Khai thác *k* mẫu tuần tự tối đại sử dụng cây dữ liệu chiếu tiền tố

Mining Top-*k* Maximal Sequential Patterns using Prefix-Projected Database Tree

Lê Hoài Bắc, Nguyễn Thị Quyên

Abstract: *This paper propose a method called TMSP to perform squential patten mining. Because maximal patterns compact representations of frequent patterns, so they are used for mining in TMSP. The main idea of TMSP is mining top-k frequent maximal equential patterns of length no less than the minimum length of each pattern (min_l) and no greater than the maximum length of each pattern (max_l) with k is the desired number of maximal sequential patterns to be mined. The proposed method helps user do not need turning specification of a minimum support threshold to perform the mining which is a disadvantage of previous studies. Experimental results on real datasets show that TMSP serves as an efficient solution for mining sequential patterns. The reults also demonstrate that TMSP is better than the maximal sequential pattern mining algorithm (MAXSP) in term memory efficient and easier for users to find the number of required patterns without adjusting minsup.*

Keywords: *Sequential, Sequential pattern mining, maximal sequential patterns, Top-k maximal sequential patterns, prefix-projected databases.*

I. GIỚI THIỆU

Khai thác mẫu tuần tự là một nhiệm vụ quan trọng của khai thác dữ liệu đã và đang được nghiên cứu rộng rãi [1-3]. Cho một tập các chuỗi, trong đó mỗi chuỗi bao gồm một danh sách các tập phổ biến và một ngưỡng hỗ trợ tối thiểu do người dùng chỉ định (*minsup*), khai thác mẫu tuần tự là tìm ra tất cả các mẫu phổ biến có độ hỗ trợ không thấp hơn *minsup*. Trong đó, thuật toán PrefixSpan [1] tổ chức dữ liệu theo chiều ngang và thực hiện phép chiếu trên CSDL để giảm chi phí lưu trữ dữ liệu, xuất phát từ tập mẫu tuần tự độ dài là 1, PrefixSpan tạo ra CSDL được chiếu với mỗi mẫu đó. Trong CSDL chiếu, mỗi chuỗi dữ liệu chỉ giữ lại phần hậu tố đối với tiền tố đã chiếu. Mẫu được phát triển bằng những item phổ biến tìm được trong CSDL được chiếu. Quá trình này được lặp lại cho đến khi CSDL chiếu không còn item phổ biến nào. Thuật toán SPADE [2] tổ chức dữ liệu theo chiều dọc, ứng với mỗi item sẽ lưu danh sách định danh của các chuỗi dữ liệu và định danh của các itemset có chứa item đó. Độ hỗ trợ của item được tính trực tiếp từ danh sách các định danh. Mặt khác, SPADE còn dựa trên lý thuyết dàn để chia nhỏ không gian tìm kiếm và thao tác kết đơn giản để tạo ra tập ứng viên. Thuật toán này gom nhóm các mẫu tuần tự dựa theo tiền tố thành các lớp tương đương. Thuật toán SPAM [3] cũng tổ chức dữ liệu theo chiều dọc như thuật toán SPADE, trong đó các mẫu ứng viên được biểu diễn dưới dạng bảng bit dọc, mỗi bit ứng với một itemset của một chuỗi trong CSDL. Nếu item có mặt trong itemset j thì bít tương ứng itemset j được đánh dấu là 1, ngược lại là 0. Độ hỗ trợ cũa mẫu được xác định dựa trên bảng bit.

Các thuật toán trên tạo ra các mẫu phổ biến mà các mẫu đó có thể có cùng độ hỗ trợ hoặc là mẫu con của mẫu phổ biến khác và khi chọn *minsup* quá cao sẽ tạo ra ít các mẫu bỏ qua các thông tin có giá trị, còn ngược lại thì quá nhiều mẫu dẫn đến thuật toán thực thi chậm. Để chọn một giá trị *minsup* hợp lý đòi hỏi phải hiểu rõ về dữ liệu. Để khắc phục vấn đề này, P. Fournier-Viger và cộng sự đã đề xuất thuật toán khai thác *k* mẫu tuần tự (TKS) [4], thuật toán TKS khai thác các mẫu tuần tự dựa vào thuật toán SPAM xuất phát từ giá trị *minsup* = 0 bước tiếp theo tìm các mẫu thỏa giá trị

minsup khi số lượng mẫu thu được lớn hơn k mẫu sẽ xóa các mẫu có độ hỗ trợ bằng *minsup*, xóa đến khi bằng k mẫu và tăng giá trị minsup bằng độ hỗ trợ thấp nhất trong tập k mẫu thu được, quá trình này được lặp lại cho đến khi CSDL chiếu không còn mẫu phổ biến nào. Thuật toán TKS khắc phục được vấn đề tinh chỉnh giá trị *minsup* nhưng trong số mẫu tuần tự thu được còn tồn tại các mẫu tuần tự có cùng độ hỗ trợ. Ngoài ra P. Tzvetkov và cộng sự cũng đã đề xuất thuật toán khai thác *k* mẫu tuần tự đóng TSP [5], sử dụng phương pháp chiếu dựa vào thuật toán PrefixSpan thực hiện tương tự như thuật toán TKS. Kết quả thuật toán TSP thu được không còn tồn tại các mẫu tuần tự có cùng độ hỗ trợ nhưng thuật toán TSP vẫn còn tồn tại các mẫu tuần tự con trong các mẫu tuần tự thu được. Do đó, Philippe và cộng sự đã đề xuất thuật toán khai thác các mẫu tuần tự tối đại (MaxSP) [7] khai thác các mẫu tuần tự dựa trên thuật toán PrefixSpan, kết quả nhận được không tồn tại mẫu tuần tự con nhưng rất khó cho người sử dụng phải điều chỉnh giá trị *minsup* hợp lý và dung lượng bộ nhớ sử dụng lớn. Chính vì vậy, trong bài báo này sẻ trình bày thuật toán khai thác *k* mẫu tuần tự tối đại (TMSP) để tối ưu hóa về bộ nhớ sử dụng và giúp người sử dụng dễ dàng tìm được số lượng mẫu tuần tự như mong muốn.

II. MỘT SỐ ĐỊNH NGHĨA CƠ BẢN

Chuỗi $s = < t_1, t_2, \dots t_m > (t_i \subseteq I)$ là một danh sách có thứ tự [7]. Cơ sở dữ liệu chuỗi *D* là một tập các chuỗi $S = \{s_1, s_2, \dots s_s\}$ và tập các item $I = \{i_1, i_2, \dots, i_k\}$ xảy ra trong chuỗi đó. Item là giá trị tượng trưng. Itemset $I = \{i_1, i_2, \dots, i_k\}$ là tập các item riêng biệt không có thứ tự.

Ví dụ 1: Xét CSDL chuỗi như sau:

Bảng 1. Cơ sở dữ liệu chuỗi

SID	Sequences
1	<(*a*)(*abc*)(*ac*)(*d*)(*cf*)>
2	<(*ad*)(*c*)(*bc*)(*ae*)>
3	<(*ef*)(*ab*)(*df*)(*c*)(*b*)>
4	<(*e*)(*g*)(*af*)(*c*)(*b*)(*c*)>

Trong Bảng 1, Chuỗi s_1 có 5 itemset xảy ra theo thứ tự (*a*), (*abc*), (*ac*), (*d*) và sau cùng là (*cf*).

Chiều dài của *s*, *l*(*s*) là tổng số các item trong s còn được gọi là *l-sequence*.

Ví dụ 2: Chuỗi <(*ad*)(*c*)> là một *3-sequence* có kích thước là 2.

Chuỗi $\alpha = < a_1, a_2, \dots, a_m >$ là một chuỗi con của chuỗi khác $\beta = < b_1, b_2, \dots, b_n >$, kí hiệu là $\alpha \subseteq \beta$, nếu và chỉ nếu $\exists i_1, i_2, \dots i_m$, sao cho $1 \le i_1 < i_2 < \dots < i_m \le n$ và $a_1 \subseteq b_{i1}, a_2 \subseteq b_{i2}, \dots, v$à $a_m \subseteq b_{im}$. Người ta gọi β là chuỗi cha của α.

Ví dụ 3: Chuỗi <(*a*)(*c*)> là chuỗi con của <(*ad*)(*c*)(*bc*)(*ae*)>, nhưng <(*a*)(*c*)> không phải là chuỗi con của chuỗi <(*ac*)> và ngược lại.

Định nghĩa 1. (*Tiền tố - Prefix*): Giả sử các items trong một itemset được sắp xếp theo thứ tự từ điển. Cho hai chuỗi $\alpha = < e_1, e_2, \dots, e_n > v$à $\beta = < e'_1, e'_2, \dots, e'_m >$, $(m \le n)$. Chuỗi β là tiền tố của α nếu và chỉ nếu [1]:

a. $e'_i = e_i$ với mọi $i \le m - 1$.
b. $e'_m \subseteq e_m$.
c. Các item trong $(e_m - e'_m)$ theo thứ tự là những item đứng sau e'_m trong e_m.

Ví dụ 4: Xét CSDL như Bảng 1, Các tiền tố của chuỗi α = <(*a*) (*abc*) (*ac*) (*d*) (*ef*)> là <(*a*)>, <(*a*) (*a*)>, <(*a*) (*ab*)>, <(*a*) (*abc*)>… nhưng các chuỗi <(*ab*)>, <(*a*) (*bc*)> không phải là tiền tố của α.

Định nghĩa 2. (*Hậu tố - Postfix*): Cho chuỗi $\alpha = < e_1, e_2, \dots, e_n >$ có chuỗi $\beta = < e'_1, e'_2, \dots, e'_m >$ là tiền tố của α. Chuỗi hậu tố của α có dạng: $\gamma = < e''_m, e_{m+1}, \dots, e_n >$. Trong đó, $e''_m = e_m - e'_m, \gamma = \alpha - \beta$ [1].

Nếu chuỗi $\beta \not\subset \alpha$ thì: $\gamma = \alpha - \beta = \emptyset$.

Ví dụ 5: Xét CSDL như Bảng 1. Chuỗi α = <(*a*)(*abc*)(*ac*)(*d*)(*ef*)> có các tiền tố:

β_1= <(*a*)> ⟹ Hậu tố γ_1 = <(*abc*)(*ac*)(*d*)(*ef*)>

β_2= <(*a*)(*a*)> ⟹ Hậu tố γ_2 = <(_*bc*)(*ac*)(*d*)(*ef*)>

β_3= <(*a*)(*ab*)> ⟹ Hậu tố γ_3 = <(_*c*)(*ac*)(*d*)(*ef*)>

Định nghĩa 3. (*Phép chiếu cơ sở dữ liệu theo tiền tố*):

Cho α là mẫu trong cơ sở dữ liệu *D*.

Cơ sở dữ liệu được chiếu theo α, kí hiệu: $D|_\alpha$, là tập hợp các hậu tố của các chuỗi dữ liệu trong *D* có liên quan đến tiền tố α [1].

Ví dụ 6: Xét CSDL như Bảng 1, tiền tố α = <(*f*)>. Khi đó cơ sở dữ liệu được chiếu theo α. $D|_\alpha$= {<(*ab*)(*df*)(*c*)(*b*)>, <(*c*)(*b*)(*c*)>}

Định nghĩa 4. (*Độ hỗ trợ - Support*): Xét CSDL chuỗi *D*, mỗi chuỗi có một chỉ số định danh duy nhất. Độ hỗ trợ tuyệt đối mẫu α là tổng số chuỗi trong *D* có chứa *p*, ký hiệu $sup_D(p) = |\{s|s \in D \text{ và } \alpha \subseteq s\}|$. Độ hỗ trợ tương đối của *p* là tỉ lệ phần trăm chuỗi trong *D* chứa *p*. Ở đây, mức hỗ trợ tuyệt đối hoặc tương đối sẽ được sử dụng chuyển đổi qua lại, kí hiệu là *sup*(*p*).

Ví dụ 7: Xét CSDL như Bảng 1, Mẫu *p* = <(*a*)(*c*)> xuất hiện trong chuỗi s_1, s_2, s_3, s_4. Vậy độ hỗ trợ của mẫu *p* là 4.

Định nghĩa 5. (*Mẫu tuần tự*): Cho trước ngưỡng hỗ trợ tối thiểu (*minsup*) xác định bởi người dùng. Một mẫu α được coi là phổ biến nếu độ hỗ trợ của nó lớn hơn hoặc bằng *minsup*: *sup(α)* ≥ *minsup*, khi đó α được gọi là mẫu tuần tự [8].

Ví dụ 8: Xét CSDL như Bảng 1, có tập các item phân biệt là {*a, b, c, d, e, f, g*} và *minsup* = 2. Xét chuỗi s_1 = <(*a*)(*abc*)(*ac*)(*d*)(*cf*)> chuỗi s_1 có 5 itemset là: (*a*), (*abc*), (*ac*), (*d*), (*cf*) và có 9 lần xuất hiện của các item.

Vậy s_1 có kích thước là 5 và có độ dài là 9. Trong chuỗi s_1, item *a* xuất hiện ba lần nhưng nếu tính độ hỗ trợ thì độ hỗ trợ của item *a* chỉ được tính là 1 đối với chuỗi s_1 đó. Mẫu *p* = <(*a*)> xuất hiện trong chuỗi s_1, s_2, s_3, s_4. Vậy độ hỗ trợ của mẫu *p* là 4. Vì *sup*(*p*) > *minsup* nên *p* là mẫu tuần tự.

Định nghĩa 6. (*Mẫu tuần tự tối đại*): Mẫu tuần tự s_a là mẫu tuần tự tối đại nếu nó không tồn mẫu tuần tự s_b sao cho mẫu s_b là cha của mẫu s_a, $s_a \sqsubseteq s_b$ [8,9].

Ví dụ 9: Xét CSDL như Bảng 1, mẫu <{*f*},{*b*},{*c*}> có *sup* = 2 là mẫu tuần tự tối đại nhưng mẫu <{f}> có *sup* = 3 không là mẫu tuần tự tối đại vì mẫu <{*f*}> là con của mẫu <{*f*},{*b*},{*c*}>.

Định nghĩa 7. (*Khai thác k mẫu tuần tự tối đại*): Để tìm ra tập *F* gồm có *k* mẫu tuần tự tối đại mà mỗi mẫu tối đại $s_a \in F$, $l(s_a) \geq \min_l$, $l(s_a) \leq \max_l$ và không tồn tại mẫu tuần tự tối đại $s_b \notin F$ mà $l(s_b) \geq \min_l$, $l(s_b) \leq \max_l$ và $\sup(s_b) > \sup(s_a)$.

Ví dụ 10: Xét CSDL như bảng 1, với *k* = 2, *min_l* = 1, *max_l* = 3. Thuật toán tìm được 2 mẫu như sau: <(*a*)(*c*)(*b*)>: 3, <(*a*)(*c*)(*c*)>: 3. Mặc dù thuật toán tìm được hơn 2 mẫu với độ hỗ trợ bằng 3: <(*c*)(*c*)>: 3, <(*c*)(*b*)>: 3, những mẫu này không nằm trong kết quả bởi vì chúng không là mẫu tuần tự tối đại và chúng là con của mẫu kết quả.

Định nghĩa 8. Chuỗi *s* cho trước, tập chuỗi *IDs* của tất cả các chuỗi trong CSDL D chứa *s* gọi là danh sách chuỗi *ID*, ký hiệu: *SIDList*(*s*). Tổng của *SIDList*(*s)* gọi là tổng chuỗi *ID*, ký hiệu là *SIDSum*(*s*) [5].

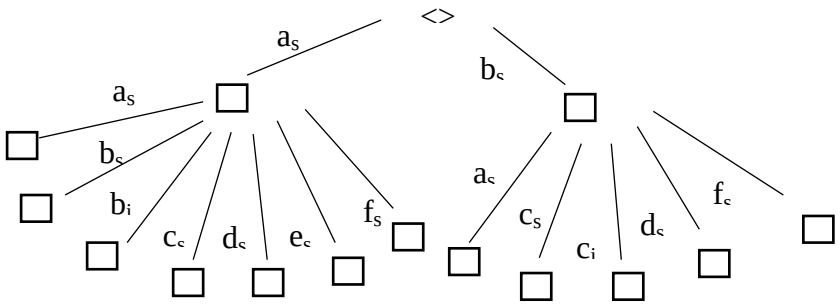
Ví dụ 11: Xét CSDL như bảng 1 thì danh sách chuỗi *ID* của <(*a*)(*b*)> là [1, 2, 3, 4] và tổng chuỗi *ID* của <(*a*)(*b*)> là 10.

III. PHƯƠNG PHÁP PHÁT TRIỂN

III.1. Cấu trúc cây tiền tố

Cây tiền tố là một cây có thứ tự, trong đó quan hệ giữa nút cha với nút con tương ứng là quan hệ giữa chuỗi con và chuỗi cha. Cây tiền tố được xây dựng như sau: Bắt đầu từ nút gốc của cây tại mức 0, nút gốc được gán nhãn là một chuỗi rỗng ⟨⟩. Tại mức *min_l* bất kỳ, mỗi nút được gán nhãn là một mẫu tuần tự độ dài *min_l*. Các nút ở mức (*min_l* + 1) kế tiếp được xây dựng đệ quy bằng cách mở rộng mẫu độ dài *min_l* ở mức trước đó. Mẫu mở rộng bằng cách thêm vào một item phổ biến đó là mở rộng theo chuỗi và mở rộng theo itemset. Quá trình mở rộng theo một thứ tự cho đến khi không còn item phổ biến.

Ví dụ 12: xét mẫu *p* = <{*a*}>, nếu thêm một item *b* vào mẫu *p* thì <{*a*}{*b*}> là mở rộng theo chuỗi và <{*ab*}> là mở rộng theo itemset. (Hình 1)



Hình 1. Cây tiền tố

III.2. Cấu trúc cây PDB

Cấu trúc cây PDB là một bộ nhớ mô tả cây tìm kiếm tiền tố và lưu trữ thông tin về CSDL chiếu đã được khai thác một phần trong suốt quá trình khai thác đa duyệt.

Khai thác các mẫu tuần tự dựa trên cây PDB như khai thác cây tìm kiếm tiền tố nhưng khai thác trên cây PDB làm giảm rất nhiều số lần duyệt trên CSDL chiếu bằng cách thực hiện chiếu các tiền tố theo thứ tự giảm dần theo độ hỗ trợ thay vì thực hiện tìm kiếm như cây tiền tố. Do đó, Cây PDB tại mỗi điểm của tiền tố có cùng độ dài sẽ được sắp xếp giảm dần theo độ hỗ trợ, khi giá trị *minsup* tăng những tiền tố có độ hỗ trợ thấp sẽ dẫn đến không phổ biến nên bộ nhớ sử dụng ít.

Cây PDB có kích thước nhỏ hơn nhiều so với cây tìm kiếm tiền tố trong suốt quá trình khai thác dữ liệu. Độ sâu lớn nhất của cây PDB luôn luôn nhỏ hơn *min_l*. Mặt khác, giảm bộ nhớ để lưu trữ cây PDB, chúng ta sử dụng CSDL chiếu giả tại tất các nút cây PDB. Chúng ta chỉ lưu trữ danh sách các điểm của chuỗi hiện tại trong CSDL chuỗi ban đầu.

Ví dụ 13: Xét CSDL như Bảng 1. Từ hình 2 ta thấy cây PDB chỉ lưu trữ danh sách các điểm của chuỗi, Khi chiếu theo tiền tố a_s có độ hỗ trợ là 4 thì ta tìm được các item a_s, b_s, b_i, c_s, c_i, d_s, d_i, e_s, e_i, f_s, f_i nhưng tại mỗi điểm sau khi được sắp xếp và lưu trữ giảm dần theo độ hỗ trợ như sau: b_s: 4, c_s: 4, a_s: 2, d_s: 2, f_s: 2, b_i: 2, e_s: 1, f_i: 1, d_i: 1, c_i: 1, e_i: 1. Chính vì vậy, các item có độ hỗ lớn sẽ thực hiện phép chiếu trước và giá trị *minsup* tăng nhanh dẫn đến các item có độ hỗ trợ thấp không phổ biến. Trong đó s mỡ rộng theo chuỗi, i mỡ rộng theo itemset. Quá trình lặp lại cho đến khi không còn item phổ biến.

IV. THUẬT TOÁN TMSP

IV.1. Thuật toán

Thuật toán khai thác các mẫu tuần tự tối đại (MaxSP) sử dụng thuật toán PrefixSpan dựa trên cây tiền tố để khai thác các mẫu tuần tự. Đối với thuật toán khai thác *k* mẫu tuần tự tối đại (TMSP) với số mẫu tuần tự *k* biết trước cũng sử dụng thuật toán PrefixSpan để khai thác các mẫu tuần tự nhưng dựa trên cấu trúc cây PDB được mô tả chi tiết như Hình 3.

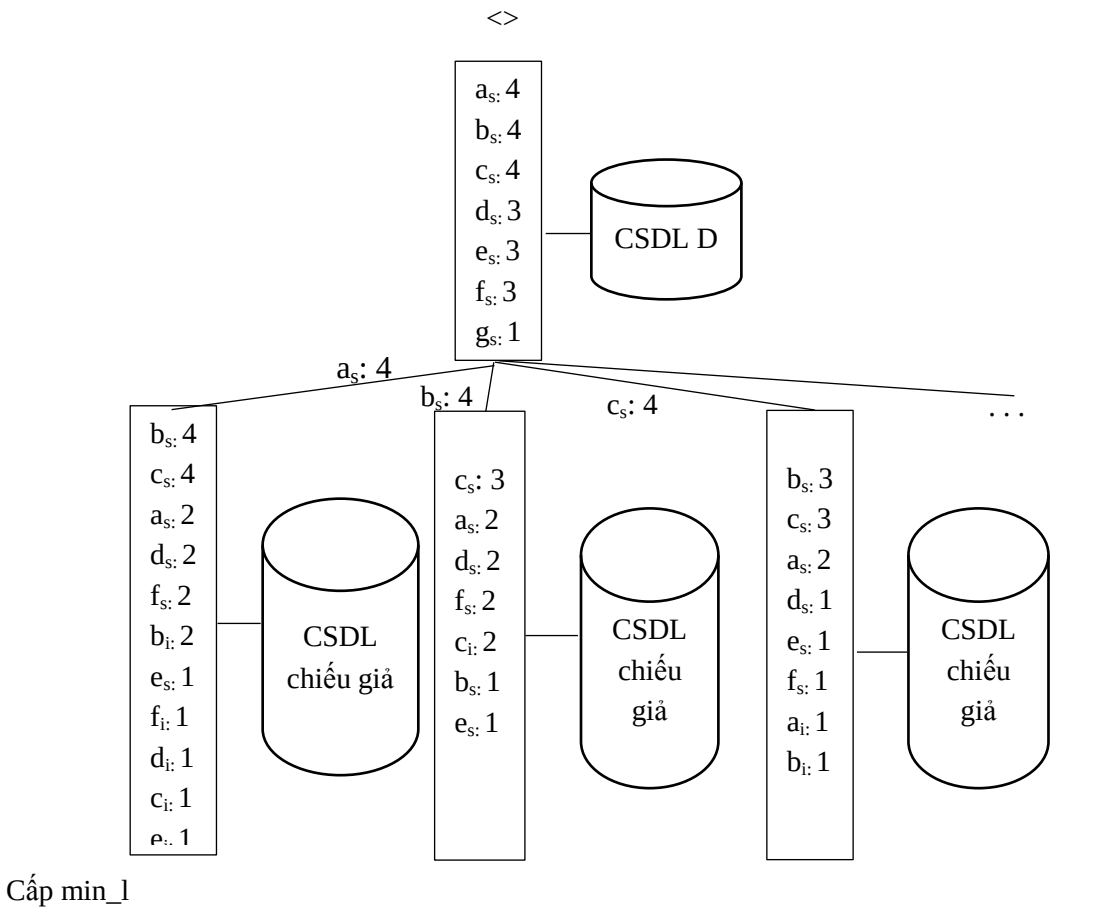
Mô tả thuật toán TMSP thực hiện các bước như sau:

- Bước 1: Tính hằng số θ được tính theo công thức $\theta = (k * \min_l)/N_{Items}$ (dòng 1).

- Bước 2: Tìm tập tất cả các item phổ biến chèn vào histogram có độ dài là 1 (dòng 2).

- Bước 3: Sắp xếp giảm dần các item có độ dài bằng 1 trong histogram (dòng 3).

- Bước 4: Ứng với mỗi α phổ biến gọi thủ tục TopSequencesTraversal (dòng 4).



Hình 2. Cây PDB

Đầu vào: CSDL *D, k, min_l, max_l;*
Đầu ra: Tập *F* gồm có *k* mẫu tuần tự tối đại
Phương pháp thực hiện:

1. $\theta = (k * \min_l)/N_{Items}$; $minsup = 1$;
2. Duyệt CSDL *D* một lần, tìm mỗi item phổ biến α sao cho:
   s có thể mở rộng đến *s* <> α;
   Chèn α vào histogram H[1];
3. Sắp xếp giảm dần các item trong H[1] dựa trên độ hộ trợ của chúng;
4. Với mỗi α, *support*(α) $\geq$ *minsup*
   Gọi TopsequencesTraversal(*s*<> α, $D_{s<>\alpha}$, *min_l*);

TopsequencesTraversal(*s* <> α, $D_{s<>\alpha}$, *min_l*)

5. Nếu *support*(*s*) < *minsup* thì kết thúc;
6. Nếu $l(s) = min_l$ thì
7. Savepattern(*s*);
8. Gọi PrefixSpanWSR(*s, minsup*, D_s, *F*);
9. Kết thúc;
10. Nếu *l*(*s*) > *max_l* thì kết thúc;
11. Duyệt CSDL D_s một lần, tìm mỗi item phổ biến α sao cho:
    s có thể mở rộng đến *s* <> α;
    Nếu *l*(α) $\geq$ *max_l* thì ngưng mở rộng;
    Chèn α vào histogram H[*l*(*s*) + 1];
12. Sắp xếp giảm dần các item trong H[*l*(*s*) + 1] dựa trên độ hộ trợ của chúng;
13. *Next_level_top_support* ← GetLevelTopSupportFromHistogam(θ, H[*l*(*s*) + 1])
14. Với mỗi α, *support*(α) $\geq$ *next_level_top_support*
15. Gọi TopSequencesTraversal(*s* <> α, $D_{s<>\alpha}$, *min_l*);
16. Kết thúc;

PrefixSpanWSR(*s, minsup*, D_s, *F*)

17. Nếu *l*(*s*) = *max_l* thì kết thúc;
18. Sắp xếp giảm dần các item trong H[*l*(*s*) + 1] dựa trên độ hộ trợ của chúng;
19. Duyệt CSDL D_s một lần, tìm mỗi item phổ biến

α sao cho:
20. s có thể mở rộng đến $s <> \alpha$;
21. Chèn α vào histogram H[$l(s)$ + 1];
22. Savepattern(α);
23. Nếu $l(\alpha) < max_l$ thì
24. PrefixSpanWSR($s <> \alpha$, $minsup$, D_s, F);

GetLevelTopSupportFromHistogam(θ, H[$l(s)$ + 1])

25. $Index = \theta * |histogram|$;
26. Nếu $index \geq histogram$ thì $index = |histogram| - 1$;
27. Kết quả trả về là độ hỗ trợ tại vị trí $index$;

SavePattern(s)

28. Nếu $l(s) < min_l$ hoặc $support(s) < minsup$ thì kết thúc;
29. Nếu MaximalPatternVerification(s) = add_and_raise thì
30. Nếu $|F| \geq k$ thì
31. Nếu $support(s) > minsup$ hoặc $l(s) > l_min(r) \in F$ thì
32. Trong khi $|F| \geq k$ và $\exists\ r \in F \mid l(r) < l(s)$
33. Xóa các mẫu có độ dài ngắn nhất từ F;
34. Trong khi $|F| \geq k$ và $\exists\ r \in F \mid support(r) = minsup$
35. Xóa các mẫu có độ hỗ trợ bằng minsup từ F;
36. $F := F \cup \{s\}$;
37. Cập nhật lại minsup bằng độ hỗ trợ nhỏ nhất trong tập F;
38. Ngược lại $F := F \cup \{s\}$;
39. Ngược lại nếu MaximalPatternVerification(s) = add_but_no_raise thì
40. Nếu $|F| = 0$ hoặc $l(s) \geq l_min(r)$ thì
41. $F := F \cup \{s\}$;

MaximalPatternVerification(s)

42. Nếu tồn tại item u, sao cho $support(s <>_i u) = support(s)$ hoặc $support(s <>_s u) = support(s)$ thì trả về no_add;
43. Nếu $SIDSum(s)$ không nằm trong $SIDSum_Hash$ thì trả về add_and_raise;
44. Với mọi s'
45. Nếu $s \subset s'$ thì trả về no_add;
46. Nếu $s' \subset s$ thì
47. Thay thế s' với s;
48. Trả về no_add;
49. Trả về add_and_raise;

Hình 3. Mô tả thuật toán TMSP

Mô tả chi tiết các thủ tục thực hiện trong thuật toán TMSP như sau:

Thủ tục TopsequencesTraversal được miêu tả như sau:

- Nếu độ hỗ trợ của s nhỏ hơn *minsup* thì kết thúc (dòng 5).
- Nếu chiều dài của s bằng min_l thì lưu s vào tập F và gọi thủ tục PrefixSpanWSR (dòng 6 - 9).
- Nếu chiều dài của s không bằng min_l mà lớn hơn max_l thì kết thúc ngược lại thì duyệt CSDL D_s và tìm mỗi item phổ biến sao cho s có thể mở rộng $s <> \alpha$ và chỉ mỡ rộng khi chiều dài mẫu nhỏ hơn max_l và chèn α vào histogram có độ dài bằng $l+1$, sau đó sắp xếp các item trong histogram giảm dần theo độ hỗ trợ để tìm *support* dựa vào thủ tục GetLevelTopSupportFromHistogam. Cuối cùng nếu độ hỗ trợ của α lớn hơn hoặc bằng *support* thì gọi đệ qui TopSequencesTraversal đến khi nào bằng min_l (dòng 10 - 16).

Thủ tục SavePattern trả về một trong ba kết quả sau:

+ add_but_no_raise: Thêm vào tập F nhưng không tăng giá trị *minsup*.

+ add_and_raise: Thêm vào tập F và tăng giá trị *minsup*.

+ no_add: Không thêm vào tập F.

Thủ tục SavePattern được mô tả cụ thể như sau: Nếu chiều dài mẫu tuần tự s nhỏ hơn min_l hoặc độ hỗ trợ mẫu s nhỏ hơn *minsup* thì kết thúc (dòng 28). Nếu thủ tục ClosePatternVerification(s) cho kết quả add_and_raise thì kiểm tra nếu số lượng mẫu trong tập F lớn hơn hoặc bằng k và độ hỗ trợ mẫu s lớn hơn *minsup* hoặc độ dài mẫu s lớn hơn mẫu có độ dài thấp nhất trong tập F thì sẽ xóa các mẫu trong tập F có độ dài thấp hơn mẫu s cho đến khi số lượng mẫu trong tập F bằng k (dòng 29-33), trong trường hợp tất cả các mẫu trong tập F đều bằng nhau thì xóa các mẫu trong tập F có độ hỗ trợ bằng *minsup* cho đến khi số lượng mẫu trong tập F bằng k (dòng 34 - 35), sau đó thêm s vào và cập nhật lại *minsup* bằng độ hỗ trợ thấp nhất trong tập F (dòng 36 - 37). Nếu số lượng tập F nhỏ

hơn *k* thì thêm *s* vào (dòng 38). Trong trường hợp thủ tục ClosePatternVerification(*s*) cho kết quả add_but_no_raise thì kiểm tra nếu chưa có mẫu hoặc độ dài mẫu *s* lớn hơn hoặc bằng độ dài mẫu thấp nhất trong tập *F* thì thêm mẫu *s* vào tập *F* (dòng 39 - 41).

Thủ tục MaximalPatternVerification được mô tả các bước như sau:

- Nếu tồn tại một item mở rộng *u* có cùng độ hỗ trợ với *s* thì không thêm *s* vào tập *F* (dòng 42).
- Nếu *SIDSum*(*s*) (tổng chuỗi ID) không nằm trong bảng *Hash* thì thêm *s* vào tập *F* và tăng độ hỗ trợ (dòng 43).
- Với mỗi s' thì thủ tục sẽ thực hiện 1 trong 2 điều kiện như sau (dòng 44):
  - Nếu *s* là cha của s' thì không thêm vào tập *F* (dòng 45).
  - Nếu s' là cha của *s* thì thay thế s' với *s* nhưng không thêm vào tập *F* (dòng 46 - 48).
- Trường hợp còn lại thêm vào tập *F* và tăng độ hỗ trợ (dòng 49).

IV.2. Ví dụ minh họa

Ví dụ 14: Cơ sở dữ liệu như Bảng 1, minh họa thuật toán TMSP với *k* = 2, *min_l* = 1 và *max_l* = 3 như sau:

Minsup = 1; θ = 0.4;

Duyệt CSDL *D* tìm được các item phổ biến, mỗi item phổ biến tạo nên một chuỗi có độ dài là 1. Vậy có 7 mẫu phổ biến chèn vào histogram có độ dài là 1 đã được sắp xếp giảm dần theo độ hỗ trợ như sau: <(*a*)>: 4, <(*b*)>: 4, <(*c*)>: 4, <(*d*)>: 3, <(*e*)>: 3, <(*f*)>: 3, <(*g*)>: 1.

Ứng với mỗi mẫu phổ biến thực hiện gọi TopSequencesTraversal để tìm 2 mẫu tuần tự tối đại:

Lần lượt thực hiện TopSequencesTraversal(<(*a*)>, $D|_{<(a)>}$, 1), TopSequencesTraversal(<(*b*)>, $D|_{<(b)>}$, 1), ….

❖ TopSequencesTraversal(<(*a*)>, $D|_{<(a)>}$, 1)

Chiều dài mẫu <(*a*)> = *min_l* = 1 nên:

- SavePattern(<(*a*)>) nhưng mẫu này không lưu vào tập *F* vì mẫu <(*a*)> tồn tại item *b* có thể mở rộng và cùng độ hỗ trợ <(*a*)>.
- Gọi PrefixSpanWSR(<(*a*)>, 1, $D|_{<(a)>}$, *F*), tìm được các mẫu phổ biến có độ dài là 2 đã được sắp xếp giảm dần theo độ hỗ trợ như sau: <(*a*)(*b*)>: 4, <(*a*)(*c*)>: 4, <(*ab*)>: 2, <(*a*)(*a*)>: 2, <(*a*)(*d*)>: 2, <(*a*)(*e*)>: 2, <(*a*)(*f*)>: 1, <(*ac*)>: 1, <(*ad*)>: 1, <(*ae*)>: 1, <(*af*)>: 1.

Thực hiện đệ qui <(*a*)(*b*)>: 4, ta tìm các mẫu phổ biến tối đại có độ dài là 2:

- SavePattern(<(*a*)(*b*)>) lưu vào tập *F* vì *SIDSum*(<(*a*)(*b*)>) = 10 chưa có trong bảng *Hash.*

  ⇨ Tập *F:* <(*a*)(*b*)>: 4, *SIDSum* = 10

- Chiều dài mẫu <(*a*)(*b*)> < 3, đệ qui PrefixSpanWSR(<(*a*)(*b*)>, 1, $D|_{<(a)(b)>}$, *F*), tìm được các mẫu phổ biến có độ dài là 3 đã được sắp xếp giảm dần theo độ hỗ trợ như sau: <(*a*)(*bc*)>: 2, <(*a*)(*b*)(*a*)>: 2, <(*a*)(*b*)(*c*)>: 2, <(*a*)(*b*)(*d*)>: 1, <(*a*)(*b*)(*e*)>: 1, <(*a*)(*b*)(*f*)>: 1.

Thực hiện đệ qui <(*a*)(*bc*)>: 2, tìm các mẫu phổ biến tối đại có độ dài là 3:

- SavePattern(<(*a*)(*bc*)>) nhưng mẫu này không lưu vào tập *F* vì mẫu <(*a*)(*bc*)> tồn tại item *a* có thể mở rộng và cùng độ hỗ trợ <(*a*)(*bc*)>.
- Chiều dài mẫu <(*a*)(*bc*)> = 3, dừng đệ qui PrefixSpanWSR(<(*a*)(*bc*)>, 1, $D|_{<(a)(bc)>}$, *F*).

Thực hiện đệ qui <(*a*)(*b*)(*a*)>: 2, tìm các mẫu phổ biến tối đại có độ dài là 3:

- SavePattern(<(*a*)(*b*)(*a*)>) lưu vào tập *F* vì *SIDSum*(<(*a*)(*b*)(*a*)>) = 10 chưa có trong bảng *Hash.*

⇨ Tập *F:* <(*a*)(*b*)>: 4, *SIDSum* = 10

<(*a*)(*b*)(*a*)>: 2, *SIDSum* = 3

- Chiều dài mẫu <(*a*)(*b*)(a)> = 3, dừng đệ qui PrefixSpanWSR(<(*a*)(*b*)(*a*)>, 1, $D|_{<(a)(b)(a)>}$, *F*).

Thực hiện đệ qui mẫu <(*a*)(*b*)(*c*)>: 2, tìm các mẫu phổ biến tối đại có độ dài là 3:

- SavePattern(<(*a*)(*b*)(*c*)>) lưu vào tập *F* vì *SIDSum*(<(*a*)(*b*)(*c*)>) = 5 chưa có trong bảng *Hash.*

Số lượng mẫu trong tập F ≥ 2 và độ hỗ trợ mẫu <(a)(b)(c)> lớn hơn 1, do đó xóa mẫu <(*a*)(*b*)>: 4 có độ dài mẫu ngắn nhất.

Cập nhật lại *minsup* = 2.

⇨ Tập *F:* <(*a*)(*b*)(*a*)>: 2, *SIDSum* = 3

<(*a*)(*b*)(*c*)>: 2, *SIDSum* = 5

- Chiều dài mẫu <(*a*)(*b*)(*c*)> = 3, dừng đệ qui PrefixSpanWSR(<(*a*)(*b*)(*c*)>, 2, $D|_{<(a)(b)(c)>}$, *F*).

Thực hiện đệ qui <(*a*)(*c*)>: 4, tìm các mẫu phổ biến tối đại có độ dài là 2:

- SavePattern(<(*a*)(*c*)>), không lưu vào tập *F* vì tồn tại chuỗi cha <(*a*)(*b*)(*c*).
- Chiều dài mẫu <(*a*)(*c*)> < *3*, đệ qui PrefixSpanWSR(<(*a*)(*c*)>, 2, $D|_{<(a)(c)>}$, *F*), tìm được các mẫu phổ biến có độ dài là 3 đã được sắp xếp giảm dần theo độ hỗ trợ như sau: <(*a*)(*c*)(*b*)>: 3, <(*a*)(*c*)(*c*)>: 3, <(*a*)(*c*)(*a*)>: 2,….

Thực hiện đệ qui <(*a*)(*c*)(*b*)>: 3, tìm các mẫu phổ biến tối đại có độ dài là 3:

- SavePattern(<(*a*)(*c*)(*b*)>) lưu vào tập *F* vì *SIDSum*(<(*a*)(*c*)(*b*)>) = 9 chưa có trong bảng *Hash*.

Số lượng mẫu trong tập *F* ≥ 2 và độ hỗ trợ mẫu <(*a*)(*c*)(*b*)> lớn hơn 2, do đó xóa mẫu <(*a*)(*b*)(*a*)>: 2.

Cập nhật lại *minsup* = 2.

⇨ Tập *F:* <(*a*)(*b*)(*c*)>: 2, *SIDSum* = 5

<(*a*)(*c*)(*b*)>: 3, *SIDSum* = 9

- Chiều dài mẫu <(*a*)(*c*)(*b*)> = 3, dừng đệ qui PrefixSpanWSR(<(*a*)(*c*)(*b*)>, 2, $D|_{<(a)(c)(b)>}$, *F*).

Thực hiện đệ qui <(*a*)(*c*)(*c*)>: 3, tìm các mẫu phổ biến tối đại có độ dài là 3.

- SavePattern(<(*a*)(*c*)(*c*)>) lưu vào tập *F* vì *SIDSum*(<(*a*)(*c*)(*c*)>) = 7 chưa có trong bảng *Hash*.

Số lượng mẫu trong tập *F* ≥ 2 và độ hỗ trợ mẫu <(*a*)(*c*)(*c*)> lớn hơn 2, do đó xóa mẫu <(*a*)(*b*)(*c*)>: 2 có độ hỗ trợ thấp nhất.

Cập nhật lại *minsup* = 3.

⇨ Tập *F:* <(*a*)(*c*)(*b*)>: 3, *SIDSum* = 9

<(*a*)(*c*)(*c*)>: 3, *SIDSum* = 7

- Chiều dài mẫu <(*a*)(*c*)(*c*)> = 3, dừng đệ qui PrefixSpanWSR(<(*a*)(*c*)(*c*)>, 3, $D|_{<(a)(c)(c)>}$, *F*).

❖ Tiếp tục thực hiện TopSequencesTraversal(<(*b*)>, $D|_{<(b)>}$, 1), TopSequencesTraversal(<(*c*)>, $D|_{<(c)>}$, 1), … thực hiện tương tự như tiền tố *a*.

Vậy với *k* = 2, *min_l* = 1, *max_l* = 3. Ta có tập *F* gồm 2 mẫu tuần tự tối đại như sau:

Bảng 2. Kết quả thuật toán TMSP với *k* =2, *min_l* =1, *max_l* =3

Mẫu tuần tự tối đại	***SIDSum***
<(*a*)(*c*)(*b*)>: 3	9
<(*a*)(*c*)(*c*)>: 3	7

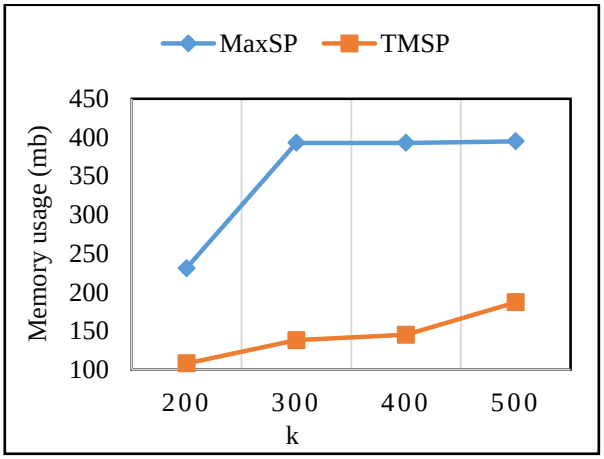
V. KẾT QUẢ THỰC NGHIỆM

Đặc điểm CSDL được mô tả như Bảng 3. CSDL thực - Leviathan được lấy từ kho dữ liệu máy học UCI, CSDL này mỗi itemset chỉ có một item. CSDL tổng hợp - C6T5S4I4N1kD1k và N1KD10K_a1 phát sinh bởi bộ phát sinh dữ liệu IBM, CSDL này mỗi itemset là một tập các item. Việc thực nghiệm được tiến hành trên máy tính DELL có cấu hình CPU Intel core i5-2410M, 6G RAM và sử dụng hệ điều hành Microsoft Windows 10, cài đặt trên ngôn ngữ lập trình Java.

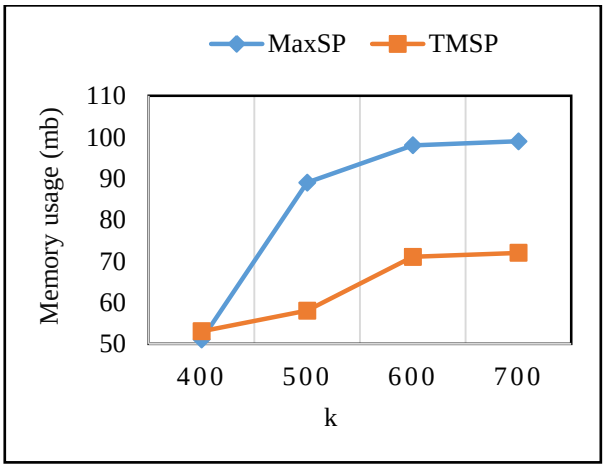
Hình 4, Hình 5 và Hình 6 cho thấy bộ nhớ sử dụng của thuật toán TMSP vượt trội hơn thuật toán MaxSP trên các tập dữ liệu thực nghiệm. Ví dụ 15: Xét cơ sở dữ liệu Leviathan (có số lượng item phân biệt lớn) với $k = 300$, bộ nhớ sử dụng MaxSP là 393 (mb) trong khi bộ nhớ sử dụng TMSP chỉ tốn 138 (mb).

Bảng 3. Đặc điểm của CSDL

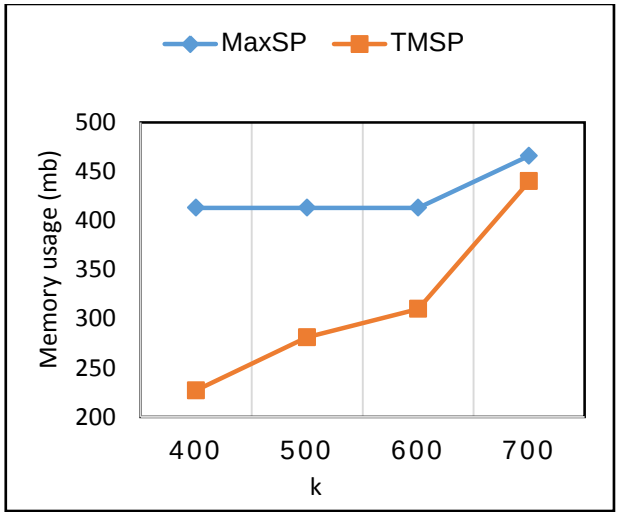
CSDL	**Số chuỗi**	**Số lượng item phân biệt**
Leviathan	5834	9025
C6T5S4I4N1kD1k	1000	1000
N1kD10k_a1	10000	1000



Hình 4. So sánh bộ nhớ sử dụng của hai thuật toán trên CSDL – Leviathan với *min_l* = 1, *mal_l* = 5



Hình 5. So sánh bộ nhớ sử dụng của hai thuật toán trên CSDL–C6T5S4I4N1kD1k với *min_l* =1, *mal_l* =3



Hình 6. So sánh bộ nhớ sử dụng của hai thuật toán trên CSDL – N1KD10K_a1 với *min_l* = 1, *mal_l* = 4

VI. KẾT LUẬN

Bài báo này trình bày thuật toán khai thác *k* mẫu tuần tự tối đại dựa trên cây PDB. Thuật toán TMSP sử dụng phương pháp chiếu tìm các mẫu tuần tự phổ biến để loại bỏ các mẫu con trong các mẫu thu được. Kết quả thực nghiệm cho thấy thuật toán TMSP đạt hiệu quả cao hơn MaxSP về bộ nhớ sử dụng và giúp người sử dụng dễ dàng tìm được số lượng mẫu cần khai thác mà không phải tinh chỉnh giá trị *minsup*.

Hướng phát triển: thuật toán khai thác k mẫu tuần tự tối đại dựa trên phương pháp chiếu để tìm các mẫu tuần tự phổ biến dễ dàng tìm được k mẫu tuần tự tối đại nhưng mất nhiều thời gian thực hiện. Vì vậy, hướng phát triển tiếp theo là sử dụng phương pháp vector bit động [10] để tối ưu hóa về thời gian thực hiện và bộ nhớ sử dụng.

LỜI CÁM ƠN

Nghiên cứu này được tài trợ bởi Quỹ phát triển Khoa học và Công nghệ Quốc gia (NAFOSTED) trong đề tài mã số 102.05-2015.07.

TÀI LIỆU THAM KHẢO

[1] J. P. J. PEI, J. H. J. HAN, B. MORTAZAVI-ASL, H. PINTO, Q. C. Q. CHEN, U. DAYAL, AND M.-C. H. M.-C. HSU, "PrefixSpan,: mining sequential patterns efficiently by prefix-projected pattern growth," *Proc. 17th Int. Conf. Data Eng.*, 2001.

[2] M. J. ZAKI, "SPADE: An efficient algorithm for mining frequent sequences," *Mach. Learn.*, vol. 42, no. 1–2, pp. 31–60, 2001.

[3] J. AYRES, J. GEHRKE, T. YIU, AND J. FLANNICK, "Sequential pattern mining using a bitmap representation," *Proc. eighth ACM SIGKDD Int. Conf. Knowl. Discov. data Min.*, pp. 429–435, 2002.

[4] P. FOURNIER-VIGER, A. GOMARIZ, T. GUENICHE, E. MWAMIKAZI, AND R. THOMAS, "TKS: Efficient mining of top-k sequential patterns," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8346 LNAI, no. PART

1, pp. 109–120, 2013.

[5] P. TZVETKOV, X. YAN, AND J. HAN, "TSP: Mining top-k closed sequential patterns," *Knowl. Inf. Syst.*, vol. 7, no. 4, pp. 438–457, 2005.

[6] K.SOHINI AND MR.V.PURUSHOTHAMA RAJU, "Mining Top-k Closed Sequential Patterns in Sequential Databases," *IOSR J. Comput. Eng.* , vol. 15, no. 4, pp. 20–23, 2013.

[7] P. FOURNIER-VIGER, C. W. WU, AND V. S. TSENG, "Mining maximal sequential patterns without candidate maintenance," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8346 LNAI, no. PART 1, pp. 169–180, 2013.

[8] P. FOURNIER-VIGER, C. W. WU, A. GOMARIZ, AND V. S. TSENG, "VMSP: Efficient vertical mining of maximal sequential patterns," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8436 LNAI, pp. 83–94, 2014.

[9] M. J. ZAKI AND W. MEIRA JR., *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press, New York, 2014.

[10] M.-T. TRAN, B. LE, AND B. VO, "Combination of dynamic bit vectors and transaction information for mining frequent closed sequences efficiently," *Eng. Appl. Artif. Intell.*, vol. 38, pp. 183–189, 2015.

Nhận bài ngày: 09/03/2016

SƠ LƯỢC VỀ TÁC GIẢ

LÊ HOÀI BẮC

Hiện là Phó Trưởng khoa, Trưởng Bộ môn Khoa học Máy tính, Khoa CNTT, Trường ĐH 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 QUYÊN

Tốt nghiệp ĐH ngành CNTT năm 2008 tại Trường ĐH Sư phạm kỹ thuật TP HCM và Cao học ngành CNTT năm 2015 tại Trường ĐH Công nghệ TP HCM.

Hiện công tác tại Khoa CNTT, Trường Cao đẳng nghề Công nghệ cao Đồng An.

Hướng nghiên cứu: khai thác dữ liệu.

Email: nguyenquyen@dongan.edu.vn