Một số cải tiến thuật toán Index-BitTableFI cho khai thác tập tin phổ biến

Improvements of Index-BittableFI Algorithm for Mining Frequent Itemsets

Lê Hoài Bắc, Nguyễn Thị Bảo Chi, Võ Đình Bảy

***Abstract**: Index-BitTableFI is an algorithm based on BitTable which is very effective in recent (Song & Yang, 2008). It finds out itemsets based on BitTable in vertical and horizontal, and also sets up sorting array and equivalent computing method to fast identify itemsets which occur concurrently with representative items. Although Index-BitTableFI algorithm reduces considerablely cost of finding out candidate itemsets and computing the support, but if number of transactions and items is large then intersection computing of vector-bits in BitTable still costs time. Besides, finding out frequent itemsets in depth has not used property of equivalent computing method yet. To resolve this problem, some improvements for improving more performance of Index-BitTableFI algorithm are proposed in this research.*

I. GIỚI THIỆU

Từ khi bài toán khai thác luật kết hợp được đề xuất vào năm 1993 đến nay, đã có nhiều thuật toán được phát triển để khai thác tập phổ biến như: Apriori [2], DCP[5], CBAR[7], FP-growth [4], Eclat [8], v.v… . Gần đây, các tiếp cận dựa trên định dạng dữ liệu dọc được đề xuất, trong số này phải kể đến hai thuật toán là BitTableFI [3] và Index-BitTableFI [6]. Với thuật toán BitTableFI, cấu trúc BitTable được dùng theo cả chiều ngang và chiều dọc để nén dữ liệu. Việc phát sinh các tập ứng viên trở nên nhanh hơn và việc tính toán độ hỗ trợ tương ứng cũng thực thi hiệu quả hơn so với thuật toán Apriori [1]. Tuy nhiên trong tình huống với số lượng lớn tập phổ biến, tập nhiều phần tử hoặc ngưỡng hỗ trợ nhỏ, thuật toán này phải chịu những chi phí bất thường như chi phí tính toán lớn, việc lưu trữ các ứng viên đòi hỏi không gian bộ nhớ lớn và tính toán độ hỗ trợ của các ứng viên này rất phức tạp. Để giải quyết vấn đề này, thuật toán Index-BitTableFI được đề xuất, cấu trúc BitTable được sử dụng theo cả chiều ngang và chiều dọc, sự tìm kiếm kép được thực hiện và không gian tìm kiếm được giảm đáng kể.

Tuy nhiên, ngoài việc nén dữ liệu BitTable theo chiều dọc ta cần nén dữ liệu theo chiều ngang để vận dụng phương pháp tính toán tương đương, trong khi số lượng item thường nhỏ hơn rất nhiều lần so với số lượng giao tác. Mặt khác thuật toán chưa vận dụng triệt để tính chất của phương pháp tính toán tương đương, vì thế trong bài báo này, chúng tôi đề xuất một số cải tiến bao gồm: không cần lưu trữ dữ liệu theo chiều ngang, việc tính toán tương đương dựa trên dữ liệu dọc sẵn có, đồng thời vận dụng triệt để phương pháp này khi tìm kiếm các itemset phổ biến theo chiều sâu.

II. TẬP PHỔ BIẾN VÀ THUẬT TOÁN INDEX-BITTABLEFI

II.1. Một số định nghĩa

Cho cơ sở dữ liệu (CSDL) D gồm tập các item là I = $\{i_1, i_2, \ldots, i_n\}$ và tập các giao tác T = $\{t_1, t_2, \ldots, t_m\}$ trong đó mỗi giao tác t chứa một tập các item, nghĩa là t = $\{i_{k1}, i_{k2}, \ldots, i_{kj}\}$. Trong đó $i_{kj} \in I$ ($1 \leq kj \leq n$).

Định nghĩa 1: độ hỗ trợ của một itemset X, kí hiệu sup(X), chính là số các giao tác trong D có chứa X.

Định nghĩa 2: Cho X là một itemset, X được gọi là tập phổ biến nếu sup(X) ≥ *minsup*, trong đó *minsup* là ngưỡng độ hỗ trợ tối thiểu do người dùng xác định.

Nhiệm vụ chính của quá trình khai thác tập phổ biến là tìm tất cả các itemset trong CSDL có độ hỗ trợ lớn hơn hoặc bằng *minsup*.

Bổ đề [1]: Mọi tập con của một tập phổ biến đều là tập phổ biến, mọi tập cha của một tập không phổ biến cũng là tập không phổ biến.

Ví dụ: Cho CSDL D như trong Bảng 1

Bảng 1. CSDL D

Tid	Item
1	A, B, C, E, F
2	A, C, G
3	E
4	A, C, D, E, G
5	A, C, E, G
6	E
7	A, B, C, E, F
8	A, C, D
9	A, C, E, G
10	A, C, E, G

Với *minsup*=2, ta có sup(AC)=8>*minsup* nên AC là tập phổ biến.

II.2. Cấu trúc BitTable

BitTable là tập các số nguyên mà sự hiện diện của nó biểu thị cho các item. Nếu item thứ i xuất hiện trong giao tác t thì bit thứ i của t trong BitTable sẽ mang giá trị 1, ngược lại sẽ mang giá trị 0. Với dữ liệu được nén, thì BitTable được sử dụng theo chiều dọc. Nếu kích cỡ (số giao tác) của item là S, kích cỡ của cơ sở dữ liệu là lớn hơn kích cỡ W của bộ nhớ thì kích cỡ của mảng BitTable sẽ là: $\frac{S}{W}+1$ được sử dụng để lưu trữ dữ liệu nén.

Bảng 3 là biểu diễn thập phân của các item trên bảng 2. Chẳng hạn, xét item A (Bảng 2), ta có dãy bit là 11011011,11, nghĩa là cần 2 byte để lưu dãy bit này trong đó byte đầu chứa giá trị là 219 và byte thứ 2 chứa giá trị 3.

Bảng 2. Minh họa dữ liệu được nén vào bảng BitTable

Tid	A	B	C	D	E	F	G
1	1	1	1	0	1	1	0
2	1	0	1	0	0	0	1
3	0	0	0	0	1	0	0
4	1	0	1	0	1	0	1
5	1	0	1	0	1	0	1
6	0	0	0	0	1	0	0
7	1	1	1	0	1	1	0
8	1	0	1	1	0	0	0
9	1	0	1	0	1	0	1
10	1	0	1	0	1	0	1

Bảng 3. Bảng BitTable

A	B	C	D	E	F	G
219	130	219	1	190	130	80
3	0	3	0	3	0	3

II.3. Mảng Index và cách xây dựng

Mảng Index được xây dựng dựa trên hàm sau: g(X)={t∈D | X ⊆ t} là tập các giao tác có chứa itemset X.

Ví dụ: g(A) = {1, 2, 4, 5, 7, 8, 9,10}, g(B) = {1, 2, 4, 5, 7, 8, 9,10}. Để tính g(AB), chúng ta chỉ cần lấy phần giao của g(A) với g(B), nghĩa là g(AB) = g(A)∩g(B)= {1, 2, 4, 5, 7, 8, 9, 10}∩{1, 7} = {1,7}.

Định nghĩa 4: Mảng Index là một mảng có kích thước m. Trong đó, m là số lượng các tập phổ biến 1-item. Mỗi phần tử của mảng là bộ đôi (item, subsume).

Trong đó :

$$subsume(item) = \{j \in I | item \prec j \wedge g(item) \subseteq g(j)\}$$

j đứng sau item

Nghĩa là: subsume gồm tập các item j, sao cho j đứng sau item và tập các giao tác chứa item j phải bao phủ các tập giao tác có chứa item.

Thuật toán 1: Xây dựng bảng Index [6]

Input: CSDL D, ngưỡng độ hỗ trợ tối thiểu *minsup.*

Output: Mảng Index.

1. Duyệt D và xóa những item không phổ biến.
2. Sắp xếp danh sách tập phổ biến 1-item tăng dần theo sup: $a_1,a_2,..,a_m$.
3. Với mỗi phần tử j của mảng Index thực hiện:
4. Gán Index[j].item =a_j
5. Xây dựng BitTable từ cơ sở dữ liệu.
6. Với mỗi phần tử j của mảng Index thực hiện:
7. Gán Index[j].subsume=Ø.
8. Gán candidate = $\bigcap_{t \in g(index[j].item)} t$
9. Với mỗi i > j thực hiện
10. Nếu giá trị của bit thứ i trong candidate là 1 thì
11. Đưa index[i].item vào index[j].subsume
12. Xuất mảng Index

Ví dụ: Xét CSDL trên bảng 1 với *minsup*=2, ta có kết quả như bảng 4, bảng 5 và bảng 6.

Bảng 4. CSDL D sau khi xóa bỏ những item không phổ biến và sắp xếp tăng dần theo độ hỗ trợ.

Tid	Item	Sắp xếp item
1	A, B, C, E, F	B, F, A, C, E
2	A, C, G	G, A, C
3	E	E
4	A, C, D, E, G	D, G, A, C, E
5	A, C, E, G	G, A, C, E
6	E	E
7	A, B, C, E, F	B, F, A, C, E
8	A, C, D	D, A, C
9	A, C, E, G	G, A, C, E
10	A, C, E, G	G, A, C, E

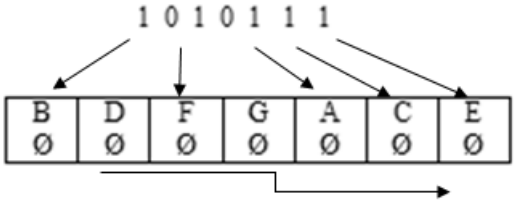
Bảng 5. Bảng Index ban đầu

B	D	F	G	A	C	E
Ø	Ø	Ø	Ø	Ø	Ø	Ø

Bước kế tiếp thực hiện việc tìm subsume(B). Do trong bảng BitTable, tại cột B chỉ có Tid 1 và Tid 7 có giá trị bằng 1, điều này có nghĩa rằng chỉ có Tid 1 và Tid 7 chứa item B mà thôi. Vậy ta lấy Tid 1 giao với Tid 7:

Candidate = 010111&1010111=1010111.

Nhận thấy có 5 bit 1 trong candidate



Bit 1 đầu tiên tương ứng với item B, bit 1 thứ 2 tương ứng với item F, các bit 1 tiếp theo tương ứng với các item: A, C, E. Vậy tính từ vị trí sau item B trở đi ta có subsume(B) = FACE. Tiến trình tương tự, ta sẽ có mảng Index như trong Bảng 6.

Bảng 6. Bảng Index kết quả

B	D	F	G	A	C	E
FACE	AC	ACE	AC	C	Ø	Ø

II.3.1. Định lý 1 [6]

Nếu Index[j].item là tập phổ biến và sup(Index[j].item)=*minsup* thì sẽ không tồn tại item i nào sao cho Index[j].item $\prec$ i và i$\notin$ Index[j].subsume(item) để cho (Index[j].item$\cup$i) là tập phổ biến.

II.3.2. Định lý 2 [6]

Nếu item là phần tử đại diện và subsume(item) = $a_1,a_2,..a_k$, khi kết hợp item với (2^k-1) tập con khác rỗng của a_1, a_2,.., a_k thì độ hỗ trợ của chúng là như nhau và bằng sup(item).

Thuật toán 2: Index-BitTableFI [6]

Input: Mảng Index, *minsup.*

Output: Danh sách tập phổ biến.

1. Với mỗi thành phần Index[j] của mảng Index thực hiện.
2. Xuất Index[j].item và sup(Index[j].item)
3. Nếu Index[j].subsume=Ø thì
4. Nếu (sup(Index[j].item) >*minsup*) thì
5. Depth_First(Index[j].item, t(Index[j].item))
   // *t(Index[j].item) là tập các tập phổ biến 1-phần tử*

// đứng sau Index[j].item trong mảng Index

6. Ngược lại thực hiện
7. Với mỗi s_item⊆Index[j].subsume thực hiện
8. Xuất (Index[j].item∪s_item) và sup(Index[j].item).
9. Nếu (sup(Index[j].item) >minsup) thì
10. Gán tail = t(Index[j].item) \ items∈ Index[j].subsume
11. Depth_First (Index[j].item,tail)
12. Với mỗi tập con s_item⊆Index[j].subsume thực hiện
13. Depth_First (Index[j].item ∪ s_item,tail)

Thuật toán 3: Depth_First [6]

Input: Itemset, tail.

Output: Xuất ra các tập phổ biến và độ hỗ trợ của chúng

Depth_First(itemset,tail)

1. Nếu tail=Ø thì return
2. Với mỗi thành phần i∈ tail thực hiện
3. Gán f_itemset=itemset∪i
4. Nếu sup(f_itemset) ≥ *minsup* thì
5. Xuất ra f_itemset và sup(f_itemset)
6. Gán tail = tail \ i *//loại i ra khỏi tail.*
7. Depth_First(f_itemset,tail).

Hình 1 minh họa kết quả của thuật toán Index-BitTableFI trên CSDL ở Bảng 1. Có thể thấy, do subsume(B) = FACE nên ta chỉ việc kết hợp B với các tập con của FACE để tạo ra 16 tập phổ biến với độ hỗ trợ chính là độ hỗ trợ của B. Chính nhờ đều này mà nhiều itemset không cần tính độ hỗ trợ nên thuật toán tiết kiệm được chi phí.

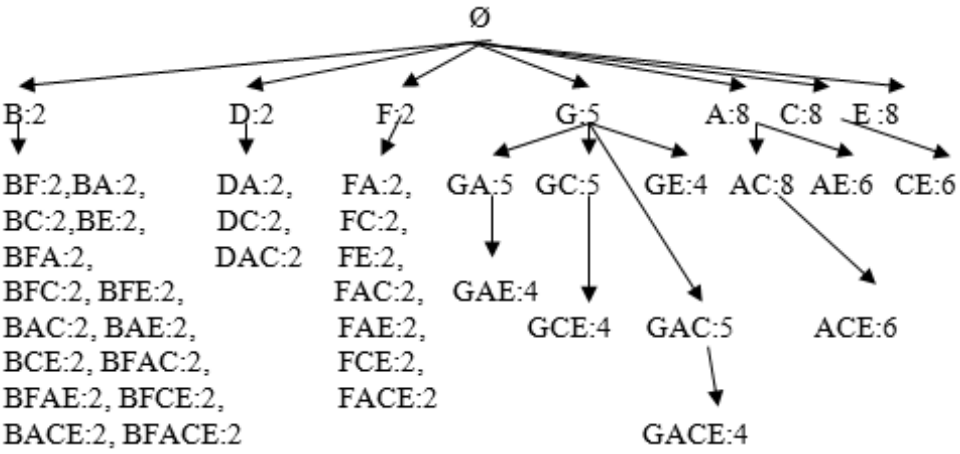
III. MỘT SỐ CẢI TIẾN

III.1. Nhận xét 1

Việc tính toán Index[i].subsume ở thuật toán 1 (bước 7 - 8) được thực hiện bằng cách giao tất cả các giao tác có chứa mục dữ liệu Index[i].item trong bảng BitTable theo chiều ngang. Trong trường hợp số lượng Tid, số item là lớn thì quá trình giao các tid có chứa Index[i].item sẽ mất nhiều thời gian. Trong khi số lượng item thường nhỏ hơn rất nhiều so với số lượng Tid. Mặt khác, subsume(item) gồm những item j đứng sau item (định nghĩa 1). Quá trình tìm tất cả các tập phổ biến phải lưu trữ dữ liệu theo 2 dạng (ngang và dọc) như thế sẽ cần nhiều bộ nhớ để lưu trữ.

Giải pháp:

- Chỉ sử dụng bảng BitTable đã được nén theo chiều dọc để tìm nhanh subsume (item).
- Việc kiểm tra g(item) ⊆ g(j) được thực hiện đơn giản bằng cách kiểm tra Tid(item) có là con của Tid(j) hay không.



Hình 1. Danh sách các tập phổ biến.

Thuật toán 4: xây dựng bảng Index_S

Input: CSDL D, ngưỡng hỗ trợ *minsup*

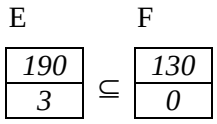
Output: Mảng Index

1. Duyệt D và xóa những item không phổ biến.
2. Sắp xếp danh sách tập phổ biến 1-item tăng dần theo sup: $a_1,a_2,..,a_m$.
3. Với mỗi phần tử j của mảng Index thực hiện:
4. Gán Index[j].item = a_j.
5. Xây dựng BitTable từ cơ sở dữ liệu.
6. Với mỗi phần tử j của mảng Index thực hiện:
7. Gán Index[j].subsume=Ø.
8. Với mỗi i>j thực hiện
9. Nếu g(Index[j].item) ⊆ g(Index[i].item) thì đưa index[i].item vào index[j].subsume

Ví dụ: với bảng BitTable (Bảng 3) ta có:

	A	B	C	D	E	F	G
BitTable	219	130	219	17	190	130	88
	3	0	3	0	3	0	3

Ta thực hiện phép kiểm tra như được trình bày ở Hình 2.



Hình 2. Minh họa tính subsume(E)

Nếu kết quả phép kiểm tra là đúng thì đưa F vào subsume(E).

Tương tự ta tiến hành kiểm tra E với G.

III.2. Nhận xét 2

Với kết quả danh sách tập phổ biến ở Hình 1 ta có nhận xét sau:

a. Đối với thuật toán 2 (Index-BitTableFI):

- Ở bước thứ 9, nếu sup(Index[j].item) >*minsup* thì thuật toán mở rộng theo chiều sâu (gọi Depth_First) trên danh sách tail cho Index[j].item.
- Sau đó bước thứ 12 lại mở rộng theo chiều sâu trên danh sách tail cho (Index[j].item∪s_item), trong đó s_item⊆Index[j].subsume.
- Theo định lý 2, ta có sup(Index[j].item)=sup(Index[j].item∪s_item) cho nên khi mở rộng theo chiều sâu cho (Index[j].item∪s_item) trên tail, cũng giống như mở rộng chiều sâu cho (Index[j].item) trên tail. Hai bước này hoàn toàn giống nhau về kết quả.

b. Đối với thuật toán 3 (Depth_Fist):

- Mỗi lần kiểm tra sup(f_itemset) xem có thỏa *minsup* hay không là một quá trình giao tid nên sẽ tốn nhiều chi phí.

Chính vì những điều này, ngay bước thứ 9 của thuật toán Index-BitTableFI ta ghi nhận lại kết quả những mục dữ liệu trong tail mà Index[j].item kết hợp được, đồng thời lưu trữ lại độ hỗ trợ của nó. Sau đó, đến bước thứ 13 thay vì tìm kiếm theo chiều sâu cho các tập con của Index[j].subsume, ta chỉ kết hợp với danh sách kết quả ở bước trên với độ hỗ trợ sẵn có. Vì vậy, sẽ tiết kiệm chi phí tính toán độ hỗ trợ, quá trình xử lý sẽ nhanh hơn.

Thuật toán 5: Index_BitTableFI_S

Input: mảng Index, *minsup.*

Output: Danh sách tập phổ biến.

1. Với mỗi thành phần Index[j] của mảng Index thực hiện
2. Xuất Index[j].item và sup(Index[j].item)
3. Nếu Index[j].subsume=Ø thì
4. Nếu (sup(Index[j].item) >*minsup*) thì Depth_First(Index[j].item, t(Index[j].item)) *//t(Index[j].item) là tập các tập phổ biến 1-phần tử //đứng sau Index[j].item trong mảng Index*
6. Ngược lại thực hiện
7. Với mỗi s_item⊆Index[j].subsume thực hiện
8. Xuất (Index[j].item∪s_item) và sup(Index[j].item)
9. Nếu (sup(Index[j].item) >*minsup*) thì

10. Gán tail = t(Index[j].item) \ items∈ Index[j].subsume
11. Depth_First_S(Index[j].item,tail,kq)
12. Với mỗi tập con s_item⊆Index[j].subsume thực hiện
13. Với mỗi phần tử s_kq trong danh sách kq
14. Xuất (s_item∪s_kq.item) và s_kq.sup

Thuật toán 6: Depth_First_S

Input: Itemset, tail, kq.

Output: Xuất ra các tập phổ biến và độ hỗ trợ của chúng

Depth_First_S(itemset,tail,kq)

1. Nếu tail=Ø thì return
2. Với mỗi thành phần i∈ tail thực hiện
3. Gán f_itemset=itemset ∪ i
4. Nếu sup(f_itemset) ≥ *minsup* thì

4.1 Thêm i và sup(f_itemset) vào danh sách kq

5. Xuất ra f_itemset và sup(f_itemset)
6. Gán tail=tail \ i

//loại i ra khỏi tail

7. Depth_First_S(f_itemset,tail,kq)

Trong đó kq là danh sách dùng để lưu trữ những mục dữ liệu trong tail mà kết hợp được Index[j].item cùng với độ hỗ trợ tương ứng.

Ví dụ: Khi xét tới item G, ta có sup(G)=5 và subsume(G)=AC.

- Xuất: G:5.
- Kết hợp G với tất cả các tập con subsume(G)=AC và xuất: GA:5, GC:5, GAC:5.
- Tính tail(G)=E, ta tính sup(GE)=4>*minsup* nên xuất: GE:4.
- Vì sup(GE) = 4, mà sup(G) = sup(GA) = sup(GC) = sup(GAC) nên xuất: GAE:4, GCE:4, GACE:4.

Kết quả quá trình tìm tập phổ biến sẽ nhanh hơn và giảm bớt số lần tính toán độ hỗ trợ của tập ứng viên.

VI. KẾT QUẢ THỰC NGHIỆM

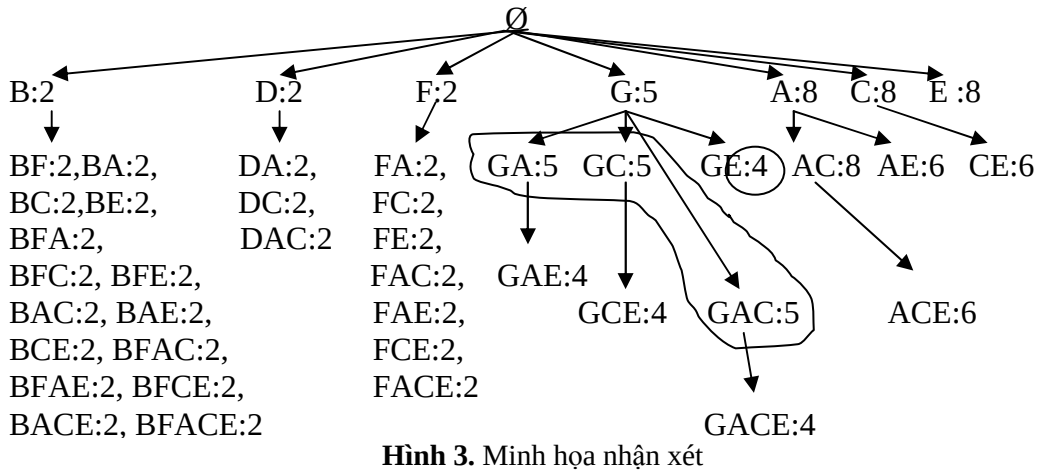
Các kết quả thực nghiệm được thực hiện trên máy laptop HP, duo core 2.1GHz, 3GB RAM, các thuật toán đều được cài đặt trên C# 2008.

Khai thác luật kết hợp dựa trên dữ liệu chi tiết các cuộc gọi điện thoại và các nguồn số liệu cước bổ sung khác như cước thông tin di động, cước quốc tế, cước internet do GPC, VTI và VDC cung cấp tại Viễn thông Ninh Thuận.

Các thuộc tính chính trong dữ liệu khách hàng gồm:

Bảng 7. CSDL khách hàng

Tên thuộc tính	Ý nghĩa
Ma_kh	Mã khách hàng
So_dt	Số điện thoại
Ten_kh	Tên khách hàng
Dc_kh	Địa chỉ khách hàng
Ma_donvi	Mã đơn vị, khách hàng thuộc đơn vị nào quản lý



Hình 3. Minh họa nhận xét

Các thuộc tính chính trong dữ liệu chi tiết cuộc gọi gồm:

Bảng 8. CSDL chi tiết cuộc gọi

Tên thuộc tính	Ý nghĩa
Ma_kh	Mã khách hàng
So_may	Số chủ gọi
Sm_den	Số bị gọi
Huong	Hướng gọi
Datc	Ngày gọi
Gio_bd	Giờ bắt đầu
Gio_kt	Giờ kết thúc
Timeo	Thời gian gọi
Loai	Là liên tỉnh, quốc tế, nội hạt
Voip	Gọi theo truyền thống, hay 171
Nha_cc	Nhà cung cấp dịch vụ

Từ những dữ liệu trên tiến hành tổng hợp thành dữ liệu doanh thu khách hàng, gồm một số thuộc tính chính như sau:

Bảng 9. CSDL doanh thu khách hàng

Tên thuộc tính	Ý nghĩa
Ma_donvi	Khách hàng thuộc đơn vị (huyện) nào
Cuoc_lt	Doanh thu sử dụng dịch vụ gọi liên tỉnh
Cuoc_171lt	Doanh thu sử dụng dịch vụ gọi IP 171 liên tỉnh
Cuoc_internet	Doanh thu sử dụng internet
Cuoc_dvu	Doanh thu sử dụng dịch vụ
Cuoc_dnk	Doanh thu sử dụng doanh nghiệp khác
Cuoc_qt	Doanh thu cước quốc tế
Cuoc_171qt	Doanh thu cước 171 quốc tế
Doanh thu	Tổng doanh thu

Chú thích: Ma_kh trong Bảng 8 là khóa ngoại của Ma_kh trong Bảng 8, Ma_donvi trong Bảng 9 là khóa ngoại của Ma_donvi trong Bảng 7. Các loại cước sẽ được tính thông qua các chi tiết cuộc gọi trong Bảng 8. Chẳng hạn: Dựa trên thuộc tính So_may, Sm_den và Voip có thể biết được cuộc gọi đó là liên tỉnh hay quốc tế? có sử dụng dịch vụ 171 hay không? v.v..

Chọn nguồn dữ liệu từ danh sách khách hàng và chi tiết cuộc gọi năm 2008 và năm 2009, trong 6 tháng đầu năm 2010, từ tháng 01/2010 đến tháng 06/2010, tiến hành tích hợp các loại cước thành dữ liệu doanh thu khách hàng.

Tiến hành rời rạc hóa dữ liệu cho các thuộc tính số và thuộc tính hạng mục để chuyển về thuộc tính dạng nhị phân. Bảng 10 trình bày cách thức rời rạc hóa dữ liệu cước viễn thông trong đó các thuộc tính LT (liên tỉnh) là cước liên tỉnh trong bảng 9, LT171 (liên tỉnh 171) là cước liên tỉnh có sử dụng dịch vụ 171 trong bảng 9, QTE (quốc tế) là cước quốc tế, QTE171 là cước quốc tế có sử dụng dịch vụ 171, v.v..

Bảng 10. Dữ liệu doanh thu sau khi chuyển đổi về dạng giao tác

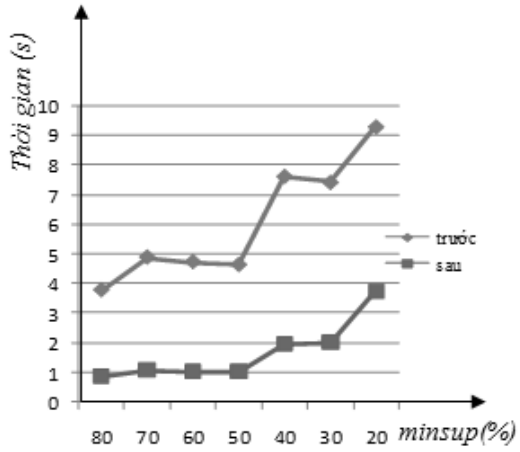
Tid	Item
1	LT<100.000, QTE<50.000, INTER≥100.000, DNK<100.000, 100.000≤DTHU<200.000
5	LT≥100.000, QTE<50.000, DVU>10.000, QTE171<50.000, DTHU≥200.000
199	LT≥100.000, LT171≥50.000, DNK≥100.000, DTHU≥200.000
1587	LT<100.000, DVU≥10.000, INTER≥100.000, LT171≥50.000, 100.000≤DTHU<200.000
1747	DNK<100.000, DTHU<100.000
1970	LT171<50.000, INTER<100.000, 100.000≤DTHU<200.000

Như vậy, dữ liệu doanh thu khách hàng sau khi chuyển đổi về dạng giao tác có tổng số giao tác như sau:

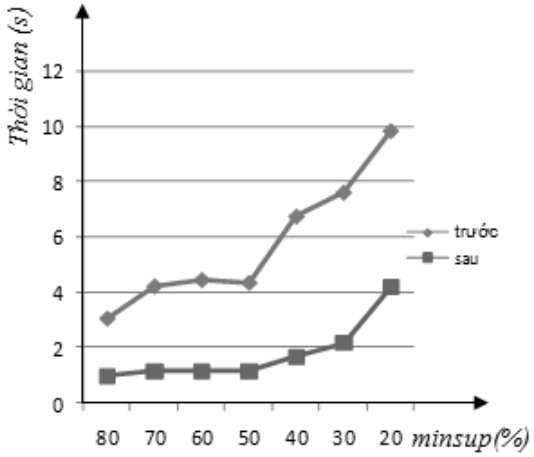
Bảng 11. Số lượng giao tác dữ liệu cước

Năm	Số lượng Item	Số lượng mẫu tin
2008	22	580.588
2009	22	643.050
01-06/2010	22	320.685

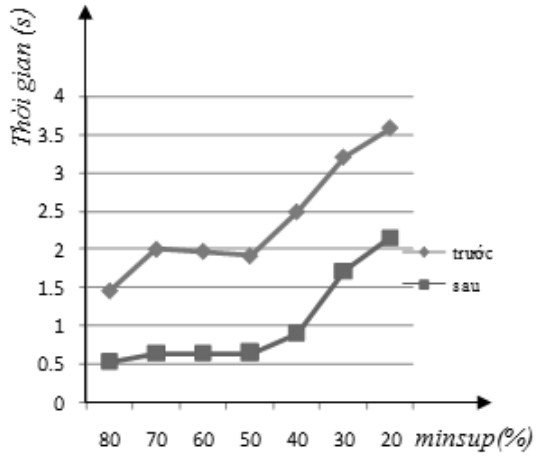
Để so sánh hiệu quả của thuật toán Index-BitTableFI trước và sau khi cải tiến, thử nghiệm trên tập dữ liệu cước tại Viễn thông Ninh Thuận, tập dữ liệu Accident (lấy từ http://fimi.cs.helsinki.fi/data/) ta có kết quả như sau:



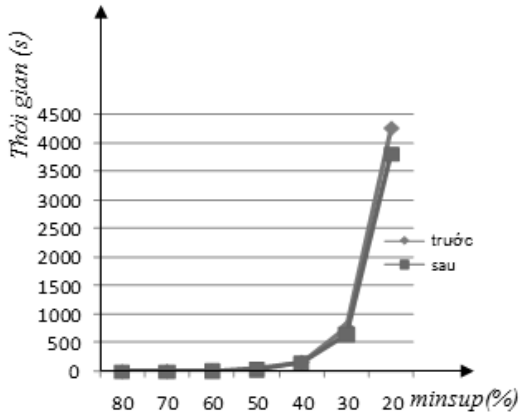
Hình 4. Thời gian thực thi trên dữ liệu cước năm 2008



Hình 5. Thời gian thực thi trên dữ liệu cước năm 2009



Hình 6. Thời gian thực thi trên dữ liệu cước năm 2010



Hình 7. Thời gian thực thi trên dữ liệu accident

Các kết quả trên cho thấy thuật toán cải tiến thường hiệu quả hơn so với Index-BitTableFI nguyên thủy trên CSDL cước viễn thông. Tuy nhiên, đối với CSDL Accident thì thuật toán cải tiến không cải thiện đáng kể về mặt thời gian. Điều này là do số lượng subsume của các item trên CSDL Accident không đáng kể.

V. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Bài báo trình bày một số cải tiến của thuật toán Index-BitTbaleFI bao gồm: 1) Chỉ tổ chức dữ liệu BitTable theo chiều dọc để tiết kiệm bộ nhớ; 2) Kiểm tra subsume đơn giản bằng cách xét xem g(item) có là con của g(j) hay không? Công việc này không tốn nhiều thời gian; 3) Cải tiến phương pháp duyệt theo chiều sâu nhằm hạn chế việc tính phần giao giữa các tid. Các cải tiến này đã được thực nghiệm trên CSDL cước viễn thông Ninh Thuận và kết quả tương đối khả quan.

Nghiên cứu cách thức khai thác hiệu quả tập luật là một trong những chủ đề sẽ được quan tâm nghiên cứu trong thời gian tới. Ngoài ra, nghiên cứu cách thức cập nhật lại tập phổ biến khi CSDL thay đổi cũng là một hướng nghiên cứu trong tương lai.

LỜI CẢM ƠN

Nghiên cứu này được tài trợ bởi Quỹ phát triển khoa học và công nghệ quốc gia (NAFOSTED) trong đề tài mã số 102.01-2012.17

TÀI LIỆU THAM KHẢO

[1] R. AGRAWAL, T. IMILIENSKI, A. SWAMI. *Mining association rules between sets of large databases*. Proceedings of the ACM SIGMOD International Conference on Management of Data, Washington, DC, 1993, pp. 207-216.

[2] R. AGRAWAL, R. SRIKANT. *Fast algorithms for mining association rules*. Proceedings of International Conference on Very Large Data Base, Santiago, Chile, 1994, pp.478-499.

[3] J. DONG, M. HAN. *BitTableFI: An efficient mining frequent itemsets algorithm*, Knowledge-Based Systems 20(4) (2007), pp. 329–335.

[4] J.HAN, J.PEI, Y.YIN. *Mining frequent patterns without candidate generation*. Proceedings of the 2000 ACM-SIGMOD International Conference on Management of Data (SIGMOD'00), Dallas, Texas, 2000, pp. 1–12.

[5] S. ORLANDO, P. PALMERINI, R. PEREGO. *Enhancing the Apriori algorithm for frequent set counting.* Proceedings of Third International Conferrence on Data Warehousing and Knowledge Discovery (DaWak'01), Munich, 2001, pp.71-82.

[6] W.SONG, B.YANG. *Index-BitTableFI: An improved algorithm for mining frequent itemsets*, Knowledge-Based Systems 21 (2008), pp. 507-513.

[7] Y.J. TSAY, J.Y. CHIANG, *CBAR: An efficient method for mining association rules*, Knowledge-Based Systems 18 (2-3) (2005), pp. 99-105.

[8] M.J. ZAKI. *Scalable algorithms for association mining*. IEEE Transactions on Knowledge and Data Engineering 12(3) (2000), pp. 372-390.

Nhận bài ngày: 20/08/2012

SƠ LƯỢC VỀ CÁC TÁC GIẢ

LÊ HOÀI BẮC

Hiện là Phó Trưởng khoa, Trưởng Bộ môn Khoa học Máy tính, khoa CNTT, Trường Đại học Khoa học Tự nhiên Tp HCM.

Hướng nghiên cứu: Trí tuệ nhân tạo, Tính toán mềm và Data mining.

Email: lhbac@fit.hcmuns.edu.vn.

NGUYỄN THỊ BẢO CHI

Sinh năm 1976.

Tốt nghiệp Đại học năm 1998 tại Đại học Đà Lạt và thạc sĩ tại Trường Đại học Khoa học Tự nhiên TP. HCM năm 2011.

Hiện công tác tại Viễn Thông Ninh Thuận.

Lĩnh vực nghiên cứu: Khai thác dữ liệu.

Điện thoại: 0918951167,

Email: binhchichau@yahoo.com

VÕ ĐÌNH BẢY

Sinh năm 1974.

Tốt nghiệp Đại học năm 2002, Cao học năm 2005 và Tiến sĩ năm 2011 tại Trường Đại học Khoa học Tự nhiên TP.HCM.

Qiện đang là Trưởng phòng quản lý Nghiên cứu Khoa học và CNTT, Trường Cao đẳng CNTT TP. HCM.

Hướng nghiên cứu: Khai thác luật kết hợp, Khai thác mẫu tuần tự, Phân lớp dữ liệu, Khai thác dữ liệu trên cơ sở dữ liệu phân tán, Khai thác dữ liệu trên cơ sở dữ liệu tăng trưởng.

Điện thoại: 0903696987,

Email: bayvodinh@gmail.com