow B) = wsparcie(A \cup B)/wsparcie(A) \tag{4}$$

3. Algorytmy do tworzenia reguł związków

3.1. Algorytm Apriori

Algorytm Apriori zaproponowany przez Agrawala i Srikanta (Agrawala, Srikanta, 1994) jest najczęściej wykorzystywanym algorytmem przy wykrywaniu reguł związków. Polega on na wielokrotnym odczycie danych wejściowych. W trakcie pierwszego odczytu zliczane są wsparcia zbiorów jednoelementowych (czyli pojedynczych elementów), co pozwala określić czy są one częste, tzn. czy spełniają warunek minimalnego wsparcia. Przed każdym kolejnym odczytem danych zbiory wygenerowane

w poprzednim kroku algorytmu wykorzystuje się jako zbiór wejściowy i bazę do generacji kolejnych zbiorów. Nowe zbiory kandydujące wyznaczane są przez rozszerzanie znalezionych zbiorów o pozostałe elementy, które występują w tej transakcji. Obliczane jest wsparcie dla tych zbiorów, następnie określa się, które zbiory są rzeczywiście częste. Są one wykorzystywane do kolejnego cyklu odczytu. Proces ten powtarzany jest do chwili, gdy nie zostanie znaleziony żaden nowy wzorzec częsty.

3.2. Algorytm AprioriTID

Algorytm AprioriTID jest rozszerzeniem algorytmu Apriori o tablicę counting_base. Jest to nowa struktura danych, która jest uaktualniana dla każdego kolejnego kroku. Algorytm AprioriTID oblicza wsparcie dla kandydatów skanując wyłącznie strukturę counting_base.

Ten algorytm jest wydajny w przypadku gdy mamy do czynienia ze zbiorami częstymi o bardzo dużej liczbie elementów. Dlatego wykorzystano tą własność przy budowie algorytmu AprioriHybrid zaproponowany przez Agrawala i Srikanta (Agrawala, Srikanta, 1994), w którym na początku wykonuje się algorytm Apriori a gdy przewiduje się że tablica counting_base zmieści się w pamięci zostaje przełączony na algorytm AprioriTID. Takie podejście nie jest wystarczające do stworzenia wszystkich istotnych reguł dla zadanych parametrów w dość krótkim czasie.

W algorytmie AprioriTID tworzenia reguł związków w początkowym etapie wymaga znacznie więcej czasu. Spowodowane jest to faktem budowania oddzielnej tablicy counting_base, która jest bardzo duża i może się nie mieścić w pamięci. W końcowym etapie budowania reguł związków dostrzega się gwałtowne zmniejszenie struktury counting_base. Powodowane jest to usuwaniem ze zbioru tych częstych wzorców, które nie spełniają założeń minimalnego wsparcia i zaufania.

3.3. Algorytm FP-Tree

Ezeife i Su (Ezeife, Su, 2002) wprowadzili dwa inkrementacyjne algorytmy odkrywania reguł związków: DB-Tree oraz PotFP-Tree, oparte na drzewach częstych wzorców (ang. frequent pattern tree, FP-tree). Pierwszy z algorytmów zapisuje w formie drzewa FP-tree częstości wszystkich atrybutów w zbiorze testowym, dzięki czemu przy modyfikacji danych nie jest wymagane powtórne przetwarzanie całego zbioru, a jedynie analiza samych zmian. Ceną za to jest potencjalnie duży rozmiar drzewa FP-tree w stosunku do metod zapisujących tam jedynie częste wzorce. Drugi z proponowanych algorytmów, PotFP-Tree stanowi rozwiązanie pośrednie, w którym drzewo przechowuje częstości zbiorów aktualnie częstych oraz tych, dla których istnieje duże prawdopodobieństwo, że staną się częste po zmianie danych. Wyznacznikiem tego jest miara średniego poparcia, na którym jest oparta większość procesów odkrywania reguł w pewnym okresie (ang. watermark). Zakładając pewną tolerancję można przyjąć, że wszystkie zbiory, które aktualnie nie są częste, ale ich częstość mieści się w przedziale [t; średnie_poparcie], dla pewnego t, są zbiorami potencjalnie częstymi. Badania eksperymentalne wskazują, że tak skonstruowany algorytm PotFP-Tree zmniejsza liczbę powrotów do uprzednio przetworzonych danych.

Algorytm FP-Tree:

1: **if** $Tree$ contains a single path P **then**
2: **for** each combination γ of the nodes in P **do**
3: generate pattern $\gamma \cup \alpha$ with
support = minimum support of nodes in γ.
4: **end for**
5: **else**
6: **for** each a_i in the header table of $Tree$ **do**
7: generate pattern $\beta = a_i \cup \alpha$ with
$support = a_i.support$
8: construct conditional pattern base for β and conditional FP-tree $Tree$
9: **if** $Tree_\beta \neq \emptyset$ **then**
10: call FP-GROWTH($Tree_\beta$, β)
11: **end if**
12: **end for**
13: **end if**

3.4. Nowy algorytm CFP-SFP do tworzenia reguł związków

Poprzednie rozwiązania skupiały się na poszukiwanie danych w zbiorze głównym. W nowym modelu postanowiono zająć się poszukiwaniem reguł związków w częstych wzorcach stworzonych w poprzedniej iteracji. To posunięcie zmniejsza obszar przeszukiwań tylko do zawężonego już zbioru danych, co daje znacznie szybszy czas tworzenia reguł związków.

Algorytm CFP-SFP:

1: $L_1 = \{$wszystkie częste zbiory jednoelementowe z listą transakcji$\}$
2: $L_1 = \{l \in L_1 \mid l.wsparcie \geq minWsparcie\}$
3: **for** $(k = 2; |L_{k-1}| \neq \emptyset; k++)$ **do begin**
4: **for** $(i = 0; i < |L_{k-1}|; i++)$ **do begin**
5: **for** $(j = 1; j < |L_{k-1}|; j++)$ **do begin**
6: **if** $(l_i \in L_{k-1}$ **and** $l_j \in L_{k-1}$ **takie że** $|l_i \cap l_j| = k)$ **then begin**
7: $c = l_i \cup l_j$
8: **if** $(c \notin L_k)$ **then**
9: L_k dodaj c
10: **end if**
11: **end if**
12: **end for**
13: **end for**
14: **end for**

3.4.1. Przykład działania algorytmu CFP-SFP (min_sup = 2)

W początkowej fazie algorytm CFP-SFP tworzy nową strukturę danych wymaganą do kolejnych iteracji. Przy danych wejściowych z tabeli 1 otrzymujemy zbiór danych wynikowy w tabeli 2.

Tabela 1. Dane wejściowe

TID	Elementy
1	1,3,5,7
2	8,9,2,4
3	6,8,9
4	1,3,5,7
5	8,3,5,6

Z tabela 2 usuwane są dwa elementy 2 i 4 ze względu na brak spełnienia warunku minimalnego wsparcia. Wykonuje się tylko w pierwszej iteracji. Tabela 3 reprezentuje wynik określający częste wzorce jednoelementowe.

Tabela 2. Wynik po pierwszej iteracji

Wzorzec	Transakcje	Wsparcie
1	1,4	2
2	2	1
3	1,4	2
4	2	1
5	1,4,5	3
6	3,5	2
7	1,4	2
8	2,3,5	3
9	2,3	2

Tak przygotowane dane z tabeli 3 są przekazane do następnej iteracji, w której są łączone wzorce jednoelementowe w dwuelementowe na podstawie podobieństwa transakcji. Na przykład w tabeli 3 {1} i {3} są takie same transakcje {1,4}, dlatego można zbudować nowy wzorzec dwuelementowy {1,3}, którego wsparcie wynosi 2. Przekazanie danego wzorca do kolejnej iteracji może być wykonane tylko wtedy gdy będzie spełniony warunek minimalnego wsparcia. Tabela 4 prezentuje wynik tej iteracji.

W kolejnej iteracji są szukane wzorce trzyelementowe w tym przypadku poszukujemy dwóch wspólnych elementów częstych wzorców. Z danych z tabeli 4 pobieramy pierwszy wzorzec {1,3} i sprawdzamy z pozostałymi: {1,3} i {1,5} -> część wspólna {1}, dlatego można zbudować wzorzec o następującej postaci {1,3,5}, przy

Tabela 3. Wynik po pierwszej iteracji z usunięciem wzorców $<$ min_sup

Wzorzec	Transakcje	Wsparcie
1	1,4	2
3	1,4	2
5	1,4,5	3
6	3,5	2
7	1,4	2
8	2,3,5	3
9	2,3	2

Tabela 4. Wynik po drugiej iteracji

Wzorzec	Transakcje	Wsparcie
1,3	1,4	2
1,5	1,4	2
1,7	1,4	2
3,5	1,4	2
3,7	1,4	2
5,7	1,4	2
6,8	3,5	2
8,9	2,3	2

założeniu, że będzie spełniony warunek minimalnego wsparcia. Tabela 3 zawiera wszystkie wzorce trzyelementowe.

Otrzymany zbiór poddajemy kolejnej analizie w poszukiwaniu wzorca czteroelementowego w tym przypadku można zbudować tylko jedne taki element {1,3,5,7} o wsparciu 2 przez atrybuty {1,4}. Następuje koniec algorytmu gdy ze zbioru poprzedniego nie uda się zbudować żadnego wzorca w kolejnym kroku.

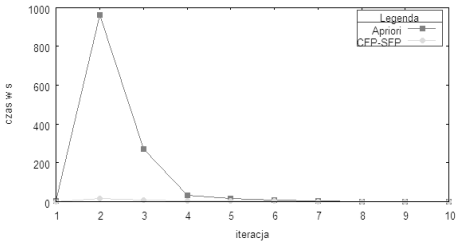
4. Badania

Analiza porównawcza została przeprowadzona na komputerze z procesorem Intel Pentium Mobile 1,5 MHz z 1024 MB pamięci operacyjnej. Przeprowadzona badania na zbiorze: T10I4D100K.dat [2]. Zostały również przeprowadzono badania na części pliku T10I4D100K.dat w celu przedstawienia różnic w wydajności w przypadku algorytmu AprioriTID i Apriori. Wszystkie implementacje zostały wykonane w języku Java.

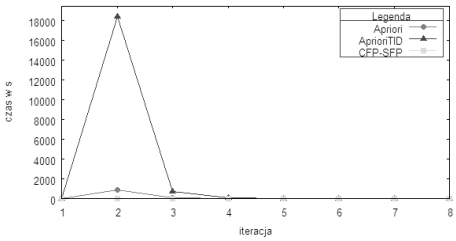
[2] Zestawy danych użyty w badaniach jest publicznie dostępne na stronie WWW: http://http://fimi.cs.helsinki.fi/data/

Tabela 5. Wynik po trzeciej iteracji

Wzorzec	Transakcje	Wsparcie
1,3,5	1,4	2
1,3,7	1,4	2
1,5,7	1,4	2
3,5,7	1,4	2



Rysunek 1. Rozkład czasu wykonania algorytmów przy wsparciu 15



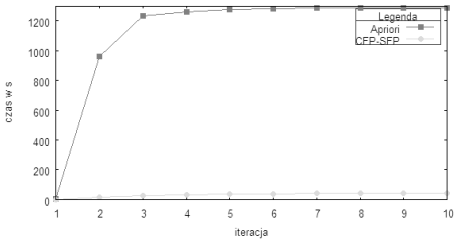
Rysunek 2. Rozkład czasu wykonania algorytmów przy wsparciu 25

4.1. Wyniki dla 10000 wierszy z pliku T10I4D100K.dat

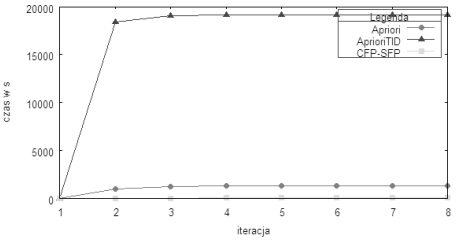
W przypadku algorytmu Apriori można zauważyć na wykresie 1, że czas wykonania drugiej iteracji jest bardzo długi. Wynika to z tego że jest przeszukiwany cały zbiór danych wejściowych. W algorytmie Apriori następuje powolne schodzenie w dół, co oznacza że przeszukiwanie całego zbioru nadal bardzo opóźnia w otrzymaniu wyniku. Proponowany algorytm CFP-SFP w drugiej iteracji na wykresie 1 ma bardzo mały zbiór wejściowy. Wynika to z tego, że wzorców jednoelementowych jest dużo mniej niż wielkość całego zbioru. W przypadku algorytmu AprioriTID na wykresie czas wykonania drugiej iteracji jest wyraźnie większy od wykonania się algorytmu Apriori, dlatego w tej części nie wykorzystuje się AprioriTID w algorytmie AprioriHybrid.

Wyniki prezentowane na wykresie 3 wskazują, że algorytm Apriori w początkowym etapie traci bardzo dużo czasu na wykonanie obliczeń. Proponowany algorytm CFP-SFP eliminuje tą wadę. Po iteracji 4 oba algorytmy mają dość zbliżony przyrost czasu. Na wykresie 4 przedstawiono przebieg czasowy algorytmu AprioriTID w porównaniu z algorytmem Apriori i CFP-SFP. Algorytm AprioriTID wymaga długiego czasu wykonanie iteracji 2, kolejne iteracje wykonywane są w krótszym czasie.

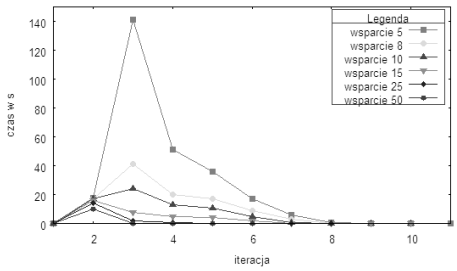
Na wykresach 5 i 6 czas wykonania iteracji 2 nieznacznie się różni przy różnych minimalnych wsparciach. Istotne różnice są w iteracjach od 3 do 5 wynika to stąd, że zbiór wejściowy dla tych iteracji znacznie się rozszerzył.



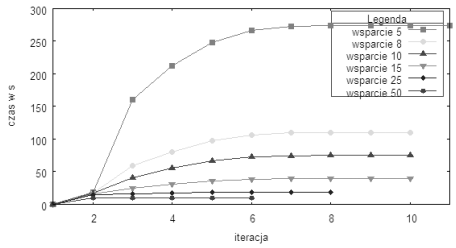
Rysunek 3. Przebieg czasu wykonania algorytmów przy wsparciu 15



Rysunek 4. Przebieg czasu wykonania algorytmów przy wsparciu 25



Rysunek 5. Rozkład czasu wykonania algorytmu CFP-SFP przy różnych współczynnikach wsparcia



Rysunek 6. Przebieg czasu wykonania algorytmu CFP-SFP przy różnych współczynnikach wsparcia

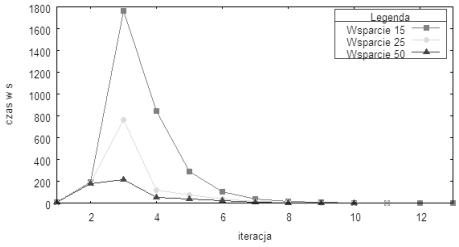
4.2. Wyniki dla całego pliku T10I4D100K.dat

Algorytm CFP-SFP bardzo dobrze sobie radzi z tak dużym zbiorem danych przy tak mało wydajnym procesorze. Jak widać na wykresach 7 i 8 czas wykonania istotnie się wydłuża przy bardzo małym minimalnym wsparciu. Powodowane jest to znaczącym przyrostem wzorców na każdym z poziomów. Jak łatwo można zauważyć od iteracji 7 czas wykonania nieznacznie wypływa na ogólny czas wykonania algorytmu, to jest powodowane faktem że zbiór danych wejściowy dla tych iteracji znacznie się zmniejszył. W iteracji 2 możemy zauważyć że jest to identyczny czas wykonania w każdym z przypadków. Oznacza, to że pierwsza iteracja zwraca dość podobny zbiór.

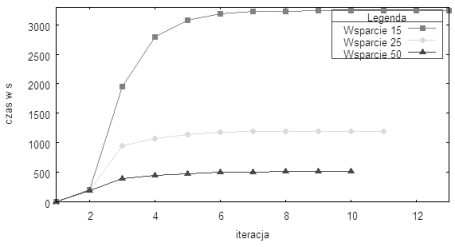
4.3. Porównanie algorytmów z FP-Tree na pliku T10I4D100K.dat

Algorytm CFP-SFP na wykresie 9 charakteryzuje się dużą wydajnością w zakresie minimalnego wsparcia od 5 do 25. Algorytm FP-Tree[3] w tym przedziale minimalnego wsparcia musi zbudować znacznie większe drzewo. Algorytm Apriori w tym przedziale także charakteryzuje się szybszym tworzeniem wszystkich częstych wzor-

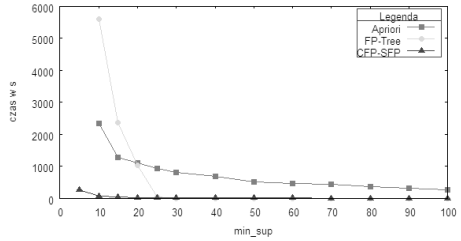
[3] implementacja algorytmu w języku java ze strony WWW: http://www.csc.liv.ac.uk/ frans/KDD/Software/



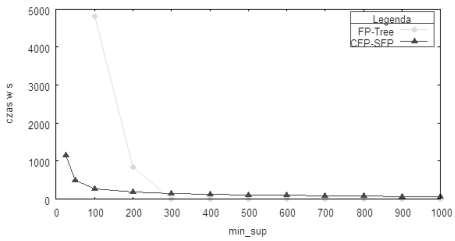
Rysunek 7. Rozkład czasu wykonania algorytmu CFP-SFP przy różnych współczynnikach wsparcia (15, 25 i 50)



Rysunek 8. Przebieg czasu wykonania algorytmu CFP-SFP przy różnych współczynnikach wsparcia (15, 25 i 50)



Rysunek 9. Czas wykonania algorytmów przy różnych minimalnych wsparciach na 10000 wierszy z pliku T10I4D100K.dat



Rysunek 10. Czas wykonania algorytmów przy różnych minimalnych wsparciach na całym pliku T10I4D100K.dat

ców. Można wnioskować, że FP-Tree jest bardzo wydajny przy dużych wartościach minimalnego wsparcia. Dobrze prezentuje to wykres 10, na którym w przedziale od 300 do 1000 minimalnego wsparcia FP-Tree jest nieznacznie wydajniejszy od proponowanego algorytmu CFP-SFP. Algorytm Apriori przy bardzo dużych zbiorach danych, wykres 10, nie jest w stanie w rozsądnym czasie wygenerować częste wzorce, dlatego nie zostały zaprezentowane wyniki dla tego algorytmu przy pełnym zbirze danych.

5. Podsumowanie

Zaproponowany algorytm CFP-SFP jest wydajniejszy przy różnych minimalnych wsparciach od pozostałych algorytmów. W przypadku algorytmu FP-Tree można zauważyć, że przy dużych minimalnych wsparciach algorytm ten wykazuje dużą wydajność. Lecz wraz ze zmniejszaniem minimalnego wsparcia ten algorytm zaczyna drastycznie długo tworzyć drzewo częstych wzorców. Proponowany algorytm CFP-SFP w przypadku dużych minimalnych wsparć jest wolniejszy od FP-Tree, ponieważ konstrukcja drzewa w algorytmie FP-Tree jest znacznie mniejsza i łatwiejsza do przejrzenia.

Porównania CFP-SFP z algorytmem Apriori wskazuje na dużą wydajność w

różnych zakresach minimalnego wsparć. Powodem takiej wydajności jest wykorzystanie zbioru wynikowego z poprzedniej iteracji. Takie posunięcie daje znacznie szybsze wykonanie pierwszych iteracji, ale wraz z rozszerzaniem się zbioru danych po kolejnych iteracjach czas wykonania tych iteracji istotnie się zwiększa. Powodem takiego zachowania jest fakt przeglądania całego zbioru w poszukiwaniu możliwych złączeń zbiorów częstych. Pomimo takiej wady i tak zysk jaki otrzymujemy przy pierwszych iteracjach znacząco zwiększa wydajność przy tworzeniu wszystkich częstych wzorców.

Planowane jest wykonanie metody grupującej częste wzorce w celu zmniejszenia liczby przeszukiwań dla każdej z iteracji. Ponadto planowane jest wykonanie zrównoleglenia algorytmu z wykorzystaniem grupowania. Te posunięcia pozwolą zwiększyć wydajność na maszynach wieloprocesorowych, które w obecnym czasie są dość popularne i ogólnie dostępne.

Literatura

Agrawal, R., Imielinski, T., Swami AN. (1993) Mining Association Rules between Sets of Items in Large Databases. *SIGMOD*, 22(2), 207–16.

Agrawal, R., Srikant R. (1994) Fast Algorithms for Mining Association Rules. *VLDB*, Chile, ISBN 1-55860-153-8.

Mannila H., Toivonen H., Verkamo A.I. (1994) Efficient algorithms for discovering association rules. *AAAI Workshop on Knowledge Discovery in Databases (SIGKDD)*, Seattle, 181-92.

Ezeife C.I., Su Y. (2002) Mining Incremental Association Rules with Generalized FP-tree *Proceedings of the Fithteenth Canadian Conference on Artificial Intelligence (AI 2002)*, Calgary, Canada, Lecture Notes in Computer Science (LNCS) , Springer-Verlag, Berlin Heidelberg 2002.

Discovering association rules in Web Log files

Existing systems for tracking the user's movements at the WWW server use various Data Mining techniques. The most frequently used models are algorithms searching for association rules. Discovering association rules is one of the most popular, non-oriented methods of knowledge discovery.
This paper proposes a new algorithm extracting association rules. This algorithm was compared with the most often used algorithms i.a. Apriori, AprioriTID, AprioriHybrid and FP-Tree. Conducted experiments show that this new algorithm is much more efficient than the above mentioned algorithms.