THUẬT TOÁN KHAI PHÁ TẬP MỤC DỮ LIỆU THƯỜNG XUYÊN TRONG CƠ SỞ DỮ LIỆU GIA TĂNG DỰA TRÊN PHÂN LỚP DỮ LIỆU

NGUYỄN HỮU TRỌNG

I. MỞ ĐẦU

Những vấn đề về khai phá luật kết hợp được tổng kết bởi Q. Zhao trong [1] và B. Goethals trong [2]. Từ thuật toán AIS lần đầu tiên được Agrawal. R giới thiệu năm 1993 trong [3], thuật toán Apriori năm 1996 [4] rồi nhiều thuật toán cải tiến và mới được các nhà nghiên cứu công bố: FP-Tree của J. Han năm 2000 [5], DCI của C. Lucchese năm 2005 [6], CHARM của M. J. Zaki năm 2005 [7], LCM của T. Uno năm 2006 [8], BFS của V. Choi năm 2006 [9], Partition-P-tree (PP-tree) của S. Ahmed năm 2006 [10]... Các thuật toán này chủ yếu xử lí trên tập dữ liệu xác định trước. Ta biết rằng, các tập dữ liệu được bổ sung và gia tăng theo thời gian [12], [13], do vậy các tập thường xuyên và các luật kết hợp đã được tính toán không còn giá trị trên tập dữ liệu mới. Ngoài ra, với một kho dữ liệu ổn định, khi cần tìm các tập thường xuyên với độ hỗ trợ khác, công việc phải thực hiện lại từ đầu.

Một thuật toán khai phá luật kết hợp trên cơ sở dữ liệu (CSDL) gia tăng đã được Nguyễn Xuân Huy, Đoàn Văn Ban và Nguyễn Hữu Trọng đề xuất trong [14]. Thuật toán này dựa vào kỹ thuật xây dựng giàn. Khi một giao tác xuất hiện, thuật toán cập nhật thông tin tại các đỉnh của giàn. Với kỹ thuật duyệt giàn ta lưu trữ độ hỗ trợ của mọi tập mục dữ liệu xuất hiện trong các giao tác, và từ đó ta có thể lọc ra các tập thường xuyên theo yêu cầu.

Kỹ thuật phân hoạch dữ liệu (DL) đã được Ashok Savasere công bố năm 1995 trong [10] và được Shakil Ahmed phát triển trong [11]. Với hướng tiếp cận này, một CSDL kích thước lớn có nhiều dòng được chia ngang thành nhiều tập DL ít dòng để có thể đưa hết vào bộ nhớ trong để xử lí. Trong [15] tác giả đã đề xuất một thuật toán phân lớp DL theo chiều dọc, một CSDL giao tác T trên tập mục dữ liệu I có n phần tử được chia thành n phần và được xử lí song song trên n máy. Tuy nhiên, thuật toán này chỉ xử lí trên tập dữ liệu ổn định.

Để phát triển các kết quả đã nghiên cứu, bài báo này chúng tôi đề nghị một lượt đồ với kỹ thuật phân hoạch dữ liệu trên CSDL gia tăng, với ý tưởng cơ bản như sau:

1) Với một CSDL giao tác $T = \{t_1, t_2, \ldots, t_m\}$ trên tập mục dữ liệu $I = \{x_1, x_2, \ldots, x_n\}$, chuyển thành cơ sở dữ liệu giao tác dọc: $T = \{P_1, P_2, \ldots, P_n\}$ trong đó $P_i = \{j_1, j_2, \ldots, j_k\}$ là tập các định danh giao tác chứa x_i và được lưu trữ thành n tập tin trên bộ nhớ ngoài. Việc tính độ hỗ trợ của các tập ứng viên là tính phần giao của các tập định danh giao tác tương ứng với các mục dữ liệu của tập ứng viên.

2) Khi đi tìm tập mục DL thường xuyên theo một ngưỡng S_0 nào đó, ta cần tính độ hỗ trợ của tất cả các tập ứng viên và lưu lại: $SC = \{(X, Sup) \mid X$ là tập ứng viên và $Sup = Supp(X)\}$. Khi dữ liệu chưa tăng thêm, cần tìm các tập mục dữ liệu thường xuyên có độ hỗ trợ $S_1 > S_0$, công việc đơn giản là lọc từ SC ra tập $F_{S1} = \{X \mid (X, Sup) \in SC$ và $Sup \geq S_1\}$.

3) Theo thời gian, số lượng các giao tác tăng dần, thuật toán tính toán lại độ hỗ trợ của các tập ứng viên trong SC chỉ dựa vào dữ liệu tăng thêm, không cần tính toán lại từ đầu.

Bài viết có 5 phần. Sau phần mở đầu, chúng tôi trình bày các khái niệm cơ bản của bài toá khai phá dữ liệu ở phần 2. Phần 3 là phần chính, chúng tôi trình bày chi tiết thuật toán và thử nghiện Phần 4 so sánh thuật toán với thuật toán Aprior. Phần 5 kết luận và hướng phát triển.

II. CÁC KHÁI NIỆM CƠ BẢN

1. Cơ sở dữ liệu giao tác

Cho $I = \{x_1, x_2, \ldots, x_n\}$ là tập hợp các mục dữ liệu. Một tập con $t = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\} \subseteq I$ gọ là một giao tác trên I. Một bảng gồm m giao tác $T = \{t_1, t_2, \ldots, t_m\}$ gọi là một cơ sở dữ liệu gia tác trên I. Với $X \subseteq I$ gọi là tập mục dữ liệu (itemset), mỗi tập con $S \subseteq T$ gọi là tập định dan giao tác (tidset). Để thuận tiện trong ký hiệu, ta viết X = ABC thay cho X = {A, B, C} viết S = 123 thay cho $S = \{t_1, t_2, t_3\}$.

Biểu diễn cơ sở dữ liệu giao tác

Có hai cách biểu diễn tập dữ liệu giao tác: Biểu diễn ngang và biểu diễn dọc.

Biểu diễn ngang: Một cơ sở dữ liệu là một danh sách những giao tác. Mỗi giac tác có mộ định danh giao tác (t_{id}) và một danh sách những mục dữ liệu trong giao tác đó.

Ví dụ 1.

Bảng 1. Biểu diễn ngang của cơ sở dữ liệu giao tác

Giao tác	Mục dữ liệu
t_1	A, B, E
t_2	B, D
t_3	B, C
t_4	A, B, D
t_5	A, C

Giao tác	Mục dữ liệu
t_6	B, C
t_7	A, C
t_8	A, B, C, E
t_9	A, B, C
t_{10}	A, B, E, G

Biểu diễn dọc: Một cơ sở dữ liệu là một danh sách những mục dữ liệu, với mỗi mục dữ liệ có một danh sách tất cả các định danh giao tác chứa mục dữ liệu này.

Bảng 2. Biểu diễn dọc của cơ sở dữ liệu giao tác

Mục dữ liệu	Định danh giao tác
A	$t_1, t_4, t_5, t_7, t_8, t_9, t_{10}$
B	$t_1, t_2, t_3, t_4, t_6, t_8, t_9, t_{10}$
C	$t_3, t_5, t_6, t_7, t_8, t_9$
D	t_2, t_4
E	t_1, t_8, t_{10}
G	t_{10}

Độ hỗ trợ (support): Ký hiệu ||X|| là số phần tử của tập X. Độ hỗ trợ của một tập mục dữ liệu X của cơ sở dữ liệu giao tác T, ký hiệu Supp(X), là số các giao tác trong T chứa X:

$$Supp(X) = ||\{t \in T \mid X \subseteq t\}||.$$

Tập mục dữ liệu thường xuyên (frequent itemset): Cho S_0 là một số nguyên, ta nói X là tập mục dữ liệu thường xuyên theo ngưỡng S_0 (gọi tắt là tập thường xuyên) nếu $Supp(X) \geq S_0$.

2. Luật kết hợp (Association rule)

Một luật kết hợp trên cơ sở dữ liệu giao tác T là một biểu thức có dạng $X \rightarrow Y$, với $X, Y \subseteq I$ và $X \cap Y = \phi$. Luật này có nghĩa là: X kéo theo Y.

Độ hỗ trợ của luật kết hợp $X \rightarrow Y$, được định nghĩa và ký hiệu như sau:

$Supp(X \rightarrow Y) = Supp(X \cup Y) = Supp(XY)$.

Luật kết hợp f: $X \rightarrow Y$ trên T là ***luật thường xuyên theo ngưỡng S_0*** nếu $Supp(X \rightarrow Y) \geq S_0$.

Với cơ sở dữ liệu giao tác cho ở bảng 1. Ta có:

f: $A \rightarrow B$ là luật thường xuyên theo ngưỡng $S_0 = 4$ vì $Supp(AB) = 5 > 4$

Độ tin cậy (Confidence) của luật $X \rightarrow Y$, được định nghĩa và ký hiệu:

$p = Conf(X \rightarrow Y) = Supp(XY) / Supp(X)$.

Với $C_0 \in (0, 1]$ ta nói f: $X \rightarrow Y$ là ***luật tin cậy (Confident rule) theo ngưỡng S_0 và C_0*** nếu f là luật thường xuyên theo ngưỡng S_0 và $Conf(X \rightarrow Y) \geq C_0$.

III. THUẬT TOÁN TĂNG TRƯỞNG

1. Phân hoạch dữ liệu

Gọi t_{id} là số thứ tự của t trong T mà ta gọi là định danh của giao tác t. Gọi P_T là tập tất cả các định danh giao tác của T: $P_T = \{t_{id} \mid t \in T\} = \{1, 2, \ldots, m\}$

Từ P_T, ta xây dựng n tập hợp $\{P_1, P_2, \ldots, P_n\}$ với P_i là tập hợp tất cả các định danh của các giao tác chứa x_i trong T: $P_i = \{t_{id} \in P_T \mid x_i \in t\} \Rightarrow Supp(\{x_i\}) = Supp(x_i) = ||P_i||$.

Thủ tục **Vertical** chuyên đổi cơ sở dữ liệu giao tác T thành biểu diễn dọc.

Procedure Vertical(T, P);

Input: *Cơ sở dữ liệu giao tác $T = \{t_1, t_2, \ldots, t_m\}$ trên $I = \{x_1, x_2, \ldots, x_n\}$;*

Output: *$P = \{P_1, P_2, \ldots, P_n\}$ với $P_i = \{t_{i_1}, t_{i_2}, \ldots, t_{i_k}\}$*

Method:

```
    For i := 1 to n do
        Pi := ∅;
    EndFor;
For j := 1 to m do
        For i := 1 to n do
            If xi ∈ tj then
                Pi := Pi ∪ {j};
            EndIf;
        EndFor;
    EndFor;
```

2. Các thủ tục chuẩn bị

a. Thủ tục Sort_List

Trong quá trình xử lí, các mục dữ liệu trong tập mục dữ liệu cần sắp xếp theo một thứ tự nhất định để phân hoạch dữ liệu vào các nhóm khác nhau. Trong bài báo này, các mục dữ liệu được sắp xếp theo độ hỗ trợ của nó trong cơ sở dữ liệu.

Trên $I = \{x_1, x_2, \ldots, x_n\}$ ta định nghĩa một quan hệ thứ tự như sau:

$$\forall x_i, x_j \in I : x_i \prec x_j \Leftrightarrow Supp(\{x_i\}) \leq Supp(\{x_j\}).$$

Trong suốt bài báo này, cho một tập mục dữ liệu X thì các thành phần của nó đã được sắp xếp theo thứ tự trên: $X = \{x_{i_1}, x_{i_2}, ..., x_{i_k}\} \Rightarrow Supp(x_{i_1}) \leq ... \leq Supp(x_{i_k})$.

Cho S_i là một số nguyên, gọi là ngưỡng cho trước, đặt F_{Si} là tập tất các các tập mục dữ liệu của T thỏa ngưỡng S_i.

$$F_{Si} = \{(X, Sup) \mid X \subseteq I \text{ và } Sup = Supp(X) \geq S_i\}$$

Từ định nghĩa, ta có: Nếu $S_1 \geq S_2$ thì $F_{S1} \subseteq F_{S2}$.

Trên F_{Si} ta định nghĩa một quan hệ thứ tự $\preccurlyeq_F$ như sau:

Với mọi $Z = \{z_1, \ldots, z_k\}$, $Y = \{y_1, \ldots, y_h\} \in F_{Si}$:

$Z \preccurlyeq_F Y \Leftrightarrow \exists i$ thỏa $0 < i < Min\{k, h\}$ sao cho $z_1 = y_1, \ldots, z_i = y_i$ và $z_{i+1} \prec y_{i+1}$.

Gọi G_k là tập hợp tất cả các tập có k phần tử của F_{Si}: $G_k = \{X \in F_{Si} \mid ||X|| = k\}$.

Thủ tục Sort_List sau đây sắp xếp các phần tử của G_k thành một tập theo thứ tự $\preccurlyeq_F$:

Procedure Sort_List(G_k);

Input: *G_k = {$X_1, \ldots, X_h$ | $X_i \in F_{Si}$ và ||X|| = k};*

Output: *$G_k = \{X_{j_1}, X_{j_2}, \ldots, X_{j_h} \mid X_{j_i} \preccurlyeq_F X_{j_{i+1}}, i \in \{1, \ldots, h\}\}$;*

Method:

```
For i := 1 to h-1 do
For j := i+1 to h do
    If Xj ≼F Xi then
            Tam := Xi, Xi := Xj; Xj:= Tam;
    EndIf;
EndFor;
EndFor;
```

3. Thủ tục Split_Class

Gọi C_{k+1} là tập các tập mục dữ liệu có k+1 phần tử, gọi là tập ứng viên, là các tập mục dữ liệu cần phải tính độ hỗ trợ để xác định nó có phải là tập thường xuyên hay không. C_{k+1} được thành lập từ tập $L_k = \{X \subseteq I \mid ||X|| = k \text{ và } Supp(X) \geq S_0\}$ như sau:

$$C_{k+1} = \{X \subseteq I \mid ||X|| = k+1 \text{ và } \forall Y \subset X, ||Y|| = k \Rightarrow Y \in L_k\}.$$

Để xác định phần tử của C_{k+1}, đầu tiên ta phân các phần tử của L_k thành các lớp, mỗi lớp gồm các phần tử có k-1 mục dữ liệu đầu tiên giống nhau, chỉ khác nhau mục thứ k. Thủ tục Split_Class thực hiện công việc đó.

Ta có: $L_k = \{X_1, \ldots, X_h\}$ với $X_i = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$, $i \in \{1, \ldots, h\}$.

Trên L_k ta định nghĩa quan hệ tương đương R_k như sau:

Với $X = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}, Y = \{y_{j_1}, y_{j_2}, \ldots, y_{j_k}\} \in L_k$: $X R_k Y \Leftrightarrow x_{i_1} = y_{j_1}, \ldots, x_{i_{k-1}} = y_{j_{k-1}}\}$.

Hai phần tử trong L_k là tương đương nhau nếu chúng chỉ khác nhau ở thành phần thứ k.

Đặt P_X là lớp tương đương chứa X: $P_X := \{Y \in L_k \mid X R_k Y\}$.

Như vậy, những phần tử trong một lớp tương đương có k-1 thành phần đầu tiên giống nhau. Phần giống nhau này gọi là tiền tố của P_X. Gọi $Z = \{x_{i_1}, x_{i_2}, \ldots, x_{i_{k-1}}\}$ là tiền tố của P_X, đặt:

$$G_{[Z]} = \{x_j \mid \{x_{i_1}, x_{i_2}, \ldots, x_{i_{k-1}}, x_j\} \in P_X\};$$

Với tập L_k đã được sắp xếp theo thứ tự $\preccurlyeq_F$ ở phần trên thì việc phân hoạch L_k thành các lớp tương đương là nhanh chóng. Thủ tục Split_Class tìm các hậu tố của từng lớp tương đương trong L_k:

Procedure Split_Class(L_k, G);

Input: *$L_k = \{X \subseteq I \mid |X| = k$ và $Supp(X) \geq So\}$.*

Output: *$G = \{G_{[Z]} \mid Z$ là tiền tố của P_X và $X \in L_k\}$*

Method:

$H = \{Y = \{x_{i_1}, x_{i_2}, \ldots, x_{i_{k-1}}\} \mid \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\} \in L_k\}$;

$G := \varnothing$;

For each $Z = \{z_{i_1}, z_{i_2}, \ldots, z_{i_{k-1}}\} \in H$ *do*

$G_{[Z]} := \varnothing$;

For each $X = \{x_{j_1}, x_{j_2}, \ldots, x_{j_k}\} \in L_k$ *do*

If $\{z_{i_1}, z_{i_2}, \ldots, z_{i_{k-1}}\} = \{x_{j_1}, x_{j_2}, \ldots, x_{j_{k-1}}\}$ *then*

$G_{[Z]} := G_{[Z]} \cup \{x_{j_k}\}$;

EndIf;

$G := G \cup G_{[Z]}$;

EndFor;

EndFor;

4. Thủ tục Candidate

Ứng dụng tính chất cơ bản của tập thường xuyên: Mọi tập mục dữ liệu có chứa một tập con không thường xuyên là một tập không thường xuyên, ta có thủ tục Candidate xây dựng tập ứng viên C_{k+1} từ L_k bằng cách:

Procedure Candidate(L_k, C_{k+1});

Input: *$L_k = \{X \subseteq I \mid |X| = k$ và $Supp(X) \geq So\}$;*

Output: *$C_{k+1} = \{X \subseteq I \mid |X| = k+1$ và $\forall Y \subset X, |Y| = k \Rightarrow Y \in L_k\}$;*

Method:

$C_{k+1} := \varnothing$;

Split_Class(L_k, G);

For each $G_{[Z]} \in G$ *do* //$Z = \{z_{i_1}, ..., z_{i_{k-1}}\}$;

For each $\{y_i, y_j\} \subseteq G_{[Z]}$ *do*

$W := \{Z \cup \{y_i, y_j\}\} := \{\{z_{i_1}, ..., z_{i_{k-1}}, y_i, y_j\}\}$

If $(\forall Y \subset W, |Y| = k \Rightarrow Y \in L_k)$ *then*

$C_{k+1} := C_{k+1} \cup W$;

EndIf;

EndFor;

EndFor;

5. Thủ tục Support

Với phân lớp P_T thành n tập hợp $P_1, P_2, \ldots, P_n$ do thủ tục Vertical thực hiện, ta có:

$$Supp(\{x_i\}) = \|P_i\|$$

Với $X = \{x_{i_1}, x_{i_2}, ..., x_{i_k}\} \Rightarrow Supp(X) = Supp(\{x_{i_1}, x_{i_2}, ..., x_{i_k}\}) = \| P_{i_1} \cap P_{i_2} \cap ... \cap P_{i_k} \|$.

Việc tính $P_i \cap P_j$ tốn một thời gian đáng kể. Một số ngôn ngữ lập trình có hỗ trợ việc tính giao của hai tập hợp với số phần tử của tập hợp không quá 255. Với P_i, P_j có số phần tử lớn không thể sử dụng sự hỗ trợ của ngôn ngữ lập trình. Với cơ sở dữ liệu có hàng triệu giao tác, các P_i phải được lưu trữ ở bộ nhớ ngoài bằng cấu trúc tập tin. Việc tính $P_{i_1} \cap P_{i_2} \cap ... \cap P_{i_k}$ tốn một thời gian khá lớn, do vậy, khi cài đặt ta phải dùng thuật toán tối ưu để tốn ít thời gian tính toán. Trong quá trình tính toán, ta xây dựng tập: SC = {(X, Sup) | Sup = Supp(X)} để lưu lại độ hỗ trợ của các tập mục dữ liệu ứng viên đã tính. Sau đó, khi cần tính độ hỗ trợ của $X \subseteq I$, trước hết ta tìm xem (X, ?) đã có trong SC hay không. Nếu đã có (X, Sup) trong SC thì ta có ngay Supp(X) := Sup mà không cần phải tính lại.

Procedure Support*(P, X, SC, Sup);*

Input: *- P = {P1, P2,..., Pn};* *- S0: ngưỡng tối thiểu;*

- X = {xi1,..., xik}; *- SC = {(X, Sup) | Sup = Supp(X)};*

Output: *- Sup = Supp(X);*

- SC = {(X, Sup) | Sup = Supp(X)};

Method:

If (X, Su) $\in$ *SC then*

Sup := Su

Else

U := Pi1;

For j := 2 to k do

U := U $\cap$ *Pij;*

EndFor;

Sup := $|U|$

SC := SC $\cup$ *{(X,Sup)};*

EndIf;

6. Khai phá tập thường xuyên

Khi tìm các tập thường xuyên theo ngưỡng S_0 lần đầu tiên, ta phải tính toàn bộ:

$$F_{S0} = \{X \mid X \subseteq I \text{ và } Supp(X) \geq S_0\}.$$

Sau đó, cần tìm tập các tập thường xuyên theo ngưỡng S_1, có hai khả năng xảy ra:

- Nếu $S_1 \geq S_0$ thì $F_{S1} \subseteq F_{S0} \Rightarrow F_{S1} = \{X \in F_{S0} \mid Supp(X) \geq S_1\}$: Ta không phải tính toán mà chỉ cần lọc trong SC những (X, Sup) với $Sup \geq S_1$.

- Nếu $S_1 < S_0$ thì $F_{S0} \subseteq F_{S1} \Rightarrow F_{S1} = F_{S0} \cup \{X \subseteq I \mid X \notin F_{S0}$ và $Supp(X) \geq S_1\}$. Do đó ta chỉ cần tính độ hỗ trợ của các tập ứng viên không có trong SC.

Thủ tục **Find_All_Frequent** tìm các tập thường xuyên theo ngưỡng S_0 của T:

Procedure Find_All_Frequent(P, SC, S_0, P_{S0});

Input: *- P = {P_1, P_2,..., P_n} là biểu diễn dọc của CSDL có m giao tác trên I.*

- SC = {(X, Sup) | X ⊆ I và Sup = Supp(X)};

- FSup = {S_i | S_i là ngưỡng tối thiểu đã khai phá};

- Ngưỡng tối thiểu S_0;

Output: *- F_{S0} = {X | X ⊆ I và Supp(X) ≥ S_0};*

- SC = {(X, Sup) | X ⊆ I và Sup = Supp(X)};

- FSup = {S_i | S_i là ngưỡng tối thiểu đã khai phá};

Method:

```
If (∃Si ∈ FSup) and (Si ≤ S0) then
    FS0 := ∅;
    For each (X, Sup) ∈ SC do
        If Sup ≥ S0 then
            FS0 := FS0 ∪ {X};
        EndIf;
    EndFor;
Else
    L1 = {{xi} | |Pi| ≥ S0};
    FS0 := L1;
    If SC = ∅ then
        SC := {({xi}, |Pi| | i ∈ P};
    EndIf;
    k := 1;
    Repeat
        Candidate(Lk, Ck+1);
        Lk+1 := ∅;
        For each X in Ck+1 do
            If (∃(X, Sup) ∈ SC) and (Sup ≥ S0) then
                Lk+1 := Lk+1 ∪ {X}
            Else
                Support(P, X , SC, Sup);
                If Sup ≥ S0 then
                    Lk+1 := Lk+1 ∪ {X};
                EndIf;
            EndIf;
```

EndFor;

$F_{S0} := F_{S0} \cup L_{k+1}$;

$k := k+1$;

Until $L_k = \varnothing$;

EndIf;

7. Thuật toán gia tăng

a. Phân lớp dữ liệu gia tăng

Khi dữ liệu tăng thêm với cơ sở dữ liệu giao tác T', ta chuyển đổi T' thành cơ sở dữ liệu biểu diễn dọc: $P' = \{P'_1, P'_2, \ldots, P'_n\}$ với $P'_i = \{t'_{id} \in P_{T'} \mid x_i \in t'\}$ bằng cách gọi thủ tục **Vertical(T', P');**

b. Thủ tục tính độ hỗ trợ trong dữ liệu gia tăng

Khi khai phá tập mục dữ liệu thường xuyên, thủ tục **Find_All_Frequent** chỉ dựa vào tập SC đã tính và các P_i. Do đó khi dữ liệu gia tăng thêm tập T', ta phải tính lại độ hỗ trợ của X trong mỗi (X, Sup) ∈ SC.

Với (X, Sup) ∈ SC, Sup1 là độ hỗ trợ của X trong T', độ hỗ trợ của X trong T ∪ T' là Sup + Sup1. Thủ tục **Support_Increment** tính độ hỗ trợ của X trong T'.

Procedure Support_Increment(P', X, Sup);

Input: *- $P' = \{P'_1, P'_2, \ldots, P'_n\}$ là biểu diễn dọc của T'.*

- $X = \{x_{i_1}, \ldots, x_{i_k}\}$;

Output: *- $Sup = |\{t'_{id} \mid t' \in T' \text{ và } X \subseteq t'\}|$*

Method:

$U := P_{i1}$;

For $j := 2$ to k do

$U := U \cap P'_{ij}$;

EndFor;

$Sup := |U|$

8. Thủ tục Increment

Thủ tục Increment tính lại độ hỗ trợ của tất cả các tập mục dữ liệu trong SC.

Procedure Increment(T, T', P, P', SC);

Input: *- $T = \{t_1, t_2, \ldots, t_m\}$; $T' = \{t'_1, t'_2, \ldots, t'_k\}$*

- $P = \{P_1, P_2, \ldots, P_n\}$, $P' = \{P'_1, P'_2, \ldots, P'_n\}$;

- $SC = \{(X, Sup) \mid Sup = Supp(X) \text{ trong } T\}$;

Output: *- $SC = \{(X, Sup) \mid Sup = Supp(X) \text{ trong } T'' = T \cup T'\}$;*

- $T = T \cup T'$; $P = \{P_1 \cup P'_1, P_2 \cup P'_2, \ldots, P_n \cup P'_n\}$

Method:

$SC' := \varnothing$;

For each (X, Sup) in SC do

Support_Increment(P', X, Su);

Sup := Sup + Su;

SC' := SC' ∪ {(X, Sup)};

EndFor;

For i := 1 to n do

For each j in P'i do

$P_i := P_i \cup \{j+m\}$

EndFor;

EndFor;

SC := SC';

$T := T \cup T'$;

Sau khi làm gia tăng dữ liệu ta được cơ sở dữ liệu giao tác mới $T = T \cup T'$ và biểu diễn dọc mới $P_i = P_i \cup P'_i$, với $i \in \{1, ..., n\}$. Để tìm tập các tập mục dữ liệu thường xuyên, ta gọi lại thủ tục **Find_All_Frequent** để tìm tất cả các tập thường xuyên theo ngưỡng S_0 của T. Vì tập Fsu là tập chứa các ngưỡng đã tính với dữ liệu T cũ không còn giá trị trong T mới, nên ta xóa tập Fsu .

8. Thử nghiệm thuật toán gia tăng

Chúng tôi đã cài đặt thuật toán và chạy trên máy IBM ThinkPad T43, quá trình thử nghiệm như sau: Cơ sở dữ liệu giao tác T có m giao tác trên n mục dữ liệu, số mục dữ liệu lớn nhất trong một giao tác là M_t, trung bình là A_t. Chạy thuật toán với ngưỡng độ hỗ trợ là S_0, thu được F_{S0} tập thường xuyên, phải tính SC_0 tập ứng viên, mất thời gian Sec_0 giây. Sau khi dữ liệu gia tăng thêm tập T' có m' giao tác có $M_{t'}$, $A_{t'}$ giống như giao tác của T, chạy thuật toán gia tăng với ngưỡng độ hỗ trợ là S_1, thu được F_{S1} tập thường xuyên, phải tính SC_1 tập ứng viên, mất thời gian Sec_1 giây. Chúng tôi chọn S_1 thỏa: $S_0/m = S_1/(m+m')$. Sau đó chạy thuật toán không gia tăng trên tập $T \cup T'$ cùng ngưỡng độ hỗ trợ S_1, thời gian chạy là Sec_2 giây. Tỷ lệ là số lần thời gian giảm: Sec_2/Sec_1. Kết quả thử nghiệm như trong bảng 3.

Bảng 3. So sánh thời gian chạy thuật toán gia tăng và không gia tăng

TT	m	n	M_t	A_t	S_0	SC_0	F_{S0}	Sec_0	m'	S_1	SC_1	F_{S1}	Sec_1	Sec_2	Tỉ lệ
1	$1*10^6$	50	14	3	500	16090	1.653	10193	$3*10^5$	650	16161	1648	2941	11785	4,0
2	$10*10^6$	20	9	2	10000	955	178	8830	$5*10^6$	15000	955	178	2921	13004	4,5
3	$20*10^6$	20	9	2	5000	1594	435	22341	$5*10^6$	6250	1908	536	4594	33543	7.3

IV. SO SÁNH THUẬT TOÁN TĂNG TRƯỞNG VÀ APRIORI

Thông thường, ma trận biểu diễn cơ sở dữ liệu giao tác có số hàng và số cột rất lớn (hàng tỷ hàng, hàng ngàn cột), tuy nhiên nó là một ma trận cực thưa. Trong hàng ngàn mặt hàng trong siêu thị (số mục dữ liệu), nhưng mỗi hóa đơn (một giao tác) chỉ só một số rất nhỏ các mặt hàng được mua.

Với biểu diễn dọc, T được chia thành n phần độc lập và được lưu trữ thành n tập tin trên bộ nhớ ngoài. Việc tính độ hỗ trợ của một tập ứng viên X là đếm số phần tử của phần giao các tập tin tương ứng với những mục dữ liệu có trong X. Để thực hiện phép giao của hai tập tin thành phần P_i, P_j, mỗi tập tin là một danh sách các số nguyên được sắp thứ tự, ta cần tối đa $Max(\|P_i\|, \|P_j\|)$ phép so sánh.

C_k là tập các tập ứng viên có k phần tử. Với quan hệ tương đương R_k (Trong III.2.2) phân C thành α lớp tương đương, lớp tương đương thứ i có β_i phần tử.

Để tính hết độ hỗ trợ của các tập trong C_k, thuật toán gia tăng cần số lần so sánh : Mỗi tậ mục dữ liệu $X = \{x_{i_1}, ..., x_{i_k}\}$ trong mỗi lớp tương đương, ta cần tối đa (k-1)*Max(P_i) phép so sán để tính (k-1) thành phần đầu tiên: $P = P_{i_1} \cap ... \cap P_{i_{k-1}}$. Trong lớp tương đương có β_i phần tử, ta cầ tối đa (k-1)*Max(P_j)*β_i*Max(P_j) = k*Max(P_j)*β_i, trung bình là k*Average(P_j)*β_i phép so sánh đ tính $P \cap P_{i_k}$, do đó ta cần có tất cả tối đa $\sum_{i=1}^{\alpha} k * Max(P_j) * \beta_i$, trung bình l $\sum_{i=1}^{\alpha} k * Average(P_j) * \beta_i$ phép so sánh.

Trong khi đó, nếu dùng thuật toán Apriori: Cần tối đa k phép so sánh k mục dữ liệu của X vớ các mục dữ liệu trong một giao tác để xem giao tác này có chứa X hay không, cần tối đa k*m phé so sánh để tính độ hỗ trợ của một tập ứng viên X, có tất cả $\|C_k\| = \sum_{i=1}^{\alpha} \beta_i$ tập ứng viên có k phần tử nghĩa là ta cần tối đa $\sum_{i=1}^{\alpha} k * m * \beta_i$, trung bình là $\sum_{i=1}^{\alpha} Average(\| t_i \|) * m * \beta_i$ phép so sánh để tín hết độ hỗ trợ của các tập trong C_k.

Việc chuyển đổi biểu diễn ngang $T = \{t_1, ..., t_m\}$ sang biểu diễn dọc $T = \{P_1, ..., P_n\}$ khôn làm thay đổi dung lượng của T. Ta có : $\sum_{j=1}^{n} \|P_j\| = \sum_{i=1}^{m} \|t_i\|$ với $\|P_j\|$ là số phần tử của P_j, $\|t_i\|$ là số mụ dữ liệu trong t_i. Ta có n*Average($\|P_j\|$) = m*Average($\|t_j\|$), do đó tỷ lệ thời gian thực hiện thu toán gia tăng và thuật toán Aprior là:

$$\sum_{i=1}^{\alpha} k * Average(P_j) * \beta_i / \sum_{i=1}^{\alpha} Average(\| t_i \|) * m * \beta_i \sim k*Average(\|P_j\|)/ m*Average(\|t_j\|)$$

k/n.

Như vậy, khi n lớn, độ hỗ trợ nhỏ, tức số lượng mục dữ liệu trong tập thường xuyên nhỏ t thuật toán gia tăng chạy nhanh hơn nhiều lần.

V. KẾT LUẬN

Thuật toán gia tăng đã giải quyết được một số vấn đề sau:

Với một cơ sở dữ liệu giao tác có kích thước lớn, không thể đưa hết vào bộ nhớ trong để : lí, thuật toán phân hoạch thành n tập nhỏ lưu ở bộ nhớ ngoài, mỗi lần xử lí chỉ đưa một số tập ph hoạch vào bộ nhớ trong. Việc tính độ hỗ trợ của một tập ứng viên chỉ là phép giao của một số ph hoạch tương ứng, tốc độ xử lí nhanh.

Khi đi tìm tập mục dữ liệu thường xuyên theo một ngưỡng S_0 nào đó, cần tính độ hỗ trợ c tất cả các tập ứng viên và lưu vào: SC = {(X, Sup) | X là tập ứng viên}. Khi dữ liệu chưa gia tăn cần khai phá tập các mục dữ liệu thường xuyên có độ hỗ trợ S_1, nếu $S_1 \geq S_0$, công việc rất đơn gi là lọc từ SC ra tập $F_{S1} = \{X | (X, Sup) \in SC \text{ và } Sup \geq S_1\}$, ngược ta chỉ cần tính độ hỗ trợ cho c tập ứng viên chư có trong SC, đồng thời cập nhật lại SC.

Theo thời gian, dữ liệu tăng thêm một tập T', thuật toán gia tăng tính toán lại độ hỗ trợ của các tập ứng viên trong SC chỉ dựa vào tập dữ liệu tăng thêm T', không cần tính toán lại từ đầu, do đó tiết kiệm nhiều thời gian tính toán.

TÀI LIỆU THAM KHẢO

1. Qiankun Zhao, Sourav S. Bhowmick - Association Rule Mining: A survey technical report, CAIS, Nanyang Technological University, Singapore **116** (2003).
2. B. Goethals - Survey on frequent pattern mining, Technical report, Helsinki Institute for Information Technology, 2003.
3. R. Agrawal, T. Imielinski, A. Swami - Mining Associations between Sets of Items in Massive Databases, In: Proc.of the ACM-SIGMOD Int'l Conf. on Management of Data, 1993, pp. 207-216.
4. J. Han, Pei H, and Yin Y. - Mining Frequent Patterns without Candidate Generation, In: Proc. Conf. on the Management of Data (SIGMOD'00, Dallas, TX). ACM Press, New York, NY, USA., 2000, pp 1-12.
5. R. Agrawal, H. Mannulla, R. Srikant, H. Toivonen, A. I. Verkamo - Fast discovery of association rules, In: Advances in Knowledge discovery and data Mining, AAAI Press, 1996, pp. 307-328.
6. Claudio Lucchese, Salvatore Orlando, Raffaele Perego - Fast and memory efficient algorithm to mine frequent closed itemsets, IEEE Transaction On Knowledge and Data Engineering **18** (1) (2006) 21-36.
7. Mohammed J. Zaki and Ching-Jui Hsiao - Charm: Efficient algorithm for mining closed itemsets and their lattice structure, IEEE Transactions on knowledge and Data engineering **17** (4) (2005) http://www.cs.rpi.edu/_zaki.
8. Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura - LCM ver.2: Efficient mining algorithms for frequent/closed/maximal itemsets IEEE ICDM'04 Workshop FIMI'04 (International Conference on Data Mining, Frequent Itemset Mining Implementations), 2004. http://research.nii.ac.jp/~uno/papers/04111cm2.pdf.
9. Vicky Choi - Faster algorithms for constructing a concept (Galois) lattice arXIV:cs.DM/0602069 v2 1 Jun 2006.
10. Ashok Savasere, Edward Omiecinski, and Shamkant Navathe - An efficient algorithm for mining association rules in large databases, In: Proc. of the 21th VLDB Conference, Zurich, Swizerland, 1995, pp. 432-444.
11. Shakil Ahmed, Frans Coenen, and Paul Leng - Tree-based partitioning of data for association rule mining, Knowledge and Information systems **10** (3) (2006) 315-331.
12. Weiqiang Lin, Mehmet A. Orgun, and Graham J. Williams - An overview of temporal aata mining, In: Proc of the Australisian Data Mining Workshop, ADM02, Sydney, Australia, 2002, pp. 83-90.
13. Srivatsan Laxman, P. S. Sastry - A survey of temporal data mining, Sadhana **31** (2) (2006) 173–198.
14. Nguyễn Xuân Huy, Đoàn Văn Ban, Nguyễn Hữu Trọng, Huỳnh Văn Đức - Thuật toán khai thác dữ liệu tăng trưởng, Tạp chí Khoa học và Công nghệ (2007) (nhận đăng).

15. Nguyễn Hữu Trọng - Thuật toán khai thác tập thường xuyên hiệu quả dựa trên kỹ thuật phân lớp dữ liệu, Tạp chí Tin học và Điều khiển học (2007) (nhận đăng).

SUMMARY

AN ALGORITHM FOR DATA MINING ON LARGE INCEMENT DATABASE BASED ON THE PARTITION DATA

With all algorithms for datamining on fixed database were announced, whenever the data increments, the problem has to be solved from the beginning again; whenever a frequent itemset is sought with another support, the seeking needs to be start again. Some algorithms for datamining on incrementing database were announced. This paper presents an algorithm for solving the data mining problems on large incrementing database, based on the vertical data format. Whenever the data increments or solving with another support, the algorithm only processes on new transactions and connected to the previous result without going back to the starting point.

Địa chỉ:

Trường Đại học Nha Trang.

Nhận bài ngày 16 tháng 5 năm 2006