

Politechnika Poznańska
Wydział Informatyki i Zarządzania
Instytut Informatyki

Praca dyplomowa magisterska

**RÓWNOLEGŁE ODKRYWANIE REGUŁ ASOCJACYJNYH
ZAIMPLEMENTOWANE NA PROCESORY GRAFICZNE**

inż. Tomasz Kujawa

Promotor
dr inż. Witold Andrzejwski

Poznań, 2011

Tutaj przychodzi karta pracy dyplomowej;
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

Spis treści

1	Wstęp	1
1.1	Cel i zakres pracy	2
2	Podstawy teoretyczne	4
2.1	Definicje	4
2.1.1	Model teoretyczny	4
2.2	Aktualna wiedza	6
2.2.1	Algorytm Apriori	6
	Generowanie zbiorów częstych	6
	Generowanie reguł asocjacyjnych	8
2.2.2	Algorytm FP-growth	8
	Kompresja bazy danych	9
	Konstrukcja FP-drzewa	10
	Eksploracja FP-drzewa	11
3	Uniwersalna architektura procesorów wielordzeniowych	12
3.1	Czas przetwarzania równoległego	12
3.1.1	Procesor	13
3.2	Wpływ procesorów graficznych na procesy obliczeniowe	13
3.2.1	Krótką historia kart graficznych	14
3.2.2	Wczesne obliczenia na kartach graficznych	14
3.3	CUDA	14
3.3.1	Czym jest architektura CUDA?	14
3.3.2	Używanie CUDA	14
	Literatura	15

Rozdział 1

Wstęp

Proces informatyzacji przedsiębiorstw, rozpoczęty kilka dekad temu, wprowadził światową gospodarkę na nowe, dotąd nieznane tory rozwoju. Skrócenie procesu produkcyjnego, wprowadzenie kontroli komputerowych, czy też skomputeryzowanych maszyn skróciło i ułatwiło produkcję, a także zarządzanie procesami w firmach i przedsiębiorstwach. Przed ludźmi stanęły możliwości, ale także wyzwania, z którymi nigdy wcześniej nikt nie musiał sobie radzić. Zmiany, jakie nastąpiły przez ostatnie trzy dekady są nieodwracalne i zmuszają informatyków do tworzenia nowych aplikacji, które będą w stanie sprostać wymaganiom im stawianym.

Informatyzacja firm, instytucji oraz innych jednostek organizacyjnych powinna realizować dwa podstawowe cele. Z jednej strony powinna ona usprawniać pracę pojedynczego pracownika poprzez automatyzację realizowanych przez niego rutynowych zadań. Dzięki wykorzystaniu możliwości komputerów działania te powinny być wykonywane szybciej i w sposób bardziej niezawodny. Z drugiej strony celem informatyzacji jest wpływanie na działanie całych firm w wyniku wspomagania decyzji kadry zarządzającej przedsiębiorstwami. Szybka analiza bazująca na pełnej i aktualnej informacji o stanie firmy może ułatwić kadrze zarządzającej podejmowanie trafnych i szybkich decyzji o strategicznym znaczeniu dla rozwoju danego przedsiębiorstwa.

Wprowadzenie komputerów do właściwie każdej przestrzeni ludzkiego życia wpłynęło na wyprodukowanie olbrzymich ilości danych. Reprezentowane są one w sposób umożliwiający ich składowanie i przetwarzanie komputerowe przez aplikacje analityczne. W chwili obecnej ludzkość jest świadkiem eksplozji ilości danych produkowanych przez różnego rodzaju systemy komputerowe. Analiza tych danych przynieść może wymierne korzyści nie tylko w kwestiach finansowych, ale również poznawczych. Dzięki analizie danych zebranych w przeszłości możliwe jest lepsze dopasowanie planów w przyszłości - na tej podstawie planowane mogą być np. akcje marketingowe, czy też promocje w supermarketach spożywczych. Wykorzystanie wiedzy uzyskanej w ten sposób jest niezwykle szerokie i może być użyte w każdym rejonie działalności firmy.

Odkrycie zależności pomiędzy zgromadzonymi danymi bez zastosowania narzędzi informatycznych jest procesem bardzo skomplikowanym i wymagającym do realizacji dużo czasu. Przy obecnej złożoności większości systemów oraz rozmiarom danych produkowanych przez te systemy, koszt czasowy jest na tyle duży, że ręczna analiza tych danych stała się niemożliwa. Dlatego też tworzone są narzędzia umożliwiające odkrywanie prawidłowości w dużych zbiorach danych, by człowiek na tej podstawie mógł podejmować decyzje i wyciągać wnioski.

Dział informatyki, który zajmuje się odkrywaniem ukrytych dla człowieka prawidłowości i reguł w danych nazywa się eksploracją danych (ang. *Data Mining*, w literaturze spotkać można również określenie drążenie danych, ekstrakcja danych, pozyskiwanie wiedzy, czy też wydobywanie danych [EN05]), który jest jednym z etapów procesu *odkrywania wiedzy z baz danych* (*KDD*,

ang. *Knowledge Discovery in Databases*). Proces odkrywania wiedzy w bazach danych obejmują zwykle działania bardziej złożone niż tylko eksploracja danych takich, jak selekcja danych, transformacja lub kodowanie danych, czy też raportowanie i prezentowanie odkrytych informacji [EN05]. Eksploracja danych to proces odkrywania wiedzy w postaci nowych, użytecznych, poprawnych i zrozumiałych wzorców w bardzo dużych wolumenach danych [FPSSU96]. Możliwości stosowania technik eksploracji danych w praktyce, wymagają efektywnych metod przeszukiwania ogromnych plików lub baz danych. Warto przy tym wspomnieć, że tego typu technologie nie są w chwili obecnej dobrze zintegrowane z systemami zarządzania bazami danych.

Eksploracja danych odbywa się najczęściej w środowisku baz lub hurtowni danych, które stanowią doskonałe źródła danych do analizy - głównie ze względu na łatwość dostępu oraz usystematyzowaną strukturę przechowywanych informacji. Ponieważ liczba odkrytych wzorców w wielu przypadkach może być bardzo duża, odkryte wzorce bardzo często zapisuje się w osobnych relacjach bazy lub hurtowni danych. Pozwala to na ich dalsze przetwarzanie w trybie off-line przez użytkowników końcowych. Pojęcie eksploracji zyskuje coraz większą popularność (również w wymiarze marketingowym) i jest wykorzystywane w wielu dziedzinach ludzkiego życia.

Jednym z najczęściej wykorzystywanych modeli wiedzy w eksploracji danych są reguły asocjacyjne. Reguła asocjacyjna ma postać $X \Rightarrow Y$, gdzie X oraz Y są wzajemnie rozłącznymi zbiorami elementów. Przykładem reguły, która mogła zostać odkryta w bazie danych sklepu komputerowego, może być reguła postaci $komputer \wedge myszka \Rightarrow monitor$. Prezentuje ona fakt, że klienci kupujący komputer oraz myszkę z dużym prawdopodobieństwem kupią również monitor. W [AIS93] po raz pierwszy sformułowany został problem odkrywania reguł asocjacyjnych wraz z algorytmem Apriori, który jest podstawą wielu algorytmów znajdujących reguły asocjacyjne. Algorytm ten został następnie rozszerzony w pracy [Agr94].

W ostatnich latach pojawiły się nowe możliwości wykorzystania współczesnych komputerów. W roku 2007 firma NVIDIA udostępniła programistom uniwersalną architekturę obliczeniową CUDA (ang. *Compute Unified Device Architecture*), który umożliwia wykorzystanie mocy obliczeniowej procesorów graficznych (*GPU*, ang. *Graphics Processing Unit*), bądź innych procesorów wielordzeniowych, do rozwiązywania ogólnych problemów obliczeniowych w sposób znacząco wydajniejszy niż w przypadku tradycyjnych, sekwencyjnych procesorów [Cor07a]. Choć w grach komputerowych moc obliczeniową jednostek graficznych można wykorzystać do obliczeń fizyki, to CUDA idzie jeszcze dalej, umożliwiając przyspieszenie obliczeń w takich dziedzinach, jak biologia, fizyka, kryptografia, bioinformatyka oraz inne. Dla potrzeb tego segmentu NVidia opracowała kartę graficzną o nazwie *Tesla* [Cor07b]. Układy te są pierwszymi układami produkowanymi na masową skalę, które przeznaczone zostały do pracy *obliczeniach ogólnego przeznaczenia na układach GPU* (*GPGPU*, ang. *General-Purpose Computing on Graphics Processing Units*), czyli segmencie do tej pory zarezerwowanym dla klasycznych procesorów obliczeniowych.

Do tej pory bardzo małe jest zainteresowanie wykorzystaniem tej technologii w procesie odkrywania wiedzy, a w szczególności znajdowania reguł asocjacyjnych. Wyniki przeprowadzonych eksperymentów pozwalają przypuszczać, że algorytm wykorzystujący możliwości procesorów wielordzeniowych będzie wyraźnie szybszy od klasycznych algorytmów eksploracji danych zaimplementowany na tradycyjnych procesorach.

1.1 Cel i zakres pracy

Celem pracy jest zaprojektowanie i zaimplementowanie algorytmu odkrywającego reguły asocjacyjne, który będzie wykorzystywał możliwości współczesnych kart graficznych dzięki wykorzystaniu technologii CUDA oraz porównanie zaprojektowanego i zaimplementowanego algorytmu do

innych, podstawowych algorytmów odkrywania reguł asocjacyjnych. W ramach pracy dokonane zostanie również zebranie wiedzy dotyczącej algorytmów eksploracji reguł asocjacyjnych.

Rozdział 1 - wstęp.. Tutaj dalszy opis struktury pracy - zrobiony na koniec, gdy wszystko dalej będzie już znane.

Rozdział 2

Podstawy teoretyczne

W rozdziale tym przedstawiony zostanie przegląd literatury, który stanowi podstawy wiedzy na temat eksploracji danych, a w szczególności problemu odkrywania reguł asocjacyjnych w dużych zbiorach danych. Zebrana wiedza posłużyła autorowi do opracowania algorytmu wykorzystującego możliwości współczesnych kart graficznych.

2.1 Definicje

2.1.1 Model teoretyczny

Niech $I = \{i_1, i_2, \dots, i_m\}$ będzie zbiorem elementów o liczności $|I| = m$. *Transakcją* nazwano dowolny, niepusty podzbiór $X \subseteq I$ zbioru elementów. Bazą danych DB nazwano dowolny zbiór par (id, X) , gdzie X jest transakcją, a id jest dowolną wartością unikalną w ramach bazy danych nazywaną *identyfikatorem transakcji*. Bez utraty ogólności założono iż $id \in \mathbb{N}$.

W tabeli 2.1 zaprezentowany został przykładowy zbiór elementów oraz zestaw transakcji. Na podstawie tych danych obliczane będą wartości wprowadzanych kolejno definicji.

Wsparciem (ang. *support*) $sup(X)$ transakcji X , w bazie danych nazwano częstość wystąpień transakcji w bazie danych. Formalnie przedstawia to wzór 2.1.

$$sup(X) = \frac{|\{id : (id, Y) \in DB \wedge X \subseteq Y\}|}{|DB|} \quad (2.1)$$

Łatwo zauważyć, że jeśli poziom ten jest niski, to oznacza to, że elementy zbioru X w transakcjach rzadko występują razem.

Dla przykładowego zbioru $X = \{\text{milk}\}$ wartość $sup(X)$ została obliczona w przykładzie 2.2, ponieważ zbiór X jest podzbiorem dwóch transakcji - 1 = { bread, milk, butter, beer } oraz 4 = {

I	{ beer, bread, butter, diapers, jam, juice, milk, water }
Transakcje	1: { bread, milk, butter, beer } 2: { bread, butter, water, jam, beer } 3: { beer, diapers, bread, butter, jam } 4: { butter, milk, juice } 5: { diapers, beer, juice, water }

TABLICA 2.1: Przykładowe dane

butter, milk, juice }.

$$\begin{aligned} \text{sup}(X) = \text{sup}(\{\text{milk}\}) &= \frac{|\{1, 4\}|}{|DB|} = \\ &= \frac{2}{5} \end{aligned} \quad (2.2)$$

Definicja 1. Niech będą dane dwie transakcje X i Y takie, że $X \cap Y = \emptyset$. Implikację postaci $X \Rightarrow Y$ nazwano *regułą asocjacyjną*.

Poziom ufności (ang. *confidence*) jest miarą określającą jakość reguły asocjacyjnej [EN05].

Definicja 2. Poziom ufności (*conf*) reguły asocjacyjnej $X \Rightarrow Y$ jest równy

$$\text{conf}(X \Rightarrow Y) = \frac{\text{sup}(X \cup Y)}{\text{sup}(X)} \quad (2.3)$$

Z definicji 2 wynika, że poziom ufności może być interpretowany, jako estymacja prawdopodobieństwa w transakcji zbioru Y pod warunkiem wystąpienia w niej również zbioru X - co oznaczone jest poprzez $P(Y|X)$.

Przykładową regułą asocjacyjną znaną dla danych przedstawionych w tabeli 2.1 może być reguła $\{\text{butter}\} \Rightarrow \{\text{bread, beer}\}$, która zostałaby znaleziona dla $\text{minsup} = 0,6$. Taka reguła posiada współczynnik pewności równy obliczony w przykładzie 2.4.

$$\begin{aligned} \text{conf}(X \Rightarrow Y) &= \text{conf}(\{\text{butter}\} \Rightarrow \{\text{bread, beer}\}) = \\ &= \frac{\text{sup}(\{\text{butter}\} \cup \{\text{bread, beer}\})}{\text{sup}(\{\text{butter}\})} = \\ &= \frac{\text{sup}(\{\text{butter, bread, beer}\})}{\text{sup}(\{\text{butter}\})} = \\ &= \frac{3}{4} \end{aligned} \quad (2.4)$$

Reguły asocjacyjne zazwyczaj powinny spełniać pewne wymagania zdefiniowane przez użytkownika - minimalne wsparcie oraz minimalny poziom ufności, oznaczane odpowiednio *minsup* oraz *minconf*. Wyznaczają one dla aplikacji progi, jakie powinny spełniać zbiory oraz reguły, aby były brane pod uwagę w trakcie analizy.

Definicja 3. Zbiorem częstym $X \subseteq I$ nazywamy taki zbiór, który spełnia zależność $\text{sup}(X) \geq \text{minsup}$.

Generowanie reguł asocjacyjnych zazwyczaj sprowadza się do dwóch, niezależnych kroków:

1. Minimalne wsparcie jest używane do odnalezienia wszystkich zbiorów częstych w bazie danych DB .
2. Znalezione zbiory często oraz minimalny poziom ufności są używane do wygenerowania reguł asocjacyjnych.

W naiwnym podejściu znalezienie wszystkich zbiorów częstych w bazie danych DB jest zadaniem wymagającym przeszukania wszystkich możliwych kombinacji bez powtórzeń ze zbioru I . Zbiór X nazywamy k -zbiorem, jeśli $|X| = k$, tzn. zbiór X ma k elementów. Zbiór możliwych zbiorów elementów ma licznosc równą $2^n - 1$ (wszystkie zbiory, poza zbiorem pustym, który nie jest w tym wypadku zbiorem sensownym w znaczeniu poddania go analizie), czyli zbiór ten jest $(2^n - 1)$ -zbiorem.

Warto zauważyć, że dla każdego zbioru częstego Y , każdy jego podzbiór X jest również zbiorem częstym [AIS93]. Korzystając z tej właściwości wsparcia możliwe jest w sposób efektywny znalezienie wszystkich zbiorów częstych w zadanej bazie danych - z tej zależności korzysta algorytm apriori opisany w rozdziale 2.2.1. Dodatkowo, wszystkie reguły zbudowane na podstawie zbioru częstego Y muszą spełniać warunek minimalnego wsparcia, ponieważ spełnia ten warunek zbiór Y , a suma zbiorów reguły jest zbiorem wyjściowym Y .

2.2 Aktualna wiedza

W rozdziale tym zebrana została oraz opracowana dotychczasowa wiedza (ang. *state-of-the-art*) na temat algorytmów odkrywania reguł asocjacyjnych. Przedstawione zostaną dwa podstawowe algorytmy wykorzystywane w tym procesie: Apriori oraz FP-growth. W chwili obecnej te dwa algorytmy stanowią podstawę, na której budowane są nowe algorytmy, wykorzystujące możliwości współczesnych algorytmów (*Czy wymienić tutaj przykłady algorytmów, które bazują na nich + refy do kilku artykułów?*).

2.2.1 Algorytm Apriori

Pierwszy algorytm odkrywający reguły asocjacyjne został przedstawiony w roku 1994 w pracy [AS94], który jest rozszerzeniem algorytmu zaprezentowanego przez autorów w [AIS93]. Niżej przedstawione zostaną szczegóły działania tego algorytmu nazwanego algorytmem *Apriori*.

Zagadnienie odkrywania reguł asocjacyjnych można podzielić na dwa etapy [AIS93]:

1. Odkrywanie zbiorów częstych, których wartość wsparcia jest wyższa od wartości *minsup*.
2. Generowanie reguł asocjacyjnych na podstawie znalezionych zbiorów częstych. Reguła $X \Rightarrow Y$ jest wynikiem działania algorytmu dla zbioru $Z = X \cup Y$, jeżeli spełnia ona nierówność $conf(X \Rightarrow Y) \geq minconf$. Ponieważ zbiór $Z = X \cup Y$ jest zbiorem częstym, to reguła spełnia również warunek przekraczania minimalnego wsparcia.

Na tym etapie możliwe jest tworzenie reguł, w których w zbiorze *poprzedników* (X z oznaczeń z definicji 1) jest wiele elementów oraz jeden w *następniku* (zbiór Y z definicji 1) [AIS93] lub dopuszczana jest możliwość wielu elementów również w następniku [AS94]. W niniejszej pracy analizowany jest sposób generowania reguł, w którym oba zbiory mogą być zbiorami wieloelementowymi.

W kolejnych podrozdziałach przedstawione zostaną etapy tworzące razem algorytmy Apriori.

Generowanie zbiorów częstych

W celu wyznaczenia zbiorów częstych algorytm dokonuje analizy bazy danych DB , by w kolejnych iteracjach generować rodziny coraz to liczniejszych zbiorów, będących zbiorami częstymi dla zadanej wartości *minsup*. Algorytm zaczyna od znalezienia wszystkich zbiorów jednoelementowych, które są zbiorami częstymi. W każdym kolejnym kroku generowane są zbiory częste na podstawie zbiorów wygenerowanych w kroku poprzednim. Proces ten jest kontynuowany do momentu aż nie zostaną znalezione żadne zbiory częste.

Algorytm generuje zbiory kandydatów jedynie na podstawie zbiorów częstych odkrytych w kroku poprzednim - co ważne generowanie ich odbywa się bez wielokrotnego przeglądania bazy danych transakcji. Intuicja podpowiada, że każdy podzbiór zbioru częstego jest zbiorem częstym.

L_k	Zbiór zawierający k -zbiory. Każdy zbiór zawarty w L_k zawiera dwa pola: i) zbiór oraz ii) wartość <i>support</i> .
C_k	Zbiór k -zbiorów kandydatów (potencjalnych zbiorów częstych). Każdy zbiór zawarty w C_k zawiera dwa pola: i) zbiór oraz ii) wartość <i>support</i> .

TABLICA 2.2: Oznaczenie w opisach algorytmów

Zatem, każdy zbiór częsty zawierający k elementów może być wygenerowany na podstawie połączenia dwóch zbiorów posiadających $k - 1$ elementów, a na koniec kasując te zbiory, których jakikolwiek podzbiór nie jest częsty [AS94].

Tabela 2.2 zawiera spis oznaczeń używanych w opisie algorytmu.

Procedura APRIORI FREQUENT SET GENERATION przedstawia pseudokod realizujący opisany w tym rozdziale algorytm generowania zbiorów częstych.

APRIORI FREQUENT SET GENERATION

```

1   $L_1 \leftarrow \{1\text{-zbiory częste}\}$ 
2  for ( $k = 2; L_{k-1} \neq \emptyset; k++$ )
3      do  $C_k \leftarrow \text{aprioriGen}(L_{k-1})$ 
4          for each transakcja  $t \in DB$ 
5              do  $C_t \leftarrow \text{subset}(C_k, t)$ 
6                  for each kandydat  $c \in C_t$ 
7                      do  $c.\text{count}++$ 
8           $L_k \leftarrow \{c \in C_k | c.\text{count} \geq \text{minsup}\}$ 
9   $\text{Answer} \leftarrow \bigcup_k L_k$ 

```

Procedura aprioriGen Procedura *aprioriGen* reprezentuje proces tworzenia zbiorów k -elementowych kandydatów na podstawie zbiorów wejściowych $(k - 1)$ -elementowych. Procedura ta jest podzielona na dwa etapy: łączenia oraz przycinania.

Jak łatwo zauważyć wynikiem działania JOIN STEP są zbiory k -elementowe, które powstały na podstawie zbiorów wejściowych L_{k-1} , a ich zawartość różni się tylko jednym elementem - ostatnim. Ważnym faktem jest to, iż elementy w zbiorach są uporządkowane leksykograficznie, co wykorzystywane jest w tej procedurze.

JOIN STEP

```

1  insert into  $C_k$ 
2  select  $p.\text{item}_1, p.\text{item}_2, \dots, p.\text{item}_{k-1}, q.\text{item}_{k-1}$ 
3  from  $L_{k-1}$   $p, L_{k-1}$   $q$ 
4  where  $p.\text{item}_1 = q.\text{item}_1, \dots, p.\text{item}_{k-2} = q.\text{item}_{k-2}, p.\text{item}_{k-1} < q.\text{item}_{k-1}$ 

```

Warto zauważyć, że JOIN STEP jest ekwiwalentem rozszerzania zbioru L_{k-1} każdym elementem zbioru elementów I , a następnie kasowania tych $(k - 1)$ -zbiorów otrzymanych przez usuwanie $(k - 1)$ elementu, które nie są w L_{k-1} .

Warunek $p.\text{item}_{k-1} < q.\text{item}_{k-1}$ zapewnia, że nie będą generowane duplikaty. Dlatego też po etapie łączenia zachodzi zależność $C_k \supseteq L_k$.

Następnym krokiem jest PRUNE STEP, w którym usuwane są wszystkie elementy $c \in C_k$, którego jakikolwiek podzbiór $(k - 1)$ -elementowy zbioru c nie należy do L_{k-1} .

PRUNE STEP

```

1  for each zbiór  $c \in C_k$ 
2      do
3          for each  $(k - 1)$ -podzbiór  $s$  zbioru  $c$ 
4              do if  $s \notin L_{k-1}$ 
5                  then delete  $c$  z  $C_k$ 

```

Celem operacji przycinania (ang. *prune*) jest ograniczenie rozmiaru zbioru C_k przed sprawdzeniem wsparcia dla kandydatów w bazie danych DB . W tym celu wykorzystywana jest właściwość, z której wynika, że jeśli jakiś $(k - 1)$ -podzbiór danego kandydata ($c \in C_k$) nie występuje w L_{k-1} , to kandydat c nie jest zbiorem częstym i powinien być usunięty z C_k .

Generowanie reguł asocjacyjnych

Po zakończeniu pierwszego etapu algorytm przystępuje do drugiego, czyli do budowania reguł asocjacyjnych na podstawie odkrytych zbiorów. Podobnie, jak w [AS94] algorytm będący przedmiotem analizy niniejszej pracy, generuje wszystkie możliwe reguły asocjacyjne dla zadanego zbioru. Mniej ogólny sposób generowania reguł został przedstawiony w pracy [AIS93], jednakże podjęto decyzję, że jest to sposób zbyt mało użyteczny w środowisku produkcyjnym.

Aby wygenerować reguły, dla każdego zbioru częstego l znajdowane są niepuste podzbiory - podzbiór taki oznaczony jest jako a . Dla takich oznaczeń wygenerowana zostanie reguła $a \Rightarrow (l - a)$, jeżeli spełniona jest nierówność $\frac{\text{support}(l)}{\text{support}(a)} \geq \text{minconf}$. Warto zauważyć, że dla każdego zbioru częstego generowane są wszystkie możliwe niepuste podzbiory - zapewnia to, że odkryte zostaną wszystkie możliwe reguły.

Procedura GENERATE FREQUENT ITEMSETS prezentuje generowanie reguł asocjacyjnych na podstawie odkrytych k -zbiorów częstych l_k będących elementami zbioru L_k ($l_k \in L_k$).

GENERATE FREQUENT ITEMSETS

```

1  for each zbiór częsty  $l_k, k \geq 2$ 
2      do call genrules( $l_k, l_k$ )

```

W powyższym algorytmie wykorzystana została funkcja GENRULES, która na podstawie dwóch zbiorów generuje reguły asocjacyjne. Zapis pseudokodu tej funkcji przedstawiony jest poniżej.

GENRULES(l_k : k -zbiór częsty, a_m : m -zbiór częsty)

```

1   $A \leftarrow \{(m - 1)\text{-zbiór } a_{m-1} \mid a_{m-1} \subset a_m\}$ 
2  for  $a_{m-1} \in A$ 
3      do  $\text{conf} \leftarrow \frac{\text{support}(l_k)}{\text{support}(a_{m-1})}$ 
4          if  $\text{conf} \geq \text{minconf}$ 
5              then output reguła  $a_{m-1} \Rightarrow (l_k - a_{m-1})$ 
              ufność =  $\text{conf}$  oraz wsparcie =  $\text{support}(l_k)$ 
6          if  $m - 1 > 1$ 
7              then call genrules( $l_k, a_{m-1}$ )
              generowanie reguł podzbiorów zbioru  $a_{m-1}$ 

```

2.2.2 Algorytm FP-growth

Podstawową wadą algorytmu Apriori jest wysoki koszt przetwarzania dużych zbiorów danych. Przykładowo, dla 10^4 1-zbiorów częstych, algorytm Apriori wygeneruje około 10^7 2-zbiorów kandydatów, które następnie poddane zostaną weryfikacji, czy są zbiorami częstymi. Poza tym algorytm

ten wymaga wielokrotnego odczytywania zawartości bazy danych - w każdym kroku algorytmu należy odczytać całą bazę danych w celu obliczenia wsparcia zbiorów kandydujących.

Wymienione wyżej wady algorytmu Apriori nie występują w algorytmie *FP-growth* przedstawionym w pełni w pracy [HPYM04]. Algorytm ten pozwala wyeliminować konieczność generowania tak dużej liczby kandydujących zbiorów elementów oraz ogranicza liczbę dostępu do bazy danych do absolutnego minimum. Co więcej algorytm ten charakteryzuje się kompletnością, co oznacza, że znajdowane są wszystkie wzorce o określonej częstości.

Algorytm *FP-growth* można podzielić na trzy podstawowe kroki.

1. W kroku pierwszym generowana jest skompresowana wersja bazy danych *DB*, mająca postać drzewa częstych wzorców.
2. Drugim krokiem jest transformacja tak skonstruowanego drzewa do postaci *FP-drzewa* (patrz definicja 4).
3. Trzeci krok polega na analizie *FP-drzewa* celem odnalezienia reguł asocjacyjnych. W kroku tym stosowana jest metoda dziel i zwyciężaj (ang. *divide-and-conquer*) zamiast podejścia Apriori, czyli generowania na każdym poziomie zbioru kandydatów na zbiory częste, a następnie odcinaniu kandydatów nie spełniających kryteriów akceptacji. Takie podejście przekształca problem znajdowania długich reguł w problem szukania krótszych, a następnie konkatenaacji wyników.

Definicja 4. *FP-drzewo* (ang. *frequent-pattern tree*) jest to ukorzeniony, etykietowany w wierzchołkach graf acykliczny spełniający poniższe cechy.

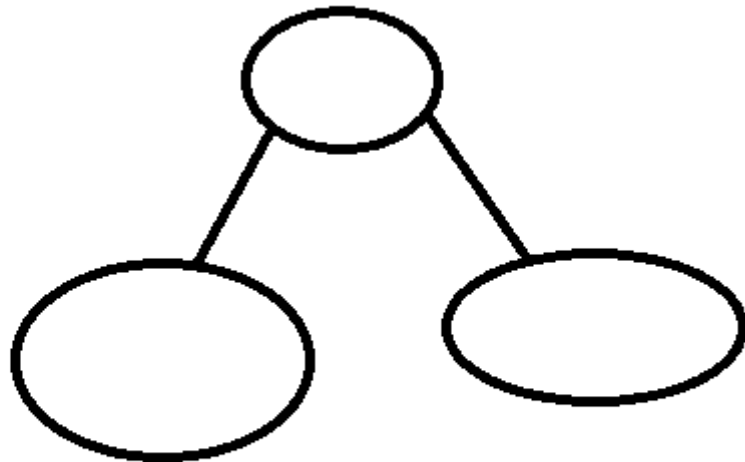
1. Korzeniem drzewa jest jeden element *null*, zbiór poddrzew prefiksowanych elementami (jako dzieci elementu *null*) oraz tablicy nagłówkowej zawierającej wpisy *element* \rightarrow *wskaźnik na element drzewa*.
2. Każdy wierzchołek poddrzewa składa się z trzech elementów: nazwy elementu (ang. *item name*), licznika (ang. *count*) oraz wskaźnika na inny wierzchołek. Nazwa elementu (*itemName*) w sposób jednoznaczny identyfikuje element ze zbioru elementów *I*, licznik przechowuje liczbę transakcji reprezentowanych przez ścieżkę od *null* do tego elementu, natomiast wskaźnik wskazuje na kolejny wierzchołek w *FP-drzewie*, którego nazwa jest identyczna do danego.
3. Każdy wpis w tablicy nagłówkowej (ang. *frequent-item-header table*) składa się z dwóch elementów: nazwy elementu oraz wskaźnika na pierwszy element w drzewie posiadający identyczną nazwę.

Rysunek 2.1 przedstawia przykładowe *FP-drzewo* wraz z tablicą nagłówkową.

Kompresja bazy danych

Pierwszy etap polega na znalezieniu wszystkich 1-zbiorów częstych występujących w bazie danych *DB*. Po ich odnalezieniu (F_1 - zbiór znalezionych 1-zbiorów częstych) z każdej transakcji *T* usuwany jest ten element, który nie należy do F_1 .

W wyniku usunięcia elementów nie tworzących jednoelementowych zbiorów częstych, baza ma zazwyczaj znacznie mniejszy rozmiar niż wyjściowa baza danych. Dodatkowo w tym kroku elementy w każdej transakcji zostają posortowane według malejącej wartości ich wsparcia.



TUTAJ BĘDZIE FP-DRZEWO

RYSUNEK 2.1: Ilustracja przykładowego FP-drzewa

Konstrukcja FP-drzewa

Algorytm 1. *Konstrukcja FP-drzewa.*

Input: Baza danych transakcji DB oraz minimalne wsparcie ($minsup$).

Output: FP-drzewo utworzone na podstawie zawartości DB

Metoda: Poniżej zostanie opisany proces konstrukcji FP-drzewa.

1. Przeskanowanie bazy danych transakcji DB odbywa się jednokrotnie. Utworzony na tej podstawie zostanie zbiór F , zawierający 1-zbiory częste. Posortowany malejąco zbiór F na podstawie wartości $support$ dla każdego elementu tworzy listę $FList$, czyli listę wszystkich elementów tworzących jednoelementowe zbiory częste.
2. Tworzony jest pierwszy element drzewa - korzeniem zostaje (zgodnie z definicją 4) element z etykietą $null$. Dla każdej transakcji w bazie danych DB wykonywane jest, co następuje.

Wybierane są elementy częste z transakcji, a następnie sortowane zgodnie z kolejnością w $FList$. Niech taka posortowana lista elementów częstych ma postać $[p|P]$, gdzie p jest pierwszym elementem, a P jest pozostałą częścią listy. Następnie wywoływana jest funkcja $insertTree([p|P], T)$, gdzie T jest FP-drzewem.

Funkcja insertTree Jeśli drzewo T ma dziecko N takie, że $N.itemName = p.itemName$, zwiększana jest wartość $N.count$ o wartość 1; w przeciwnym wypadku tworzony jest nowy element N z wartością $count = 1$, a wskaźnik rodzica ustawiany jest na T oraz wskaźnik sąsiedztwa elementu ustawiany jest na element z takim samym $itemName$. Jeśli lista P była niepusta, to wywoływana jest funkcja $insertTree(P, T)$ rekurencyjnie, w przeciwnym wypadku kończone jest działanie funkcji.

Eksploracja FP-drzewa

Po utworzeniu FP-drzewa przeprowadzana jest jego analiza w celu znalezienia wszystkich zbiorów częstych. Eksploracja bazuje na obserwacji, że dla każdego 1-zbioru częstego α wszystkie częste nadzbiory tego zbioru są reprezentowane w FP-drzewie przez ścieżki zawierające wierzchołek (bądź wierzchołki) α .

Analiza rozpoczyna się od znalezienia dla każdego 1-zbioru częstego α wszystkich ścieżek w FP-drzewie, których końcowym wierzchołkiem jest wierzchołek odpowiadający zbiorowi α . Pojedyncza ścieżka, na której końcu znajduje się wierzchołek α w dalszej analizie będzie nazywana *ścieżką prefiksową wzorca α* .

Poniżej zaprezentowany zostanie pseudokod algorytmu przeszukiwania FP-drzewa celem odnalezienia reguł asocjacyjnych.

Algorytm 2. *FP-growth: Przeszukiwanie FP-drzewa celem odnalezienia reguł asocjacyjnych.*

Input: Baza danych transakcji DB reprezentowana przez FP-drzewo zwrócone przez algorytm 1 oraz minimalne wsparcie ($minsup$).

Output: Kompletny zbiór reguł asocjacyjnych.

Metoda: Wywołanie $FP\text{-}GROWTH(FP - drzewo, null)$.

$FP\text{-}GROWTH(Tree, \alpha)$

```

1  if  $Tree$  zawiera jedną ścieżkę prefiksową
2      then  $P \leftarrow$  ścieżka prefiksowa drzewa  $Tree$ 
3           $Q \leftarrow$  wieloczęściowa ścieżka z najwyższym elementem zastąpionym przez korzeń  $null$ 
4          for each kombinacja ( $\beta$ ) elementów z  $P$ 
5              do generuj regułę  $\beta \cup \alpha$  z  $support =$  minimalna wartość  $support$  elementów w  $\beta$ 
6                  niech  $freqPatternSet(P)$  będzie zbiorem wygenerowanych do tej pory reguł
7      else  $Q \leftarrow Tree$ 
8          for each element  $a_i \in Q$ 
9              do generuj regułę  $\beta \leftarrow a_i \cup \alpha$  z  $support = a_i.support$ 
10                 stwórz drzewo warunkowe z  $\beta$  oraz FP-drzewo ( $Tree_\beta$ ) dla  $\beta$ 
11                 if  $Tree_\beta \neq \emptyset$ 
12                     then call  $FP\text{-}GROWTH(Tree_\beta, \beta)$ 
13                     niech  $freqPatternSet(Q)$  będzie zbiorem wygenerowanych do tej pory reguł
14 return  $freqPatternSet(P) \cup freqPatternSet(Q) \cup (freqPatternSet(P) \times freqPatternSet(Q))$ 

```

Rozdział 3

Uniwersalna architektura procesorów wielordzeniowych

Rozdział ten stanowi wprowadzenie to zagadnienia obliczeń wielordzeniowych. Przedstawiona została w nim pokrótce historia rozwoju technologii oraz sprzętu komputerowego, który podążał za wymaganiami stawianymi przez użytkowników. Przedstawiona zostanie historia rozwoju procesora - głównej jednostki obliczeniowej w praktycznie każdym urządzeniu elektronicznym. Później opisany zostanie rozwój karty graficznej oraz zmiana jej zastosowania. Ostatnim, lecz nie najmniej ważnym elementem tego rozdziału jest opis uniwersalnej architektury obliczeniowej CUDA (ang. *Compute Unified Device Architecture*) wprowadzonej przez firmę nVidia w roku 2007, która umożliwia wykorzystanie mocy obliczeniowej *procesorów graficznych* (GPU, ang. *Graphics Processing Unit*), bądź innych procesorów wielordzeniowych, do rozwiązywania ogólnych problemów obliczeniowych w sposób znacząco wydajniejszy niż w przypadku tradycyjnych, sekwencyjnych procesorów [Cor07a].

3.1 Czas przetwarzania równoległego

W poprzednich latach dokonał się znaczący postęp w przechodzeniu przemysłu komputerowego na obliczenia wykonywane równolegle. W roku 2010 większość komputerów konsumenckich była dostarczana do odbiorcy z procesorem zawierającym więcej niż jeden rdzeń (ang. *core*). Począwszy od procesorów dwurdzeniowych w laptopach do 8- czy 16-rdzeniowych stacji roboczych - od pewnego czasu obliczenia równoległe nie są już tylko domeną superkomputerów lub mainframe'ów (ang. *main* - główny, *frame* - struktura). Co więcej, urządzenia elektroniczne takie jak telefony komórkowe czy też odtwarzacze muzyki wyposażane są w procesory wielordzeniowe, co zapewnia im funkcjonalność dalece przekraczającą tę dostępną dla ich poprzedników.

Wynika z tego, że coraz to więcej programistów będzie musiało radzić sobie z implementacją oprogramowania przeznaczonego na jednostki równoległe, wykorzystywać nowe technologie, które będą pozwalały na dostarczenie nowatorskich rozwiązań dla co raz bardziej wymagającej rzeszy użytkowników. Wiersze poleceń to przeżytek - od dawna komputerem steruje się za pomocą skomplikowanych interfejsów graficznych. To samo tyczy się telefonów komórkowych - w chwili obecnej telefon to tylko jedna z wielu funkcji, jakich może dostarczyć. Teraz telefony mogą jednocześnie grać muzykę, dostarczać informacji o obecnej lokalizacji przy użyciu modułu nawigacji satelitarnej (GPS, ang. *Global Positioning System*) oraz wyświetlać zdjęcia.

3.1.1 Procesor

Przez około 30 lat jedną z ważniejszych metod udoskonalania centralnych jednostek obliczeniowych, a przez to zwiększaniu komfortu korzystania z urządzenia przez użytkownika, było zwiększanie prędkości z jaką operował zegar procesora. W latach 80 XX wieku procesor przeznaczony na rynek konsumencki operował z prędkością oscylującą w okolicach 1MHz. Około 30 lat później, w czasach współczesnych, większość komputerów osobistych wyposażona jest w procesory o prędkościach od 1 do 4GHz, czyli posiadają one niemalże 1000 razy szybsze procesory w porównaniu ze wczesnymi jednostkami. Pomimo tego, że zwiększenie prędkości nie jest jedyną metodą na zwiększanie wydajności procesora, to mimo wszystko jest to niezawodne źródło większej wydajności.

Jednakże ograniczenia technologiczne wyznaczają pewne granice, w jakich może wzrastać prędkość zegara procesora. Dlatego też poszukuje się innych, równie niezawodnego źródła zwiększenia wydajności. Nie można już dłużej polegać jedynie na zwiększaniu prędkości. Z powodu restrykcji na mocy oraz wydzielanym cieple oraz docieraniu do granicy rozmiaru tranzystora, naukowcy oraz producenci rozpoczęli poszukiwanie nowych źródeł.

Poza światem konsumentów, czyli w świecie tzw. superkomputerów przez dekady osiągnęto niezwykle wielkie przyrosty mocy w bardzo podobny sposób. Moc procesora używanego w tych komputerach rosła tak samo szybko, jak w przypadku przyrostów procesorów desktopowych. Jednakże, poza wielkimi przyrostami mocy obliczeniowej na jednej jednostce, producenci superkomputerów tworzyli komputery, w których solidne przyrosty w wydajności osiągnęto dzięki zwiększaniu liczby używanych procesorów. Nie jest niczym niezwykłym, że pojedynczy superkomputer składa się z dziesiątek lub setek tysięcy procesorów działających równolegle.

W poszukiwaniu dodatkowych możliwości dla komputerów osobistych, poprawa wydajności w przypadku superkomputerów rodzi pytanie: Dlaczego zamiast zwiększać wydajność pojedynczej jednostki, nie umieścić w komputerze osobistym więcej procesorów? W wypadku zwiększania liczby rdzeni rozwój jednostek obliczeniowych nie byłby ograniczony przez te same niedogodności, co w przypadku ciągłego zwiększania prędkości zegara procesora.

W roku 2005 wiodący producenci procesorów zaczęli oferować jednostki z dwoma, zamiast z jednym rdzeniem. W latach następnych kontynuowano tę taktykę, tworząc jednostki trzy-, cztero-, sześćo oraz ośmio-rdzeniowe. Czasami nazywa się ten okres *rewolucją wielordzeniową*, ponieważ zmiana podejścia do zwiększania wydajności jednostek w znaczący sposób wpłynęła na ewolucję konsumenckiego rynku komputerów.

W chwili obecnej praktycznie każdy komputer osobisty jest wyposażony w procesor dwurdzeniowy. Nawet na rynku niskobudżetowych komputerów z bardzo niskim zapotrzebowaniem na moc, dokonała się rewolucja wielordzeniowa - netbooki będą wyposażone w dwa rdzenie [Cor10].

3.2 Wpływ procesorów graficznych na procesy obliczeniowe

Użycie procesora graficznego, jako jednostki obliczeniowej dla problemów nie związanych bezpośrednio z przetwarzaniem grafiki jest podejściem stosunkowo nowym. W rzeczywistości obliczenia na jednostkach graficznych nie są tak nowe, jak mogłoby się wydawać na pierwszy rzut oka.

3.2.1 Krótka historia kart graficznych

3.2.2 Wczesne obliczenia na kartach graficznych

3.3 CUDA

3.3.1 Czym jest architektura CUDA?

3.3.2 Używanie CUDA

Literatura

- [Agr94] Rakesh Agrawal. Quest: a project on database mining. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1994.
- [AIS93] Rakesh Agrawal, Tomasz Imielinski, Arun Swami. Mining association rules between sets of items in large databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, strony 207–216, 1993.
- [AS94] Rakesh Agrawal, Ramakrishnan Srikant. Fast algorithms for mining association rules. *VLDB*, 1994.
- [Cor07a] NVIDIA Corporation. Cuda - zone. [on-line]
http://www.nvidia.pl/object/cuda_home_new_pl.html, 2007.
- [Cor07b] NVIDIA Corporation. Nvidia tesla 20. [on-line]
http://www.nvidia.pl/object/cuda_home_new_pl.html, 2007.
- [Cor10] Intel Corporation. New dual-core intel atom processor-based netbooks hit shelves today. [on-line] http://newsroom.intel.com/community/intel_newsroom/blog/2010/08/23/new-dual-core-intel-atom-processor-based-netbooks-hit-shelves-today, 2010.
- [EN05] Ramez Elmasri, Shamkand B. Navathe. *Wprowadzenie do systemów baz danych*. Addison-Wesley, Reading, MA, USA, 2005.
- [FPSSU96] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, Ramasamy Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.
- [HPYM04] Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 2004.



© 2011 Tomasz Kujawa

Instytut Informatyki, Wydział Informatyki i Zarządzania
Politechnika Poznańska

Skład przy użyciu systemu L^AT_EX.

BibT_EX:

```
@mastersthesis{ key,  
  author = "Tomasz Kujawa",  
  title = "{Równoległe odkrywanie reguł asocjacyjnych zaimplementowane na procesory graficzne}",  
  school = "Poznan University of Technology",  
  address = "Pozna{\n}, Poland",  
  year = "2011",  
}
```