Kỷ yếu Hội nghị Khoa học Quốc gia lần thứ IX "Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR'9)"; Cần Thơ, ngày 4-5/8/2016
DOI: 10.15625/vap.2016.00031

KHAI THÁC TẬP PHỔ BIẾN CÓ TRỌNG SỐ DỰA TRÊN CẤU TRÚC N-LIST

Bùi Danh Hường [1], Võ Đình Bảy[2], Nguyễn Duy Hàm[3]

[1] Trung tâm Ngoại ngữ - Tin học, Đại học An ninh Nhân dân
[2] Khoa công nghệ thông tin, Đại học Công nghệ TP. Hồ Chí Minh
[3]Bộ môn Toán – Tin học, Đại học An ninh Nhân dân

buidanhhuong@gmail.com, vd.bay@hutech.edu.vn, duyham@gmail.com

TÓM TẮT— *Khai thác tập phổ biến là bài toán quan trọng trong khai thác dữ liệu. Đã có nhiều phương pháp khác nhau được đề xuất để giải quyết bài toán này. Trong đó, cấu trúc N-list được đề xuất bởi Deng với việc sử dụng hướng tiếp cận lai giữa cây FP và cây liệt kê đã đạt được hiệu quả đáng khích lệ. Tuy nhiên phương pháp này mới chỉ khai thác trên cơ sở dữ liệu (CSDL) nhị phân truyền thống. Trong bài báo này, chúng tôi đề xuất một cấu trúc mở rộng của N-list là WN-list (Weighted N-list) để giải quyết bài toán khai thác tập phổ biến có trọng số trên CSDL trọng số. Đầu tiên, một số định lý được phát triển để tính toán độ phổ biến trọng số của itemset, sau đó thuật toán NFWI được đề xuất trên cơ sở các định lý đó để khai thác nhanh tập phổ biến có trọng số. Các thử nghiệm trên nhiều loại cơ sở dữ liệu (thưa và dày) cho thấy phương pháp đề xuất hiệu quả hơn so với các phương pháp khai thác tập phổ biến có trọng số hiện có, đặc biệt là khi ngưỡng phổ biến nhỏ.*

Từ khóa— *Khai thác dữ liệu, khai thác tập phổ biến, tập phổ biến có trọng số, WN-list.*

I. GIỚI THIỆU

Từ khi được đề xuất bởi Agrawal và các đồng sự [1], khai thác tập phổ biến (FI) đã trở thành một chủ đề nghiên cứu quan trọng trong lĩnh vực khai thác dữ liệu. Nhiều phương pháp khác nhau đã được đề xuất để giải quyết bài toán này, góp phần nâng cao hiệu quả khai thác FI. Các phương pháp hiện có có thể được chia làm 4 nhóm chính như sau:

- **Các phương pháp theo hướng tiếp cận Apriori**: Hướng tiếp cận Apriori [2] đặc trưng bởi việc sinh và kiểm tra các ứng viên cấp k+1 từ các ứng viên cấp k thông qua việc quét CSDL. Nhược điểm của phương pháp này là tốn thời gian và bộ nhớ do phải quét CSDL nhiều lần.
- **Các phương pháp theo hướng tiếp cận sử dụng cây FP (Frequent Pattern - tree)**: Đại diện là các thuật toán FP-Growth [3] và FP-Growth* [4], tiếp cận theo hướng nén CSDL và khai thác FI trên cây FP [3]. Đầu tiên phương pháp này nén toàn bộ CSDL lên cây FP, sau đó duyệt cây để khai thác tập phổ biến. Ưu điểm của phương pháp này là tiết kiệm bộ nhớ do nén được CSDL trên cây FP, tuy nhiên lại tốn thời gian duyệt cây FP để khai thác FI, đặc biệt là khi số nút trên cây nhiều.
- **Các phương pháp theo hướng tiếp cận sử dụng cây IT (Itemset Tid-set tree)**: Đại diện điển hình là các thuật toán Eclat [5], dEclat [6] và DBV-FI [7], tiếp cận theo hướng khai thác FI trên cây IT [5], một cấu trúc lưu trữ cơ sở dữ liệu theo chiều dọc, trong đó mỗi item được biểu diễn tương ứng với một Tid-set (set of transaction ID - là tập tất cả các giao dịch có chứa item đó). Ưu điểm của phương pháp loại này là chỉ cần quét CSDL một lần, đồng thời tính nhanh độ phổ biến thông qua xác định giao của các Tid-set. Tuy nhiên phương pháp loại này tốn bộ nhớ để lưu trữ Tid-set, điều này dẫn đến thời gian khai thác FI chưa được tối ưu.
- **Các phương pháp lai**: Đại diện điển hình là các thuật toán PrePost [8], NSFI [9] và PrePost+ [10] tiếp cận theo hướng nén CSDL trên cây PPC (Pre-Post Code) và từ đó biểu diễn CSDL và khai thác FI trên cấu trúc N-list [8]. Phương pháp này vừa có ưu điểm của phương pháp theo họ cây FP - đó là khả năng nén CSDL trên cây PPC, vừa có cả ưu điểm của phương pháp theo họ cây IT - đó là tính nhanh độ phổ biến dựa vào giao của các N-list. Do đó các phương pháp lai ghép như PrePost, NSFI hay PrePost+ đã thể hiện hiệu quả vượt trội trong khai thác FI.

CSDL trọng số (Weighted Database - WD) là loại CSDL có nhiều trong các ứng dụng thực tế và trong các hệ thống thông minh. Khai thác tập phổ biến có trọng số (Frequent Weighted Itemsets - FWI) trên WD đã được quan tâm từ rất sớm [11,12] và được quan tâm nhiều trong thời gian gần đây [13,14,15]. Trong đó [14] đề xuất cây WIT (Weighted Itemset Tid-set tree) là một mở rộng của cây IT theo tiếp cận Eclat, nhưng phương pháp này khá tốn thời gian do bộ nhớ sử dụng chưa được tối ưu. Một cải tiến gần đây là cấu trúc IWS (Interval word segment) được đề xuất bởi [15], đây là tiếp cận theo hướng Bit-vector bằng việc cắt bỏ các đoạn byte 0 liên tiếp trong biểu diễn Tid-set của các itemset trên Bit-vector được biểu diễn dưới dạng các word (2 byte). Tuy nhiên tiếp cận này không hiệu quả trên các CSDL dày.

Trong bài báo này, chúng tôi đề xuất thuật toán NFWI dựa trên cấu trúc WN-list, một mở rộng của cấu trúc N-list, để giải quyết bài toán khai thác FWI trên WD. Kết quả thực nghiệm trên nhiều loại CSDL cho thấy thuật toán NFWI hiệu quả hơn các thuật toán khai thác FWI hiện có, đặc biệt thể hiện rõ ở các ngưỡng phổ biến nhỏ.

Phần còn lại của bài báo được trình bày như sau: Phần 2 trình bày về các nghiên cứu liên quan. Phần 3 trình bày về cấu trúc WN-list, các khái niệm, định nghĩa liên quan. Phần 4 sẽ đưa ra thuật toán NFWI để khai thác FWI trên CSDL có trọng số. Phần 5 là các thử nghiệm trên nhiều loại CSDL khác nhau để đánh giá hiệu quả của thuật toán đề xuất. Và cuối cùng là kết luận và hướng nghiên cứu tương lai được trình bày trong phần 6.

II. NGHIÊN CỨU LIÊN QUAN

Định nghĩa 1. Một CSDL trọng số (*WD*) được định nghĩa là 1 bộ ba $\langle T, I, W\rangle$, trong đó $T = \{t_1, t_2, .., t_m\}$ là tập các giao dịch, $I = \{i_1, i_2, .., i_n\}$ là tập các item, $W = \{w_1, w_2, \ldots, w_n\}$ là tập các trọng số của các item trong tập *I*.

Ví dụ 1. Trong Bảng 1 thể hiện một cơ sở dữ liệu có trọng số. Tập các giao dịch $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$ (Bảng 1A). Tập các item $I = \{A, B, C, D, E, F\}$. Tập trọng số của các item $W = \{0.8, 0.1, 0.5, 0.9, 0.2, 0.3\}$ (Bảng 1B).

Bảng 1. Ví dụ về *WD*

A	
Transaction Database	
ID	**Items**
1	*A, B, D, E*
2	*B, E, F*
3	*A, B, F*
4	*A, B, C, E*
5	*A, B, C, D, E*
6	*B, C, D, E, F*

B	
Item weight	
Item	**Weight**
A	0.8
B	0.1
C	0.5
D	0.9
E	0.2
F	*0.3*

Định nghĩa 2. Trọng số giao dịch (*tw*) của một giao dịch t_k được định nghĩa như sau:

$$tw(t_k) = \frac{\sum_{i_j \in t_k} w_j}{|t_k|} \quad (1)$$

Trong đó: - w_j là trọng số của item i_j

- $|t_k|$ là số lượng item xuất hiện trong giao dịch t_k

Định nghĩa 3. Độ hỗ trợ trọng số (*ws*) của một itemset *X* được định nghĩa như sau:

$$ws(X) = \frac{\sum_{t_k \in t(X)} tw(t_k)}{\sum_{t_k \in T} tw(t_k)} \quad (2)$$

Trong đó: $t(X)$ là tập các giao dịch có chứa itemset *X*

Định nghĩa 4. Cho một ngưỡng cho trước là *minws*. Một itemset có *ws* thỏa mãn ngưỡng *minws* này được gọi là FWI theo ngưỡng *minws* đó. Bài toán khai thác FWI từ WD là bài toán tìm tất cả các FWI thỏa mãn ngưỡng *minws* cho trước.

Bài toán khai thác FWI được đề xuất lần đầu tiên bởi Ramkumar và đồng sự [11], trong nghiên cứu này, các tác giả đã đưa ra mô hình mô tả khái niệm về luật kết hợp có trọng số, trong đó đề xuất thuật toán WIS để khai thác FWI. Tiếp theo là nghiên cứu của Tao và đồng sự [13] dựa trên độ đo *tw* được tính bằng trung bình cộng của trọng số các item trong giao dịch, và *ws* của một itemset được xác định bằng thương giữa tổng các *tw* của các giao dịch có chứa itemset đó chia cho tổng *tw* của tất cả giao dịch. Theo cách tiếp cận này thì giá trị *ws* của itemset vừa phản ánh được mức độ xuất hiện của itemset trong các giao dịch, vừa thể hiện được mức độ quan trọng khác nhau của các giao dịch, đồng thời thỏa mãn tính chất bao đóng giảm một cách tự nhiên. Tuy nhiên, thuật toán do Tao và đồng sự đề xuất dựa vào việc sinh ứng viên theo kiểu Apriori nên cần đọc CSDL nhiều lần, dẫn đến tốn thời gian xử lý. Sau đó, Võ và các đồng sự [14] đề xuất cách thức lưu trữ trọng số trên cây WIT, một mở rộng của cây IT. Do chỉ phải đọc CSDL một lần, cùng với áp dụng chiến lược Diffset để khai thác FWI trên cây WIT, nên phương pháp này tỏ ra hiệu quả hơn phương pháp theo hướng tiếp cận Apriori trước đó. Hạn chế của phương pháp này là ở chỗ tốn bộ nhớ để lưu trữ Tid-set. Gần đây, Nguyen và các đồng sự [15] cũng với tiếp cận Eclat đã đề xuất một cải tiến để giảm bớt bộ nhớ lưu trữ Tid-set bằng cấu trúc IWS. IWS loại bỏ các đoạn word "0" (2 byte) ở trong biểu diễn bit của Tid-set dựa trên tiếp cận Bit-vector. Tuy nhiên, thuật toán này chỉ có hiệu quả trên các cơ sở dữ liệu thưa, và không hiệu quả trên các CSDL dày.

Cấu trúc N-list được đề xuất đầu tiên bởi Deng và đồng sự [8] để biểu diễn CSDL truyền thống và khai thác FI thông qua thuật toán Prepost với hai bước như sau:

Bước 1, CSDL được loại bỏ các item không thỏa ngưỡng và sắp xếp lại các item trong từng giao dịch theo tăng dần của độ phổ biến. Từng giao dịch đã sắp xếp được đọc và nén vào một cấu trúc cây PPC. Mỗi nút của cây PPC là một bộ gồm 5 giá trị (*item-name*, *count*, *child-list*, *pre-order*, *post-order*), trong đó *item-name* và *count* là tên và số lần của item được đăng ký tại nút đó, *child-list* là danh sách các nút con, *pre-order* và *post-order* là thứ tự của các nút khi duyệt cây PPC theo hướng trên-xuống-trái-sang và trên-xuống-phải-sang. Một PP-code của một nút *N* trên cây là một bộ 3 giá trị (*pre-order*, *post-order*, *count*). Quan hệ tổ tiên của hai nút N_1 và N_2 có thể xác định thông qua việc so sánh PP-code của

chúng: N_1 là tổ tiên của N_2 nếu và chỉ nếu (N_1.*pre-order* < N_2.*pre-order*) và (N_1.*post-order*>N_2.*post-order*). Vì vậy, các PP-code của các nút trên cây phản ánh toàn bộ cấu trúc của cây PPC, từ đó cũng phản ánh toàn bộ CSDL.

Bước 2, N-list của các 1-itemset sẽ được khởi tạo, chính là danh sách các PP-code của 1-itemset đó trên cây PPC. Độ phổ biến của 1-itemset chính là tổng các giá trị *count* trong các PP-code trong N-list của nó. N-list của một *k*-itemset được xác định bằng cách giao các N-list của hai (*k*-1)-itemset tương ứng. Thuật toán giao hai N-list có độ phức tạp $O(m+n)$, trong đó *m* và *n* là số phần tử của hai N-list tương ứng. Độ phổ biến của một *k*-itemset cũng được tính bằng tổng các giá trị *count* trong các PP-code trong N-list của nó. Danh sách N-list của các 1-itemset được sử dụng để tạo ra danh sách N-list của các 2-itemset và cứ thế, theo cách đó việc khai thác FI được diễn ra.

Trong bài báo này, chúng tôi áp dụng cấu trúc WN-list, một mở rộng của cấu trúc N-list, để biểu diễn và giải quyết bài toán khai thác FWI trên WD.

III. CẤU TRÚC WN-LIST BIỂU DIỄN WD

Định nghĩa 5. (WN-Tree) Cây WN là một cấu trúc bao gồm một nút cha là "null" và các nút con, trong đó mỗi nút con là một bộ ⟨*item-name, child-list, pre, pos, weight*⟩:

- *item-name* là tên của nút, chính là tên của item đại diện cho nút.
- *child-list* là danh sách các nút con của nút hiện tại.
- *pre* là thứ tự của nút khi duyệt cây từ trên xuống trái sang.
- *pos* là thứ tự của nút khi duyệt cây từ trên xuống phải sang.
- *weight* là khối lượng của nút, được xác định thông qua tổng *tw* của các giao dịch đi qua nút.

Dựa vào Định nghĩa 4, Thuật toán 1 (**Construction_WN_Tree)** xây dựng cây WN được tiến hành như sau:

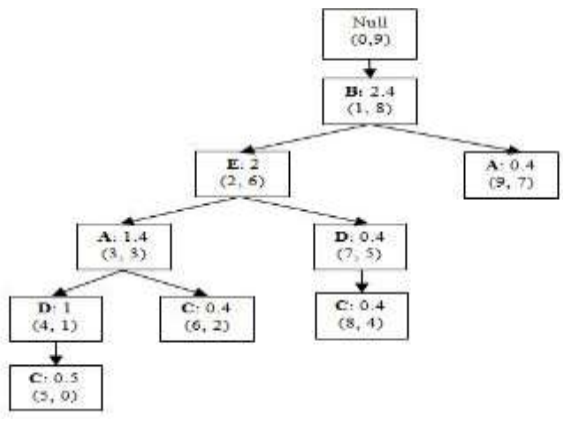
- Đọc CSDL lần đầu để tính *tw* của các giao dịch, tổng trọng số (*sumtw*) của các giao dịch và ws của các 1-itemset. Gọi I_1 là tập các 1-itemset có *ws*≥*minws*, sắp xếp I_1 theo chiều giảm dần theo giá trị *ws* của các 1-itemset.
- Đọc CSDL lần hai và xây dựng cây WN bằng cách sau:
  - Khởi tạo nút cha "null" cho cây WN
  - Với mỗi giao dịch đọc từ CSDL, loại bỏ các item không thuộc I_1 và sắp xếp các item còn lại theo thứ tự trong I_1. Sau đó đọc lần lượt từng item trong giao dịch và chèn vào cây từ gốc theo cách thức sau: kiểm tra xem item có là một trong các nút con của nút đang xét hay không, nếu có thì chuyển nút đang xét sang nút con thì cộng giá trị *tw* của giao dịch hiện tại vào *weight* của nút con đó, nếu không thì tạo ra một nút con mới có *weight* nhận giá trị *tw* của giao dịch hiện tại và cũng chuyển nút đang xét sang nút con mới tạo, và cứ thực hiện như thế cho hết các item có trong giao dịch.

Ví dụ 2. Với CSDL trong Bảng 1 và ngưỡng *minws* =0.5 sau khi quét CSDL lần đầu, ta có CSDL với từng giao dịch được loại bỏ bớt các item có *ws* nhỏ hơn *minws* và được sắp xếp theo thứ tự giảm dần của *ws* như trong Bảng 2. Sau khi áp dụng Thuật toán 1, ta xây dựng được cây WN như trong Hình 1**.**

Bảng 2. CSDL trong Bảng 1 với ngưỡng *minws*=0.5

A		
Sorted transaction database		
Transaction	**Items**	***tw***
1	*B, E, A, D*	0.50
2	*B, E*	0.20
3	*B, A*	0.40
4	*B, E, A, C*	0.40
5	*B, E, A, D, C*	0.50
6	*B, E, D, C*	0.40
Sum of *tw* values (*sumtw*)		2.40

B	
I_1	
Item	***ws***
B	1
E	0.83
A	0.75
D	0.58
C	*0.54*



Hình 1. Cây WN của CSDL trong Bảng 2

Định nghĩa 6. (WN-code) Một WN-code của một nút trên cây WN là một bộ ba giá trị (*pre, pos, weight*) của nút đó.

Định lý 1 [8]**.** Cho hai WN-code $C_1(x_1, y_1, w_1)$ và $C_2(x_2, y_2, w_2)$, C_1 là một tổ tiên của C_2 nếu và chỉ nếu $x_1<x_2$ và $y_1>y_2$.

Định nghĩa 7. (WN-list của một item) Cho một cây WN, WN-list của một item là dãy có thứ tự các WN-code của các nút đại diện cho item đó trên cây WN, trong đó các WN-code trong dãy được sắp xếp tăng dần theo giá trị *pre* của chúng.

Định nghĩa 8. (WN-list của một *k*-itemset) Cho *XA* và *XB* là hai (*k*-1)-itemset có cùng phần tiền tố là *X*, trong đó *A* xếp sau *B* theo thứ tự trong I_1. *WL*(*XA*) và *WL*(*XB*) là hai WN-list tương ứng của *XA* và *XB*. Ta có *WL*(*XAB*) là WN-list của *k*-itemset *XAB* được xác định như sau:

1. Với mỗi cặp $C_i(x_i, y_i, w_i) \in WL(XA)$ và $C_j(x_j, y_j, w_j) \in WL(XB)$, nếu C_j là tổ tiên của C_i thì ta thêm (x_j, y_j, w_i) vào *WL*(*XAB*).

2. Duyệt *WL*(*XAB*) và tổ hợp các WN-code có cùng (*pre*, *post*) thành một WN-code mới có trọng số là tổng của các giá trị trọng số trong các WN-code đang xét.

Dựa vào Định nghĩa 8 ta dễ dàng xây dựng được Thuật toán 2 (**WL_Intersection**) thực hiện phép giao giữa hai WN-list với độ phức tạp tuyến tính.

Định lý 2. Cho *X* là một itemset với WN-list của *X* là $WL(X) = \{(x_1, y_1, w_1), (x_2, y_2, w_2), \dots, (x_n, y_n, w_n)\}$. Độ phổ biến trọng số của *X* là $ws(X)$ được tính như sau:

$$ws(X) = \frac{\sum_{i=1}^{n} w_i}{\sum_{t_k \in T} tw(t_k)} \tag{3}$$

Chứng minh: Không mất tính tổng quát, ta giả sử itemset $X = A_1A_2...A_m$, trong đó A_i đứng sau A_{i+1} theo thứ tự trong I_1. Theo cách xác định WN-list trong định nghĩa 7, ta có:

(a) Với mỗi WN-code $C_i(x_i, y_i, w_i) \in WL(X)$, tồn tại một nút trên cây WN có *item-name* = A_m, *pre* = x_i và *pos* = y_i tương ứng với (x_i, y_i).

(b) w_i là tổng các weight của các nút có *item*-name = A_1 trong cây con thuộc cây WN có gốc là A_m trong (a).

Gọi T_i là tập các giao dịch chứa *X* đi qua cây con có gốc là A_m(*pre* = x_i, *pos* = y_i), lúc đó mỗi T_i sẽ tương ứng với một WN-code (x_i, y_i, w_i) và theo (a) và (b) ta có:

$$w_i = \sum_{t_k \in T_i} tw(t_k) \tag{c}$$

$$T(X) = \bigcup_{i=1}^{n} T_i \tag{d}$$

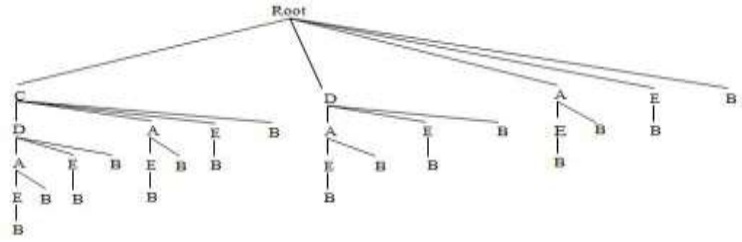
Theo công thức (2), (c) và (d) ta có:

$$ws(X) = \frac{\sum_{t_k \in T(X)} tw(t_k)}{\sum_{t_k \in T} tw(t_k)} = \frac{\sum_{i=1}^{n} \sum_{t_k \in T_i} tw(t_k)}{\sum_{t_k \in T} tw(t_k)} = \frac{\sum_{i=1}^{n} w_i}{\sum_{t_k \in T} tw(t_k)}$$

Định lý 2 đã được chứng minh.

IV. THUẬT TOÁN NFWI KHAI THÁC FWI TRÊN WD

Chúng tôi sử dụng cây liệt kê (set-enumeration tree) [16] để đơn giản hóa quá trình duyệt khai thác FWI. Cụ thể, với CSDL ví dụ trong Bảng 2, ta có cây liệt kê như Hình 2.



Hình 2. Cây liệt kê của CSDL trong Bảng 2

Vận dụng các định nghĩa và định lý đã trình bày ở phần trước, chúng tôi đề xuất thuật toán NFWI để khai thác FWI trên WD như Hình 3.

Thuật toán 3: Thuật toán NFWI

Input: CSDL có trọng số *WD* và ngưỡng *minws* ***Output:*** *FWI*, là tập tất cả các tập phổ biến có trọng số. ***Method name:*** **NFWI_Algorithm**(*WD*, *minws*)
1. Call **Construction_WN_Tree**(*WD*, *minws*) to generate *Tree* and I_1 2. Scan *Tree* to generate WN-lists of items in I_1 3. Let *FWI*←I_1 4. Call ***Find_FWI***(I_1, ∅) 5. return *FWI*

```
Procedure Find_FWI(Lk,Prek)
6.   Prenext = Prek
7.   for i = Lk.size downto 2 do
8.        Prenext← Li
9.        Lnext = ∅
10.       for j = i-1 downto 1 do
11.            WLij = WL_Intersection(WLi, WLj)
12.            if ws(WLij) ≥minws then
13.                 FWI← {Prek∪{Li}∪{Lj}}
14.                 Inext←Li
15.       if Lnext ≠ ∅ then   Find_FWI(Lnext, Prenext)
16.       remove last item of Prenext
```

Hình 3. Thuật toán NFWI

Thuật toán NFWI hoạt động như sau: đầu tiên xây dựng cây WN và phát sinh WN-list của các 1-FWI. Tiếp đó duyệt theo cây liệt kê và áp dụng thuật toán giao hai WN-list để xác định WN-list của một *k*-itemset từ WN-list của hai (*k*-1)-itemset tương ứng. Trong quá trình tính toán WN-list, áp dụng định lý 2 để tìm và cập nhật tập FWI.

Chi tiết của thuật toán NFWI được thể hiện ở trong Thuật toán 3. Sau khi xây dựng cây WN và phát sinh tập các 1-FWI là I_1, thì gọi hàm ***Find_FWI***(L_k, Pre_k) để bắt đầu tiến trình xử lý khai thác FWI. Ở đây, L_k là tập các item cuối và Pre_k là tập các tiền tố giống nhau của các itemset thuộc về lớp hiện tại. Dòng 4 thể hiện thời điểm gọi hàm **Find_FWI** lần đầu tiên, trong đó $L_k = I_1$ và $Pre_k = \emptyset$. Dòng 6, 8, 9, 14 và 16 là nơi khai báo và tính toán các biến mới là L_{next} và Pre_{next}, các biến này được sử dụng làm tham số để gọi hàm đệ quy cho lớp tìm kiếm tiếp theo (Dòng 15). Ở dòng 7-14, thuật toán thực hiện phép giao của mọi cặp WN-lists của các *k*-itemset có trong I_k, sau đó so sánh giá trị *ws* của (*k*+1)-itemset thu được với ngưỡng, và phụ thuộc vào kết quả so sánh để cập nhật tập *FWI* và L_{next} (Dòng 13-14).

Ví dụ 3. Chúng ta sẽ minh họa hoạt động của thuật toán NFWI với CSDL ví dụ ở trong Bảng 2. Sau khi gọi hàm **Construction_WN_Tree** ta thu được kết quả là cây WN như Hình 1 với tập I_1 = {*B, E, A, D, C*}, *minws* = 0.5, *sumtw* = 2.4.

Từ cây WN và I_1 ta sinh ra được WN-list cho các 1-FWI như sau:

WL(*C*) = {(5, 0, 0.5), (6, 2, 0.4), (8, 4, 0.4)}

WL(*D*) = {(4, 1, 1), (7, 5, 0.4)}

WL(*A*) = {(3, 3, 1.4), (9, 7, 0.4)}

WL(*E*) = {(2, 6, 2)}

WL(*B*) = {(1, 8, 2.4)}

Cập nhật các 1-FWI vào tập *FWI*:

FWI = {***B***, ***E***, ***A***, ***D***, ***C***}

Duyệt theo các nhánh của cây liệt kê (Hình 2) từ trái qua phải, ta lần lượt khai thác FWI theo tiến trình như sau:

Khởi đầu với L_1 = {*B, E, A, D, C*}, $Pre_1 = \emptyset$, $S_1 = \emptyset$

- **Đi theo nhánh C**: Pre_2 = {*C*}

  WL(*CD*) = {(4, 1, 0.5), (7, 5, 0.4)}

  WL(*CA*) = {(3, 3, 0.5), (3, 3, 0.4)} = {(3, 3, 0.9)}

  WL(*CE*) = {(2, 6, 0.5), (2, 6, 0.4), (2, 6, 0.4)} = {(2, 6, 1.3)}

  WL(*CB*) = {(1, 8, 0.5), (1, 8, 0.4), (1, 8, 0.4)} = {(1, 8, 1.3)}

  Do *ws*(*CD*) = (0.5+0.4)/2.4 = 0.9/2.4<*minws* = 0.5 nên *CD* không phải là FWI. Tương tự như vậy, *CA* cũng không phải là FWI.

  Do *ws*(*CE*) = *ws*(*CB*) = *ws*(*C*) = 1.3/2.4 > 0.5 nên cập nhật *FWI* và L_2 = {*E, B*}:

  FWI = {*B, E, A, D, C, CE, CB*}

  - Bởi vì L_2 = {*E, B*} nên gọi đệ quy đến hàm ***Find_FWI***(L_2 = {*E, B*}, Pre_2 = {*C*}).

    WL(*CEB*) = {(1, 8, 1.3)}

    Bởi vì *ws*(*CEB*) = 1.3/2.4 > 0.5 nên cập nhật *FWI*:

FWI = {*B, E, A, D, C, CE, CB, CEB*}
Thực hiện tương tự với các nhánh D, A và E, ta nhận được kết quả cuối cùng như sau:
FWI = {*B, E, A, D, C, CE, CB, CEB, DE, DB, DEB, AE, AB, AEB, EB*}

V. THỰC NGHIỆM VÀ ĐÁNH GIÁ

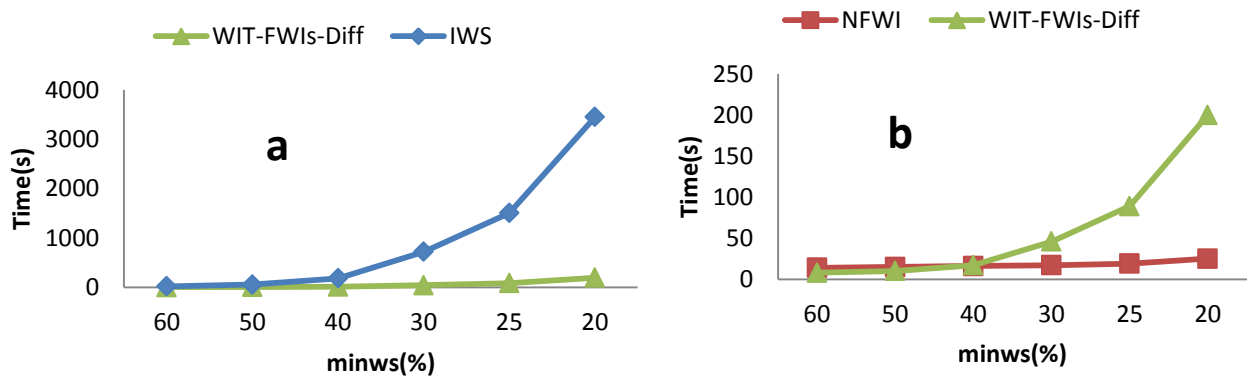
Tất cả các thử nghiệm trong phần tiếp theo đều được tiến hành trên cùng một hệ thống CPU Intel Core i5 2.5 GHz, bộ nhớ Ram 8GBs, chạy trên hệ điều hành Windows 7, sử dụng ngôn ngữ lập trình Visual C# 2012.

Chúng tôi thực nghiệm trên các CSDL Accidents, Connect, Retail và BMS-POS, được download từ http://fimi.cs.helsinki.fi/data/ và biến đổi bằng cách thêm bảng lưu trữ trọng số của các item (ngẫu nhiên trong khoảng 1 đến 10). Chúng tôi so sánh thuật toán NFWI với các thuật toán khai thác FWI mới nhất là WIT-FWIs-Diff [14] và IWS [15], trong đó WIT-FWIs-Diff được đánh giá hiệu quả trên các CSDL dày, còn IWS thì hiệu quả hơn trên các CSDL thưa.

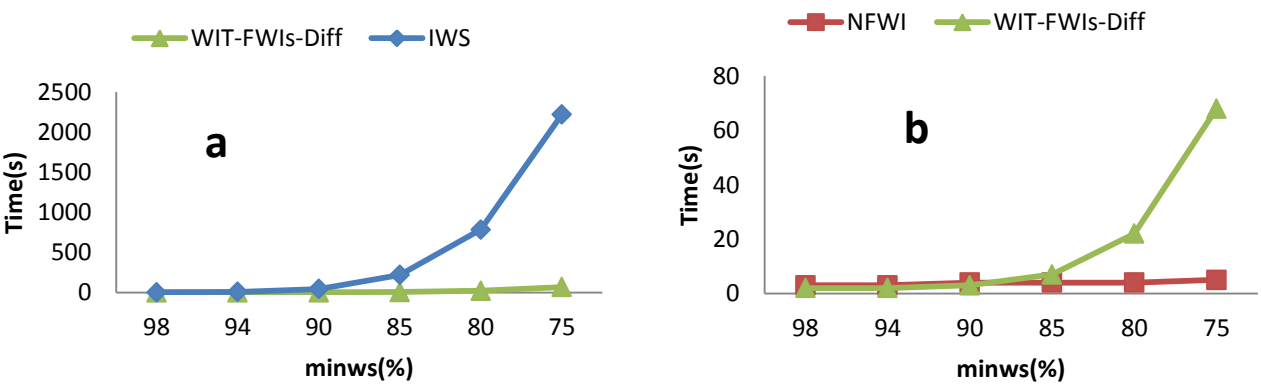
Bảng 3. Một số thông số về các CSDL dùng thử nghiệm

CSDL	Số lượng items	Số lượng giao dịch	Độ dài trung bình của giao dịch	Ghi chú
Accidents	468	340,183	33.8	Modified
Connect	130	67,557	43	Modified
Retail	16,470	88,162	10.3	Modified
BMS-POS	1,657	515,597	6.5	Modified

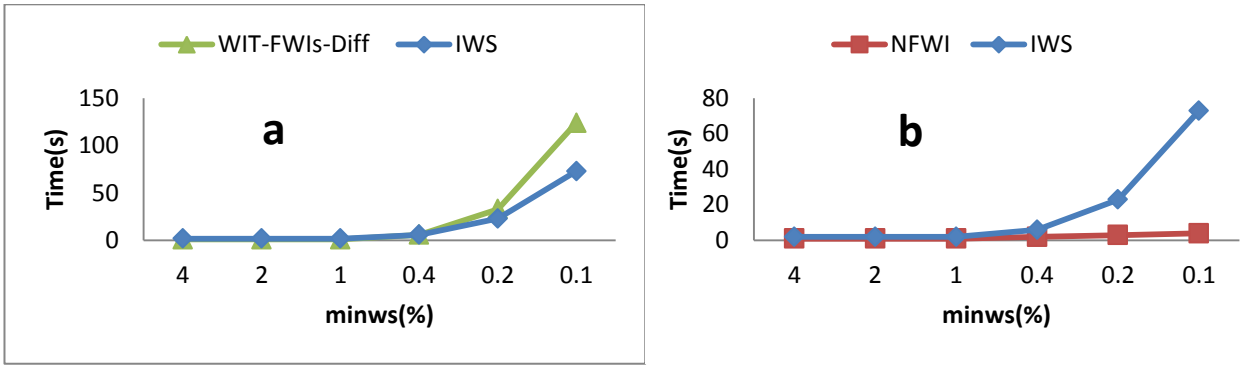
Do mức độ chênh lệch về thời gian chạy giữa thuật toán xếp cuối là quá lớn so với hai thuật toán còn lại, nên khi để chung trong một hình thì không thể hiện rõ được sự chênh lệch giữa hai thuật toán xếp thứ nhất và thứ hai. Vì vậy chúng tôi tách mỗi hình đồ thị so sánh về mặt thời gian chạy thành hình a và hình b để dễ theo dõi và so sánh. Trong đó, hình a thể hiện so sánh giữa thuật toán xếp thứ nhì với thuật toán xếp thứ ba trong 3 thuật toán NFWI, WIT-FWIs-Diff và IWS. Hình b thể hiện so sánh giữa thuật toán xếp thứ nhất với thuật toán xếp thứ hai.



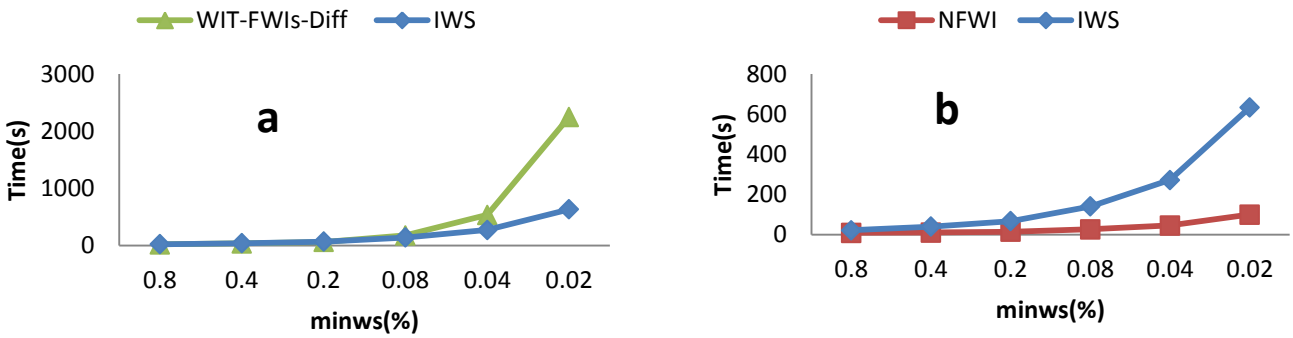
Hình 4. So sánh thời gian chạy trên CSDL Accidents



Hình 5. So sánh thời gian chạy trên CSDL Connect



Hình 6. So sánh thời gian chạy trên CSDL Retail



Hình 7. So sánh thời gian chạy trên CSDL BMS-POS

Hình 4a-7a cho thấy trên các CSDL dày như Accidents và Connect thì thuật toán WIT-FWIs-Diff hiệu quả hơn IWS, ngược lại trên các CSDL thưa như Retail và BMS-POS thì IWS hiệu quả hơn WIT-FWIs-Diff. Hình 4b-6b cho thấy thuật toán NFWI chạy nhanh hơn WIT-FWIs-Diff trên các CSDL dày, đồng thời cũng nhanh hơn IWS trên các CSDL thưa. Như vậy thực nghiệm cho thấy NFWI chạy nhanh hơn trên tất cả các loại CSDL (dày và thưa), đặc biệt khi ngưỡng càng nhỏ thì NFWI càng chiếm ưu thế so với WIT-FWIs-Diff và IWS.

Hình 4b minh họa trên CSDL dày Accidents: với ngưỡng *minws*= 50% thì thời gian chạy của NFWI là 15.1s chậm hơn so với thời gian chạy của WIT-FWIs-Diff là 10s, nhưng khi ngưỡng *minws* =40% thì thời gian của NFWI là 16.2s nhanh hơn so với thời gian của WIT-FWIs-Diff là 17s. Tiếp nữa khi ngưỡng *minws* = 30% thì thời gian của NFWI chỉ là 17.4s so với của WIT-FWIs-Diff là 45.5s. Trong bước này, thời gian của NFWI tăng 1.2s (tăng 7.41%) còn thời gian của WIT-FWIs-Diff tăng tới 28.5s (tăng 167.65%). Và với ngưỡng càng nhỏ thì mức độ chênh lệch càng tăng lên nhiều lần.

Hình 6b minh họa trên CSDL thưa Retails: với ngưỡng minws = 0.4% thời gian chạy của NFWI là 1.9s, còn của IWS là 5.5s. Khi ngưỡng minws = 0.2% thì thời gian của NFWI là 2.8s (tăng 47.37%) còn của IWS là 22.6s (tăng 310.91%). Tiếp theo khi ngưỡng minws = 0.1% thì thời gian của NFWI là 3.6s (tăng 28.57%) còn của IWS là 124.4s (tăng 450.44%). Và mức độ chênh lệch đó tiếp tục tăng lên khi ngưỡng càng nhỏ.

Phần lớn chi phí trong NFWI đều dùng để xây dựng cây WN ban đầu, còn chi phí khi tính giao WN-list phía sau là nhỏ, đó chính là lí do giải thích cho việc NFWI có thể chạy chậm hơn ở các ngưỡng lớn, nhưng khi ngưỡng càng nhỏ thì NFWI càng tỏ ra hiệu quả hơn so với WIT-FWIs-Diff và IWS.

VI. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Trong bài báo này chúng tôi đã phát triển cấu trúc WN-list, một mở rộng của cấu trúc N-lists [8], để biểu diễn CSDL có trọng số. Từ đó chúng tôi xây dựng thuật toán NFWI để khai thác nhanh tập phổ biến có trọng số. Sự hiệu quả của thuật toán NFWI thu được là nhờ các ưu điểm của cấu trúc WN-list thừa kế từ cấu trúc N-list. Các ưu điểm đó là: khả năng nén dữ liệu cao khiến kích thước WN-list nhỏ, dễ dàng tính độ phổ biến trọng số thông qua quét WN-list và thuật toán giao hai WN-list chỉ có độ phức tạp tuyến tính. Các thực nghiệm trên nhiều loại CSDL đã cho thấy NFWI hoạt động hiệu quả hơn các thuật toán khai thác FWI hiện có.

Trong tương lai, chúng tôi sẽ tập trung vào việc áp dụng cấu trúc N-list trong một số bài toán khai thác dữ liệu khác như khai thác tập phổ biến tiện ích cao hay khai thác tập phổ biến trên CSDL tăng trưởng.

TÀI LIỆU THAM KHẢO

[1] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *SIGMOD '93 Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, 1993, pp. 207-216.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and I. A. Verkamo, "Fast discovery of association rules," in *Advances in knowledge discovery and data mining*. American Association for Artificial Intelligence Menlo Park, 1996, pp. 307-328.

[3] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *SIGMOD '00 Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 1-12.

[4] G. Grahne and J. Zhu, "Fast algorithms for frequent itemset mining using FP-trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 10, pp. 1347-1362, 2005.

[5] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 3, pp. 372-390, 2000.

[6] M. J. Zaki and K. Gouda, "Fast vertical mining using diffsets," in *KDD '03 Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 326-335.

[7] B. Vo, L. Hong, and B. Le, "DBV-Miner: A Dynamic Bit-Vector approach for fast mining frequent closed itemsets," *Expert Systems with Applications*, vol. 39, no. 8, pp. 7196-7206, 2012.

[8] Z. Deng, Z. Wang, and J. Jiang, "A new algorithm for fast mining frequent itemsets using N-lists," *Science China Information Sciences*, vol. 55, no. 9, pp. 2008-2030, 2012.

[9] B. Vo, T. Le, F. Coenen, and T.-P. Hong, "Mining frequent itemsets using the N-list and subsume concepts," *International Journal of Machine Learning and Cybernetics*, pp. 1-13, 2014.

[10] Z. Deng and S. Lv, "PrePost+: An efficient N-list-based algorithm for mining frequent itemsets via Children-Parent Equivalence pruning," *Expert Systems with Applications*, vol. 42, no. 13, pp. 5424-5432, 2015.

[11] G. D. Ramkumar, S. Ranka, and S. Tsur, "Weighted Association Rules: Model and Algorithm," in *Proc. Fourth ACM Int'l Conf. Knowledge Discovery and Data Mining*, 1998, pp. 01-13

[12] C. H. Cai, A. W. C. Fu, C. H. Cheng, and W. W. Kwong, "Mining association rules with weighted items," in *Database Engineering and Applications Symposium, 1998. Proceedings. IDEAS'98. International*, 1998, pp. 68-77.

[13] F. Tao, F. Murtagh, and M. Farid, "Weighted Association Rule Mining Using Weighted Support and Significance Framework," in *KDD '03 Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 661-666.

[14] B. Vo, F. Coenen, and B. Le, "A new method for mining Frequent Weighted Itemsets based on WIT-trees," *Expert Systems with Applications*, vol. 40, no. 4, pp. 1256-1264, 2013.

[15] H. D. Nguyen, B. Vo, M. H. T. Nguyen, and T.-P. Hong, "An Improved Algorithm for Mining Frequent Weighted Itemsets," in *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*, 2015, pp. 2579-2584.

[16] R. Rymon, "Search through systematic set enumeration," in *In Proc. Int'l Conf. principles of knowledge representation and reasoning*, 1992, pp. 539-550.

A WEIGHTED N-LIST-BASED METHOD FOR MINING FREQUENT WEIGHTED ITEMSETS

Bui Danh Huong, Vo Dinh Bay, Nguyen Duy Ham

***ABSTRACT*—** *Mining frequent itemsets plays an important role in data mining. There have been many different methods proposed to solve this problem. In particular, the N-list structure proposed by Deng et al. (Deng, Wang, & Jiang, 2012) that use a hybrid approach between the FP-tree and enumerate-tree achieved encouraging efficiency. However, this method operates only on binary databases. In this paper, we proposed the WN-list (Weighted N-list) structure, an extension of the N-list structure, to solve the problem of mining frequent weighted itemsets from weighted database. First, some theorems are developed to calculate the weight support of an itemset, and then, an algorithm is built based on these theorems for fast mining frequent weighted itemsets. Experimental results on a variety of databases (sparse and dense) show that the proposed method outperforms existing methods, especially with the small threshold.*