Wykrywanie wzorców częstych w grafach

Anna Seweryn
a.seweryn@stud.elka.pw.edu.pl
Politechnika Warszawska
Wydział Elektroniki i Nauk Technicznych
Instytut Informatyki

Streszczenie

Poniższy artykuł przedstawia analizę algorytmów wykrywania wzorców częstych oraz podstawowe pojęcia eksploracji danych. Szczegółowo opisuje algorytm wykrywania wzorców częstych dla grafów skierowanych ważonych. Przedstawia zastosowanie praktyczne algorytmu oraz testy wydajnościowe.

1. Wstęp

Postęp w zdobywaniu danych cyfrowych oraz techniki ich gromadzenia przyczyniły się do powstania olbrzymich baz danych. Jednocześnie wzrosło zainteresowanie na wykorzystywanie zgromadzonej wiedzy, która mogłaby być bardzo cenna dla właścicieli tych danych. Dynamicznie rozwijającą się dziedziną stała się eksploracja danych, która zajmuje się: przesiewaniem bazy danych, podsumowuje dane i szuka wzorców.

Kiedy mamy tak olbrzymią ilość danych pojawiają się problemy związane z magazynowaniem, dostępem do danych oraz jak analizować dane w rozsądnym przedziale czasowym. Następne problemy to brakujące wartości, zanieczyszczenia i nieprawdziwe dane oraz w jaki sposób zdecydować czy dana zależność jest zjawiskiem przypadkowym czy odzwierciedla wewnętrzną rzeczywistość.

Bardzo ważne jest również dobranie odpowiedniej struktury danych tak, aby zachować wszystkie informacje o danych. Dane nie zawsze są w postaci płaskiej, mogą występować w różnych strukturach i mogą istnieć między nimi złożone zależności. Im bardziej złożone są struktury danych, tym bardziej złożone modele eksploracji danych, algorytmy i narzędzia, trzeba zastosować.

Wykrywanie wzorców częstych borykało się z problemem dostosowania algorytmów do złożonych struktur danych. Jak poprawnie przechowywać dane żeby zachować zależności między nimi?

Trywialnym rozwiązaniem było przeszukiwanie całych zbiorów danych i element po elemencie ocenianie czy jest to zbiór częsty. Następnie zaczęto stosować algorytm Apriori, który miał zastosowanie tylko do zbiorów danych. Zaczęto przetwarzać dane grafowe do postaci płaskiej tracąc podczas tego przekształcania ważne informacje na temat zależności między elementami, poza tym miało to ograniczenia, odnośnie unikalnych etykiet i zbiorów wierzchołków – okazało się całkowicie nie praktycznym rozwiązaniem. Zaczęto stosować metodyki heurystyczne i chciwe. Jednak one nie potrafiły wykrywać wszystkich wzorców częstych, co powodowało utratę czasem bardzo ważnych informacji. Potem zaczęto stosować równy podział sekwencji jednak miało to istotną wadę gdyż liczyła się kolejność dodawania wierzchołków. Zaczęły powstały systemy wykorzystujące wyszukiwanie częstych wzorców takie jak: WARM oraz FARMAR. W końcu Inokuchi, Washio, Motoda zaproponowali algorytm dla struktur grafowych, gdzie nie ma znaczenia czy graf jest skierowany oraz czy jest etykietowany. Obecnie algorytm ten jest wykorzystywany przez „Recruit Co” do analizy strony: „Achaja NAVI”, aby wykrywać częste zachowania użytkowników na stronach serwisu oraz przez Uniwersytet Oxford do odnajdywania substancji rakotwórczej w substancjach.

W rozdziale 2 zostaną przedstawione podstawowe pojęcia związane z eksploracją danych. Rozdział 3 poświęcony jest algorytmom wyszukującym wzorce częste natomiast rozdział 4 opisuje algorytm Inokuchi, Washio i Motody wyszukujący wzorce częste dla grafów skierowanych ważonych. Kolejny rozdział przedstawia wyniki testów wydajnościowych dla opisywanego algorytmu zaś rozdział 6 opisuje praktyczne zastosowania algorytmu.

2. Eksploracja danych

Eksploracja danych (data mining) – nauka o wydobywaniu informacji z dużych zbiorów danych lub baz danych. Jest to analiza zbioru danych w celu znalezienia nieoczekiwanych związków i podsumowania danych w oryginalny sposób, aby były zrozumiałe i przydatne. Zależności i podsumowania będące wynikami eksploracji danych zwane są modelami lub wzorcami.

Pojęcie eksploracji danych potocznie rozumiane jest jako odkrywanie wiedzy w bazach danych, integruje ono kilka dziedzin takich jak: statystyka, systemy baz danych, sztuczna inteligencja, optymalizacja, obliczenia równoległe.

Zainteresowanie wynika z faktu, iż przedsiębiorstwa chcą w efektywny i racjonalny sposób wykorzystywać nagromadzoną wiedzę z baz danych dla celów wspomagających ich działalność. Tradycyjny sposób korzystania z baz danych sprowadza się do realizacji zapytań poprzez aplikację lub raporty. Dysponując danymi w bazie danych opisującą działalność supermarketu możemy postawić szereg pytań np.: „ W jaki sposób wykorzystać dane do usprawnienia funkcjonowania firmy?", " Czym różnią się klienci supermarketu w Warszawie a czym w Radomiu?", "Jakie produkty kupują najczęściej klienci razem z chlebem?", „Czy można przewidzieć dalsze zachowania klientów?"

Odpowiedzi na te pytania daje nam eksploracja danych, która na podstawie analizy zbiorów danych obserwacyjnych znajduje nieoczekiwane związki i podsumowanie danych w zrozumiały sposób.

Przed rozpoczęciem omawiania algorytmów wyszukiwania wzorców częstych zapoznajmy się z przydatnymi definicjami.

Częstość fr(Q) zadanego wzorca zbioru elementów Q jest liczbą przypadków w danych spełniających Q. Częstość $fr(Q \cap \alpha)$ jest czasami nazywana wsparciem. Dokładność (ufność) zadanej reguły to frakcja (część wszystkich) wierszy spełniających wśród tych wierszy, które spełniają Q.

Zadanie znajdowania wzorców częstych zbiorów elementów jest proste: mając dany próg częstościowy s znajdujemy wszystkie wzorce zbiorów elementów, które są częste i ich częstości. Jeśli próg częstości jest niski może być wiele zbiorów częstych.

3. Algorytmy wyszukujące wzorce częste

Zacznijmy od opisu **metody trywialnej** wyszukiwania wzorców częstych, która by polegała na braniu po kolei każdego wzorca i sprawdzanie czy występuje on w danych i czy jest w jakimś sensie znaczący. Jeśli liczba możliwych wzorców jest mała, to wówczas można tę metodę zastosować, ale z reguły jest ona całkowicie niewykonalna.

Algorytm Apriori może wyciągnąć istotne zdarzenia w skuteczny sposób, ale struktura danych jest ograniczona do elementów zbioru. Znalezienie wszystkich wzorców

częstych zaczynamy od znalezienia wszystkich zbiorów częstych składających się z 1 zmiennej. Zakładając, że już je znamy, budujemy zbiory kandydujące rozmiaru 2: zbiory {A,B} takie, że {A} jest częsty i {B} jest częsty. Po zbudowaniu przeglądamy je i stwierdzamy, które są częste. Budujemy z nich zbiory kandydujące rozmiaru 3, których częstość jest następnie obliczana z danych i tak dalej.

Inokuchi, Washio, & Metoda w 1999 roku zasugerowali, że algorytm Apriori można stosować do struktur grafowych po prostym przetworzeniu danych. Jednakże ich zastosowanie ma pewne ograniczenia, np.: wszystkie wierzchołki muszą być różne oraz mają odmienne etykiety – jednakże to rozwiązanie nie jest skuteczne dla odkrywania wzorców częstych w grafach, gdzie wielokrotne wierzchołki mają te same etykiety.

SUBDUE to inne podejście wyszukiwania wzorców częstych grafu przez ściskanie oryginalnych wykresów wg zasady MDL. Jednakże te podejścia nie sprawdziły się wobec złożoności obliczeniowej. Mogą one omijać jakieś znaczące wzorce, ponieważ przyjęte strategia wyszukiwań są chciwe lub heurystyczne.

Dlatego szukano innych metod niewykorzystujących chciwych przeszukiwań i tak **De Raedt i Kramer** zaproponowali równe rozmieszczenie sekwencji do eksploracji wzorców – kolejność związków węzłów, związki charakteryzowały połączenia monotoniczne i antymonotoniczne mierzone jako częstość (frequency) i powszechność (generality). Mimo, że ta metoda używa rozmaitych miar i unika złożoności obliczeniowej ma ograniczenie do kolejności wzorców wstawianych w grafach.

Geibel i Wysotzki zaproponowali metodę by wprowadzić wywołanie podgrafu jako cechy, by sklasyfikować w grafach dane. Wywołany podgraf grafu G ma podzbiór węzłów grafu G i te same powiązania między węzłami. Ograniczeniem tej metody jest wyższa liczba węzłów, co prowadzi do wykładniczej eksplozji obliczeniowej czasu.

Pierwszym systemem wykorzystującym kompletne przeszukiwanie szerokiej klasy z częstymi strukturami w grafie jest **WARMR** proponowany przez Dohespe i Toiromen (1998). Oni zastosowali **ILP (inductire logic programming)** zgodnie z algorytmem Apriori. To podejście stawia czoło wysokiej złożoności obliczeniowej. By zmniejszyć złożoność obliczeniową stosowane są heurystyki by przyciąć badaną przestrzeń. Kolejnym systemem był **FARMAR**, który ma słabszy warunek równoważności, by przyspieszyć wyszukiwanie wzorców częstych. Bazuje on na strukturze TRIE a ścieżka od korzenia do liścia reprezentuje całe zapytanie.

Inokuchi, Washio i Motoda zaproponowali reprezentację grafu za pomocą macierzy sąsiedztwa oraz kodowania macierzy, co pozwala na wyszukiwanie wzorców w grafach ważonych z pętlami.

4. Algorytm Inokuchi, Washio i Motody

Dla rozszerzenia algorytmu Apriori zastosowano w algorytmie macierz sąsiedztwa do przechowywania struktury grafu. Zaproponowany algorytm wyszukuje częste podgrafy. Bazuje on również na pojęciach zaufania i częstości. Macierz sąsiedztwa i macierz kodowa są strukturami odpowiednimi do reprezentacji grafu, który zawiera etykietowane węzły, ale nie odpowiedni dla grafów ważonych z pętlami. Rzędy i kolumny odpowiadają węzłom w grafie. Kod macierzy to elementy leżące na przekątnej macierzy. Dla każdej macierzy wprowadzany jest kod. Zmniejsza to wymagania pamięci. Grafy są reprezentowane przez kody ich macierzy sąsiedztwa. Macierz sąsiedztwa nie jest unikalna.

Struktura grafu

Kiedy element jest transakcją dla algorytmu Apriori, graf tworzy transakcję. Graf z danych zostaje przetransformatowany do macierzy sąsiedztwa. Każdy rząd i kolumna macierzy są zgodne z węzłem i, jeżeli jest związek między węzłem i oraz j to w macierzy sąsiedztwa na pozycji ij wpisujemy 1 (nasza transakcja ma wartość 1), w przeciwnym wypadku wartość 0. Dalej reprezentujemy etykietę węzła jako Ng i etykietę związku jako Lh.
Definiujemy wsparcie, zaufanie i regułę asocjacyjną:

$$sup(G) = \frac{number\ of\ transactions\ including\ an\ induced\ subgraph\ G}{total\ number\ of\ transactions},$$

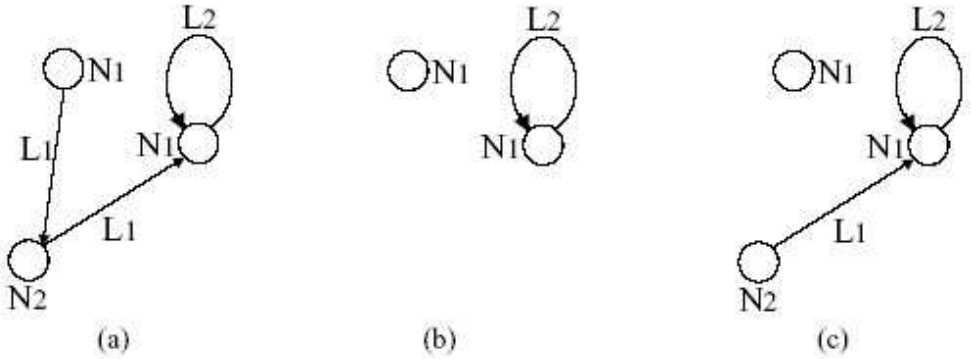
$$sup(B \Rightarrow H) = sup(B \cup H), \quad conf(B \Rightarrow H) = \frac{sup(B \cup H)}{sup(B)},$$

Gdzie B i H wywołane podgrafy transakcji i B lub H jest grafem utworzonym przez połączenie węzłów i związków należących do B i H. B=>H wskazuje, że H jest włączony do grafu transakcji pod działaniem zaufania B=>H jeśli B zawiera się w H.

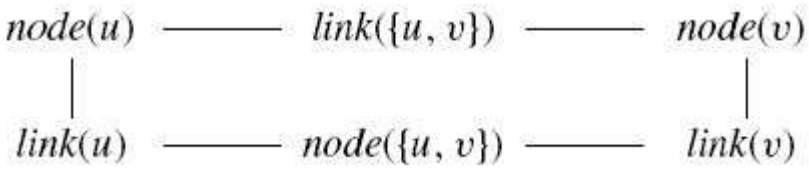
Poniżej formalna definicja wywołania podgrafu:

$$V(G') \subseteq V(G), \quad E(G') \subseteq E(G) \quad \text{and}$$
$$\forall u, v \in V(G') \ \{u, v\} \in E(G) \Leftrightarrow \{u, v\} \in E(G').$$

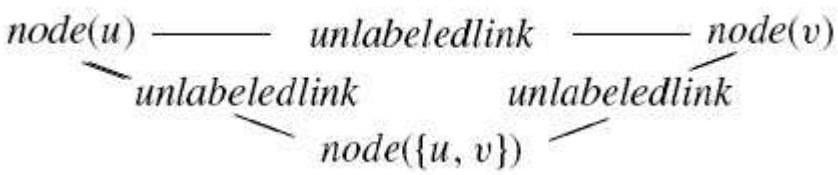
Inaczej mówiąc wywołany podgraf grafu G ma podzbiór węzłów z G i te same związki między parami węzłów jak w przykładzie: dla podgrafu (b) jest wywołany podgraf wykresu (a), ale podgraf (c) jest ogólnym podgrafem:



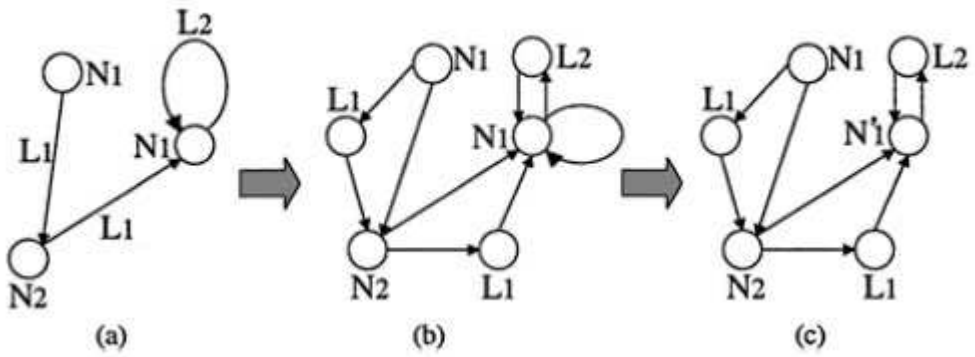
Jeśli graf ma etykietowane związki to w podgrafie również muszą być etykietowane te związki, które do niego należą. Mając parę węzłów u,v skierowaną lub niekierowaną. Związki stają się węzłami i węzły stają się związkami:



Informacja link() –etykieta może zostać usunięta i otrzymujemy zredukowany graf, gdzie oryginalna informacja o związku i węźle zostaje zachowana:



Ta operacja pozwala zmienić oryginalny graf ważony-etykietowany do postaci nie etykietowanego grafy, ale zachowując wszystkie informacje o topologii grafu, usuwa bezpośredni związek między węzłami:



Odpowiednio transformacja macierzy sąsiedztwa wygląda tak dla rysunku (b):

$$\begin{array}{c} \\ N_1 \\ N_1 \\ N_2 \end{array} \begin{array}{c} \begin{array}{ccc} N_1 & N_1 & N_2 \end{array} \\ \begin{pmatrix} L_2 & 0 & 0 \\ 0 & 0 & L_1 \\ L_1 & 0 & 0 \end{pmatrix} \end{array} \Rightarrow \begin{array}{c} \\ N_1 \\ N_1 \\ N_2 \\ L_1 \\ L_1 \\ L_2 \end{array} \begin{array}{c} \begin{array}{cccccc} N_1 & N_1 & N_2 & L_1 & L_1 & L_2 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

Kiedy jest pętla ważona tak jak w N1, zmienia się etykieta węzła na N'1 tak jak na rysunku (c), odtąd macierz wygląda tak:

$$\begin{array}{c} \\ N_1' \\ N_1 \\ N_2 \\ L_1 \\ L_1 \\ L_2 \end{array} \begin{array}{c} \begin{array}{cccccc} N_1' & N_1 & N_2 & L_1 & L_1 & L_2 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

Fakt ten pozwala na zapisywanie różnych etykiet dla węzłów, które mają wiele pętli. Zapisywanie etykiet węzłów odbywa się wg zasady:

$$N_1 < \cdots < N_{p-r} < \cdots < N_1' < \cdots < N_r' < \cdots < L_1 < \cdots < L_q,$$

Gdzie r jest liczą etykiet węzłów z pętlami, poprzez zastosowanie tego uporządkowania otrzymujemy macierz sąsiedztwa w postaci:

$$\begin{array}{c} \\ N_1 \\ N_2 \\ N_1' \\ L_1 \\ L_1 \\ L_2 \end{array} \begin{array}{c} \begin{array}{cccccc} N_1 & N_2 & N_1' & L_1 & L_1 & L_2 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array},$$

Dla grafów niekierowanych ten proces wygląda identycznie, jedyną różnicą jest to, że macierz ma postać symetryczną.

Dla wydajności algorytmu definiujemy kody macierzy sąsiedztwa. Dla grafu nieskierowanego otrzymujemy kody przez skanowanie wyższych trójkątnych elementów. Powstaje kod 01 według poniższego schematu:

$$X_k = \begin{pmatrix} 0 & x_{1,2} & x_{1,3} & \cdots & x_{1,k} \\ x_{2,1} & 0 & x_{2,3} & \cdots & x_{2,k} \\ x_{3,1} & x_{3,2} & 0 & \cdots & x_{3,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{k,1} & x_{k,2} & x_{k,3} & \cdots & 0 \end{pmatrix},$$

$$code(X_k) = x_{1,2}x_{1,3}x_{2,3}x_{1,4}\cdots x_{k-2,k}x_{k-1,k}.$$

Dla macierzy:

$$\begin{array}{c} \\ N_1 \\ N_2 \\ N_3 \\ N_3 \\ N_4 \end{array} \begin{array}{c} \begin{array}{ccccc} N_1 & N_2 & N_3 & N_3 & N_4 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}.$$

Otrzymujemy kod: code(Xk)=0101100101.

Dla grafu skierowanego, jeżeli oba elementy (i,j) oraz (j,i) są zerami to dla kodu wstawiamy 0, natomiast, gdy: element (i,j) ma wartość 1 a element (j,i) ma wartość 0 to wstawiamy 1, w przypadku gdy element (i,j) ma wartość 0 a (j,i) wartość 1 to w kod w stawiamy 2, w końcu, gdy element (i,j) i (j,i) mają wartości 1 to w kodzie wstawiamy 3.
Wzór:

$$code(X_k) = c_{1,2}c_{1,3}c_{2,3}c_{1,4}\cdots c_{k-2,k}c_{k-1,k},$$

gdzie:

$$c_{i,j} = 2x_{j,i} + x_{i,j}.$$

Dla macierzy

$$\begin{array}{c} \\ N_1 \\ N_2 \\ N_1' \\ L_1 \\ L_1 \\ L_2 \end{array} \begin{array}{c} \begin{array}{cccccc} N_1 & N_2 & N_1' & L_1 & L_1 & L_2 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array},$$

kod jest następujący: code(Xk)=101012120000300

Algorytm wyszukiwania częstego wzorca

1. Generowanie kandydata na częsty wzorzec:
Xk i Yk – macierze sąsiedztwa dwóch podrafów G(Xk) i G(Yk). Jeśli dwa częste podgrafy mają równe elementy wyjątkiem k-tej kolumny i k-tego wiersza to wtedy łączy się i tworzy się podgraf Zk+1:

$$X_k = \begin{pmatrix} X_{k-1} & \boldsymbol{x}_1 \\ \boldsymbol{x}_2^T & 0 \end{pmatrix}, \quad Y_k = \begin{pmatrix} X_{k-1} & \boldsymbol{y}_1 \\ \boldsymbol{y}_2^T & 0 \end{pmatrix},$$

$$Z_{k+1} = \begin{pmatrix} X_{k-1} & \boldsymbol{x}_1 & \boldsymbol{y}_1 \\ \boldsymbol{x}_2^T & 0 & z_{k,k+1} \\ \boldsymbol{y}_2^T & z_{k+1,k} & 0 \end{pmatrix} = \left(\begin{array}{cc|c} & & \boldsymbol{y}_1 \\ \multicolumn{2}{c|}{X_k} & z_{k,k+1} \\ \hline \boldsymbol{y}_2^T & z_{k+1,k} & 0 \end{array}\right),$$

Gdzie Xk-1 jest macierzą sąsiedztwa grafu wielkości K-1

Relacje zachodzące między macierzami sąsiedztwa Xk oraz Yk:

$$N(X_k, i) = N(Y_k, i) = N(Z_{k+1}, i) \quad \text{and}$$
$$N(X_k, i) \le N(X_k, i+1) \quad \text{for } i = 1, \ldots, k-1.$$
$$N(X_k, k) = N(Z_{k+1}, k), \quad N(Y_k, k) = N(Z_{k+1}, k+1), \quad \text{and } N(X_k, k) \le N(Y_k, k).$$

Tutaj elementy (k, k+1) oraz (k+1, k) macierz sąsiedztwa Z k+1 nie są określone przez Xk oraz Yk. W przypadku grafu nie skierowanego rozważane są 2 przypadki:

(1) jest związek między k-węzłem i k+1-węzłem grafu G(Zk+1) oraz

(2) nie ma żadnego związku między nimi.

Odpowiednio tym dwóm przypadkom generujemy 2 macierze sąsiedztwa wstawiając odpowiednio 0 lub 1 dla (k, k+1)-elementu oraz (k+1, k)-elementu.

W przypadku skierowanego grafu możliwe są 4 przypadki:

(1) skierowany związek z węzła k do k+1 grafu G(Zk+1) oraz

(2) skierowany związek od k+1 do k oraz

(3)są dwukierunkowe związki między nimi oraz

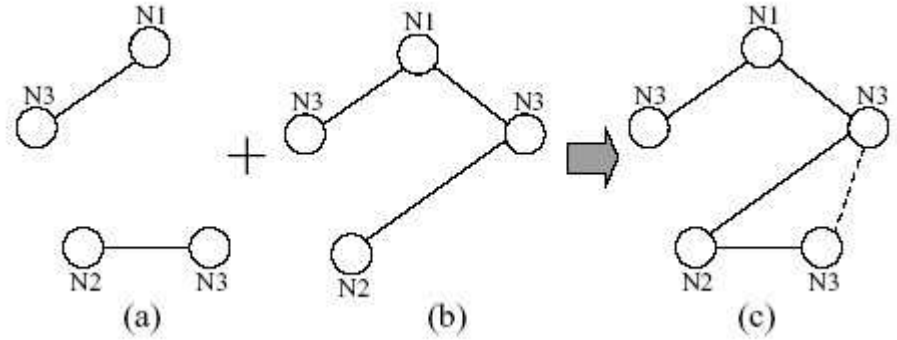
(4) nie ma żadnych związków między nimi.

Odpowiednio 4 różne macierze graniczne są generowane. Nazywamy odpowiednio Xk i Yk "pierwsza macierz" i "druga macierz" by wygenerować kolejno Zk + 1

$$code(\text{the first matrix}) \le code(\text{the second matrix}).$$

łączymy 2 macierze przez łączenie operacji (jon operation) i powstaje normalna forma macierzy sąsiedztwa.

Przykład łączenia macierzy(a) i (b):



Rozważmy przykład:

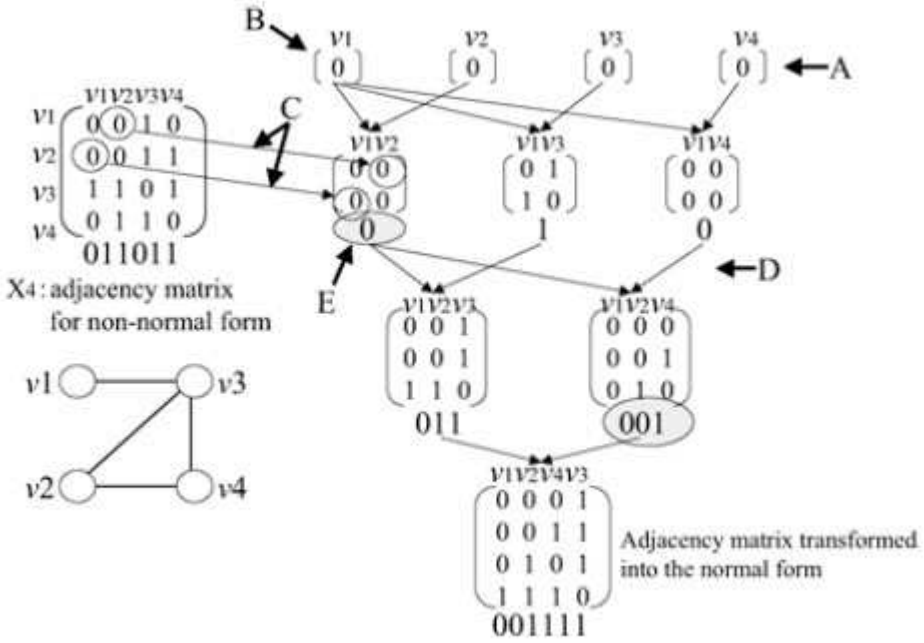
$$X_4 = \begin{array}{c} \\ N_1 \\ N_2 \\ N_3 \\ N_3 \end{array} \begin{array}{c} \begin{array}{cccc} N_1 & N_2 & N_3 & N_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{array}, \quad Y_4 = \begin{array}{c} \\ N_1 \\ N_2 \\ N_3 \\ N_3 \end{array} \begin{array}{c} \begin{array}{cccc} N_1 & N_2 & N_3 & N_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \end{array}.$$

Spełniony jest warunek 1 dla identycznych węzłów, spełniony jest warunek 2 również jest spełniony to dla warunku 3 sprawdzamy: code(X4) = 01001 < code(Y4) = 01011. Jest to prawda, więc te 2 macierze można połączyć do Z5:

$$Z_5 = \begin{array}{c} \\ N_1 \\ N_2 \\ N_3 \\ N_3 \\ N_3 \end{array} \begin{array}{c} \begin{array}{ccccc} N_1 & N_2 & N_3 & N_3 & N_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & ? \\ 1 & 1 & 0 & ? & 0 \end{pmatrix} \end{array}.$$

W algorytmie Apriori, (k + 1)-element staje się kandydatem częstym tylko wtedy kiedy wszystek k – podzbiory elementów z (k + 1)-zbioru jest częstym wzorcem wg wskazanego wsparcia. Podobnie jest w strukturze grafu, k+1 elementowy graf G(Zk+1) jest rozważany na kandydata częstego wyszukanego, podgrafu gdy macierz sąsiedztwa wszystkich wywołanych k-podgrafów G(Zk+1) jest potwierdzona, aby reprezentować częsty podgraf.. Ponieważ nasz algorytm wygeneruje tylko macierze sąsiedztwa normalnej formy wcześniejsza k macierz sąsiedztwa jest generowana przez usunięcie i-tego węzła, i-tej kolumny i wiersza prowadzi to

do powstanie nie normalnej postaci macierzy sąsiedztwa. Macierz nienormalnej formy jest przekształcana do normalnej wg schematu:



Numery pod macierzami sąsiedztwa odpowiadają kodom do znormalizowania.

2. Kanoniczna forma.

Gdy wyprowadzimy wszystkich kandydatów częstych ich wartość wsparcia jest wyliczana przez skanowanie całej bazy danych. Jednakże różne normalne macierze sąsiedztwa mogą reprezentować inne grafy np.:

$$X_5 = \begin{array}{c} \\ N_1 \\ N_1 \\ N_1 \\ N_1 \\ N_1 \end{array} \begin{array}{c} \begin{array}{ccccc} N_1 & N_1 & N_1 & N_1 & N_1 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}, \quad Y_5 = \begin{array}{c} \\ N_1 \\ N_1 \\ N_1 \\ N_1 \\ N_1 \end{array} \begin{array}{c} \begin{array}{ccccc} N_1 & N_1 & N_1 & N_1 & N_1 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}.$$

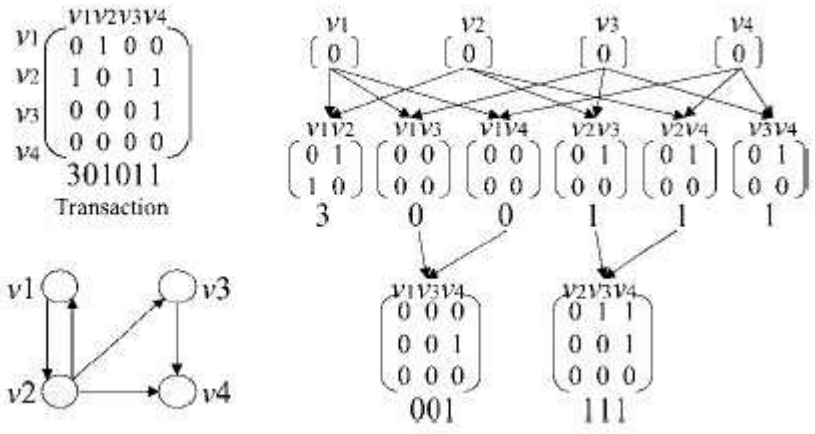
Aby poprawnie policzyć wsparcie musimy sprowadzić macierz do postaci kanonicznej.
K jest wielkością grafu G i T(G)-{Xk | Xk jest normalną formą macierzy G przystającą do G(Xk)}. Wtedy kanoniczna postać CK grafu G ma postać:

$$C_k \quad w.r.t \quad code(C_k) = \min_{X_k \in \Gamma(G)} code(X_k)$$

Kanoniczna forma macierzy Ck jest otrzymywana przez permutacje wierszy i kolumn normalnej formy macierzy Xk. Permutacje są zrobione przez operację $C_k = W_k^T X_k W_k$, gdzie każdy element Wk jest równy 1 kiedy każdy i-ty węzeł Xk powinien zostać permutowany z j-tym węzłem w przeciwnym razie ma wartość 0. Rozważmy wyprowadzenie Wk by przekształcić normalną postać do formy kanonicznej macierz Ck. Jest ona rozpatrywana dla każdego częstego k-1 podgrafu. Usuwamy ostatni rząd i kolumnę Ck, Ck-1 jest również kanoniczną formą.

3. Częstość

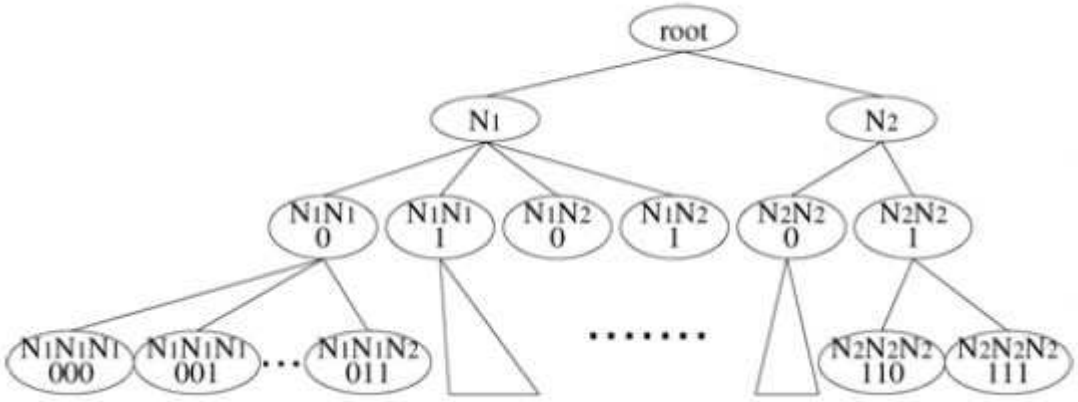
Zobrazowanie wyliczenia częstości na rysunku:



Dla grafu skierowanego nie etykietowanego o rozmiarze 3 zakładamy że mamy kody macierzy granicznej, dalej zakładamy ze wszystkie grafy wielkości 2 z wyjątkiem tego który ma kod 3 są znane. Zaczynając od macierzy sąsiedztwa wielkości 1 , kod macierzy 301011

5. Implementacja i testy wydajnościowe algorytmu

Struktura drzewa jest często stosowaną strukturą danych do przeszukiwań. Przeszukiwania zaczyna się od korzenia i schodzi się niżej. Poniżej struktura drzewa niekierowanego o dwóch etykietach N1 I N2. Punkt rozgałęzienia odpowiada normalnej formie macierzy.

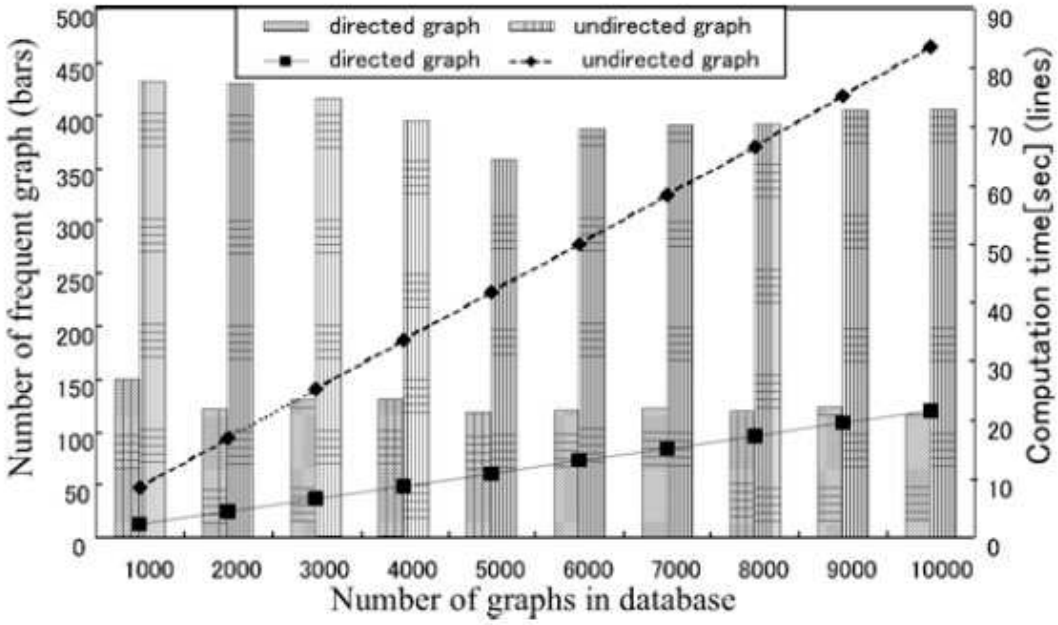


Oszacowanie wydajności:

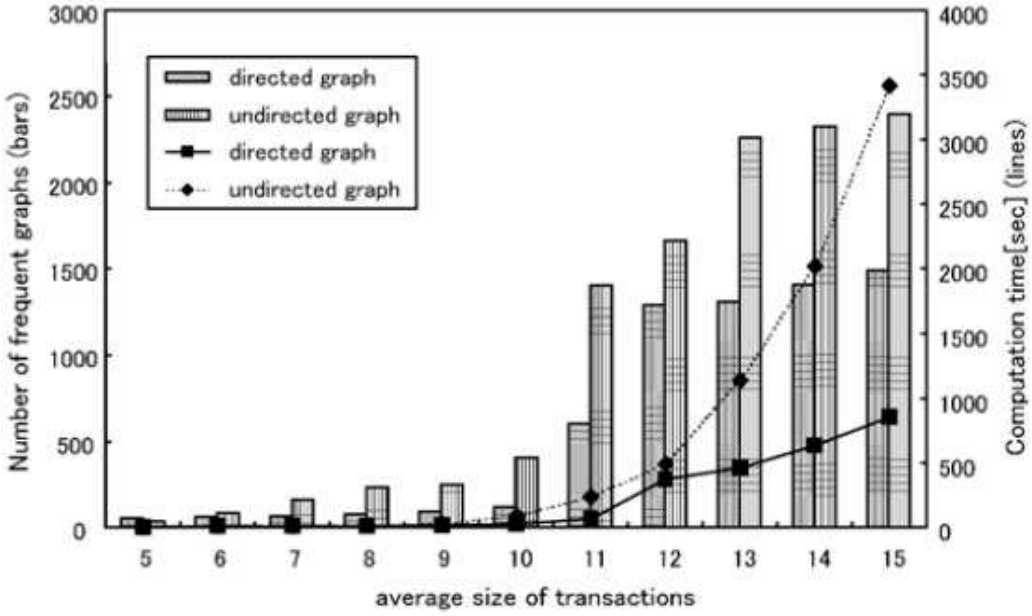
Definicja parametrów:

Parametr	Definicja	Przykładowa wartość
rM	Liczba transakcji	10 000
|T|	Przeciętna wielkość transakcji	10
L	Liczba częstych wzorców	10
|I|	Przeciętna wielkość częstego wzorca	4
rM	Liczba etykiet węzła	5
RM	Prawdopodobieństwo istnienia związku	50%
minsup	Minimalne wsparcie	10%

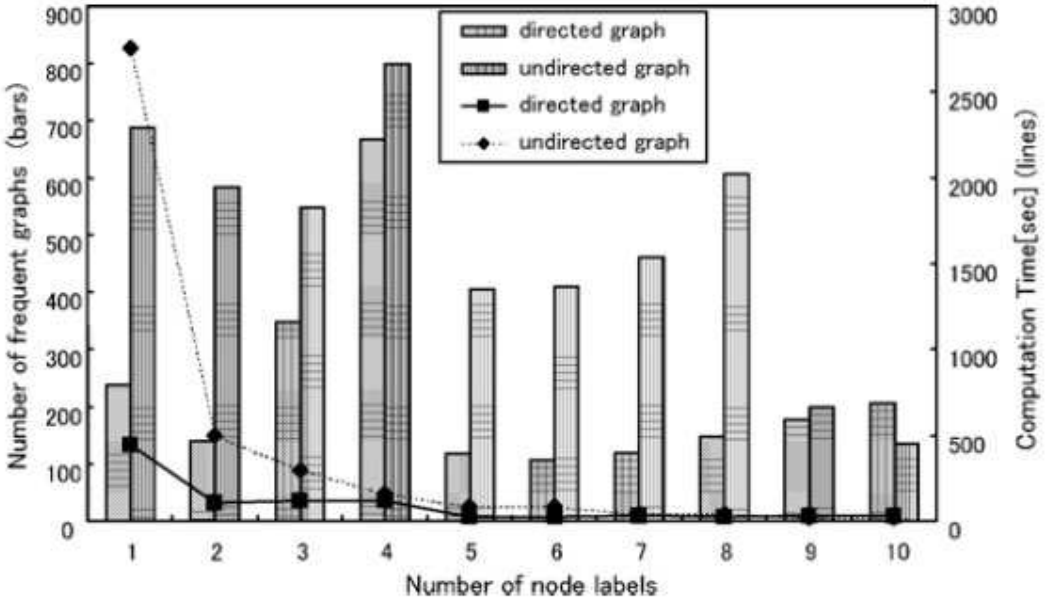
Poniższe wykresy przedstawiają rezultaty jak zmienia się czas kiedy zmieniamy ilość transakcji (wykres1), średnią wielkość transakcji (wykres 2), liczbę etykiet węzła (wykres3), minimalne wsparcie (wykres4) oraz istnienie związków (wykres 5)



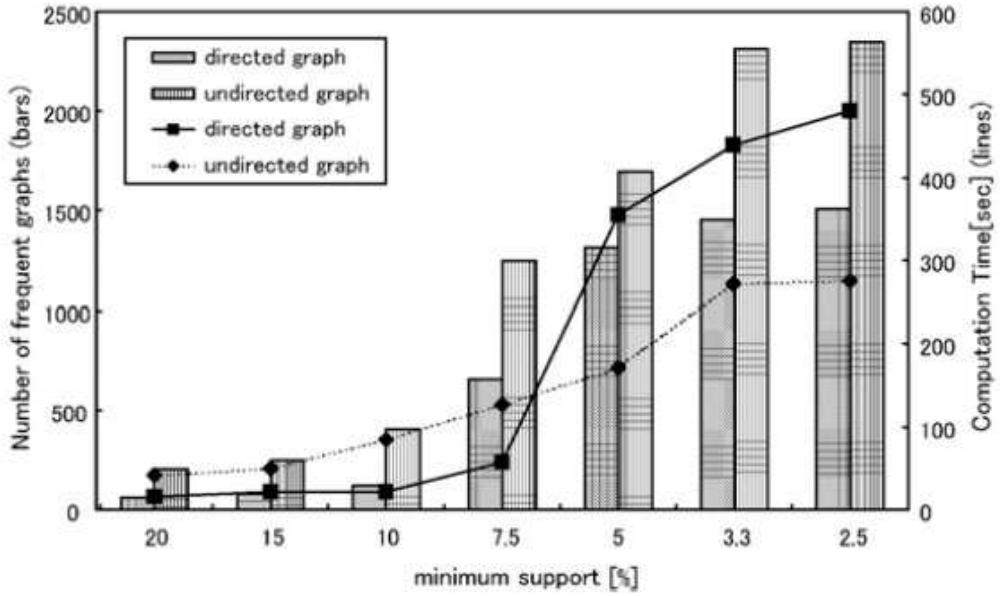
Liczba różnych częstych wywołań podgrafu jest prawie stała i czas obliczeniowy jest proporcjonalny do liczba transakcji.



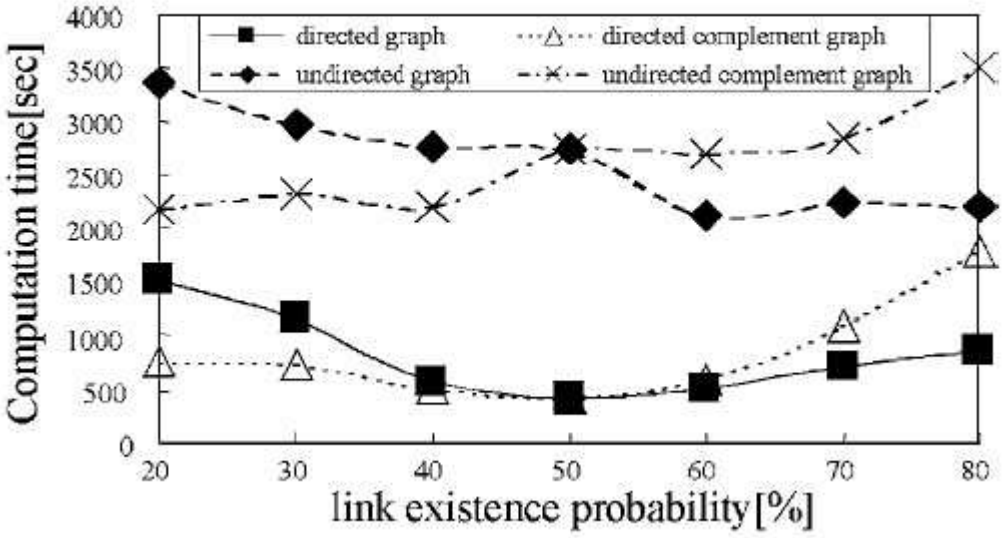
Czas obliczenia i ilość różnych częstych wywołał wzrost podgrafu wykładniczo z wielkością transakcji.



Raz obliczenia szybko spadają jako liczba węzłów etykietuje wzrosty.



Wykres wskazuje, że czas obliczeniowy i liczba częstego wzorca wywołanego podgraf wzrosła jako minimalne poparcie jest zmniejszony.

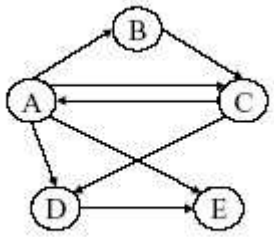


Każdy raz obliczenia dla p został oceniony, ponieważ komplet danych wygenerował pod p. spadek czasu obliczenia jest przestrzeżony jako wzrosty prawdopodobieństwa.

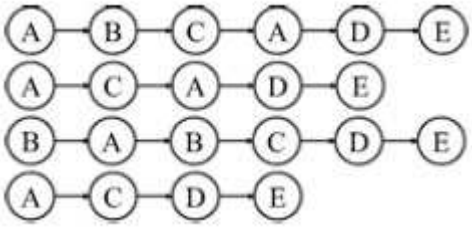
6. Zastosowania algorytmu

Analiza sieci Web (web browning analysis)

Opisany algorytm świetnie znajduje częste ścieżki poruszania się użytkowników po serwisach WWW na podstawie wiadomości zapisanych w logach serwera danego serwisu. Do analizy ruchu na serwisie wystarczy plik logów, który można przedstawić w formie grafu i znaleźć wzorce częste.



Powyższy rysunek będzie przykładem struktury strony internetowej, która zawiera 5 adresów URL. Poniżej wyciągnięte z logu ścieżki odwiedzania tej strony przez hiperłącza:



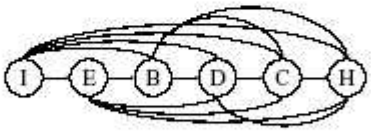
Dla przykładu przeanalizujmy algorytm dla analizy strony „Achaja NAVI” firmy Recruit Co. Firma chciała poznać jak użytkownicy poruszają się po serwisie, gdzie umieszczać reklamy, które strony są najbardziej odwiedzane.

W tym celu poddano analizie logi z serwera, które posiadały 25000 wpisów odwiedzonych adresów url, które były analizowane przez algorytm. Wielkość pliku logów wynosiła ok. 400MB, w którym znajdowało się około 8700-adresów i powiązań średni czas dostępu do strony użytkowników wynosił 5minut. Zmiana pliku testowego z logami do struktury 3.1(macierz) doprowadziła do powiększenia pliku do 800MB. Liczba wszystkich transakcji zapisanych w logach wyniosła 50666, a przeciętna wielkość jednej transakcji to 118. Po wrzuceniu transakcji do grafu okazało się, że liczba etykiet w grafie wynosi 8566 – czyli wszystkich unikalnych stron odwiedzonych przez użytkowników. Aby skrócić obliczenia każda etykieta została zastąpiona przez krótki symbol indeksu, co spowodowało, że plik zmniejszył się do rozmiaru 79MB.

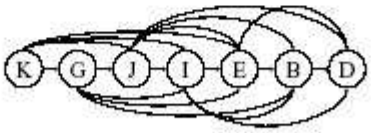
Po wykonaniu opisanego algorytmu minsup współczynnikiem minsup=15% liczba odkrytych częstych wzorców wyniosła 1898 a rozmiar maksymalny transakcji to 7. Czas wykonania algorytmu to 52.3 sekundy.

3 przykłady wyciągniętych wzorców:

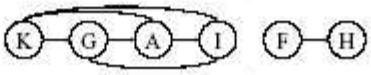
A	/NAVI/mscategory/Sub/s01.html
B	/NAVI/mscategory/Sub/s03.html
C	/NAVI/mscategory/Sub/s04.html
D	/NAVI/mscategory/Sub/s05.html
E	/NAVI/mscategory/Sub/s06.html
F	/NAVI/mscategory/Sub/s08.html
G	/NAVI/mscategory/Sub/s09.html
H	/NAVI/mscategory/Sub/s12.html
I	/NAVI/mscategory/Sub/s13.html
J	/NAVI/mscategory/Sub/s14.html
K	/NAVI/mscategory/Sub/s16.html



(1) Extracted pattern (Frequency 76).



(2) Extracted pattern (Frequency 80).



(3) Extracted pattern (Frequency 77).

7. Podsumowanie

Powyższy algorytm świetnie sobie radzi ze strukturą grafu skierowanego jak i niekierowanego. Graf może mieć wiele etykietowanych węzłów oraz może posiadać pętle. Może być stosowany w praktyce do wyszukiwania częstych zachowań użytkowników WWW jak również do analizy substancji chemicznych – badanie toksyczności substancji chemicznej- znajdywanie struktury typowej dla czynnika rakotwórczego.

Zastosowanie eksploracji danych opisujących korzystanie z zasobów sieci Web daje nam możliwość odkrywania ogólnych wzorców zachowań użytkowników. Odkryta wiedza pozwala na: budowie adaptatywnych serwerów WWW (personalizacja usług serwerów WWW), optymalizację struktury serwera i poprawę nawigacji oraz znajdowanie potencjalnie najlepszych miejsc reklamowych znając wzorce zachowań i ich preferencje konsumenckie.

Literatura:

[1] Hand, David J. „Eksploracja danych" cop. 2005. Wydawnictwa Naukowo Techniczne, Wrzesień 2005

[2] Akihiro Inokuchi, Takashi Washio and Hiroshi Motoda, "Complete Mining of Frequent Patterns from Graphs: Mining Graph Data, ,Institute for Scientific and Industrial Research, Osaka University, Osaka, 567-0047, Japan"

[3] Robert Sedgewick „Algorytmy w C++ grafy" wydanie I Warszawa 2003 wydawnictwo RM

[4] Http://pl.wikipedia.org/wiki/Log_(informatyka)

[5] Włodzimierz Gajda http://gajdaw.pl/