CSPM-DBV: KHAI THÁC HIỆU QUẢ MẪU TUẦN TỰ ĐÓNG DỰA TRÊN CẤU TRÚC VECTOR BIT ĐỘNG

Trần Minh Thái, Phạm Đức Thành
Khoa Công nghệ thông tin, Trường Đại học Ngoại ngữ - Tin học TP. HCM
minhthai@huflit.edu.vn, phamducthanh@huflit.edu.vn

TÓM TẮT

Khai thác mẫu tuần tự đang được nghiên cứu rộng rãi do chúng có nhiều ứng dụng rộng rãi trong thực tiễn. Phần lớn các nghiên cứu trên khai thác dữ liệu tuần tự tập trung vào khai thác tất cả các mẫu tuần tự. Điều này sẽ sinh ra các kết quả dư thừa, làm gia tăng không gian lưu trữ và thời gian thực thi không cần thiết. Trong khi đó, khai thác mẫu tuần tự đóng có thể giúp duy trì số lượng mẫu nhỏ hơn mà vẫn đảm bảo được những thông tin đầy đủ khi trích xuất luật. Chính vì vậy các thuật toán khai thác mẫu tuần tự đóng được ra đời. Các thuật toán này thông thường sử dụng chiến lược duy trì và kiểm tra mẫu nên không hiệu quả trên tập dữ liệu tuần tự lớn. Nhằm giải quyết vấn đề trên, bài báo này đề xuất một thuật toán, với tên gọi CSPM-DBV (Closed Sequential Pattern Mining Based on Dynamic Bit Vectors), nhằm khai thác hiệu quả mẫu tuần tự đóng thông qua cấu trúc vector bit động. Thuật toán áp dụng nhiều kỹ thuật nhằm giảm không gian lưu trữ và thời gian thực thi. Các kết quả thực nghiệm cho thấy rằng CSPM-DBV hiệu quả hơn thuật toán BIDE về mặt thời gian thực thi và bộ nhớ sử dụng.

Từ khóa: *Vector bit động, mẫu tuần tự, chuỗi phổ biến đóng, vector bit.*

1. Giới thiệu

Khai thác mẫu tuần tự phổ biến là vấn đề cơ bản trong khám phá tri thức và khai thác dữ liệu với nhiều ứng dụng rộng rãi. Như việc phân tích hành vi mua sắm của khách hàng, các mẫu truy cập web, các thí nghiệm khoa học, điều trị bệnh, ngăn ngừa thảm họa thiên nhiên và sự hình thành protein, v.v... Nhiều nghiên cứu đã hiệu chỉnh thuật toán AprioriAll algorithm [1] cho việc khai thác mẫu tuần tự phổ biến. Không giống như khai thác mẫu tuần tự phổ biến tổng quát, tìm tất cả các mẫu tuần tự phổ biến có thể có, khai thác mẫu tuần tự đóng chưa được nghiên cứu rộng rãi. Một số thuật toán được đề xuất như là CloSpan [10], và BIDE [13] có hiệu suất tốt trên một số trường hợp. Tuy nhiên, đối với dữ liệu tuần tự lớn hay ngưỡng phổ biến thấp thì vẫn còn nhiều hạn chế. Mặc dù, BIDE có thể phát hiện mẫu đóng và tỉa sớm các mẫu ứng viên không tiềm năng thay vì dùng cơ chế duy trì và kiểm tra mẫu ứng viên.

Trong bài báo này, chúng tôi đề xuất thuật toán CSPM-DBV, trong đó sử dụng cấu trúc dữ liệu nén và chia không gian tìm kiếm nhằm làm giảm không gian lưu trữ và thời gian thực thi cho việc khai thác mẫu phổ biến đóng. Thuật toán CSPM-DBV kế thừa cách tiếp cận phát hiện và tỉa sớm ứng viên không tiềm năng của BIDE thông qua cấu trúc dữ liệu mới giúp sớm loại bỏ những mẫu không cần thiết. Phần còn lại của bài báo được thể hiện các nội dung với bố cục như sau: Mục 2 trình bày các định nghĩa. Mục 3 tóm tắt các công trình nghiên cứu liên quan. Mục 4 và 5 trình bày thuật toán đề xuất và các kết quả thực nghiệm. Cuối cùng, kết luận và các hướng nghiên cứu tiếp theo được thể hiện trong Mục 6.

2. Định nghĩa bài toán

Xét cơ sở dữ liệu (CSDL) tuần tự với một tập các sự kiện phân biệt $I = \{i_1, i_2, i_3, \cdots, i_n\}$, trong đó i_j là một hạng mục (item) $1 \le j \le n$. Ví dụ, các item có thể là sữa, bia hay nước ngọt. Tập các hạng mục không thứ tự được gọi là tập các mục (itemset). Mỗi itemset được đặt trong cặp ngoặc tròn, ví dụ ⟨(ABC)⟩. Nhằm đơn giản ký hiệu, đối với các itemset chỉ có chứa một item thì cặp dấu ngoặc có thể loại bỏ, ví dụ itemset ⟨B⟩.

Một dãy $S = \{s_1, s_2, s_3, \cdots, s_m\}$ là một danh sách các itemset, trong đó s_j $(1 \le j \le m)$. Giả sử ℓ là số lượng các item trong một dãy. Một dãy có chiều dài ℓ được gọi là $\ell-$sequence. Ví dụ, ⟨AB(AE)CB⟩ là dãy 6-sequence. Một dãy $S_a = \langle a_1, a_2, \cdots, a_m \rangle$ được chứa trong một dãy $S_b = \langle b_1, b_2, \cdots, b_n \rangle$ nếu tồn tại một số nguyên $1 \le i_1 < i_2 < \cdots < i_m \le n$ sao cho $a_i = b_{i1}, a_2 = b_{i2}, \cdots, a_m = b_{im}$. Nếu dãy S_a được chứa trong dãy S_b thì S_a được gọi là dãy con (subsequence) của dãy S_b và S_b được gọi là dãy cha của S_a, ký hiệu là $S_a \subseteq S_b$. Một CSDL tuần tự được ký hiệu là

$D = \{s_1, s_2, s_3, \cdots, s_{|D|}\}$, trong đó $|D|$ là số lượng dãy trong D và s_i $(1 \leq i \leq |D|)$ là một giao dịch theo dạng <*dat, sequence*>, trong đó thuộc tính <*dat*> được dùng để mô tả thông tin của s_i tương ứng với thông tin giao dịch theo thời gian.

Bảng 1: *Ví dụ về CSDL tuần tự D*

Dat	Chuỗi
1	⟨CAA(AC)⟩
2	⟨AB(ABC)B⟩
3	⟨A(BC)ABC⟩
4	⟨AB(BC)A⟩

Hỗ trợ (support) của một dãy S_a trong một CSDL D được tính là số lượng sự xuất hiện dãy con S_a trong các giao dịch của D, được ký hiệu là $sup^D(S_a)$. Hỗ trợ của dãy có thể được thể hiện theo dạng tỉ lệ phần trăm ($\frac{sup^D(S_a)}{|D|}$). Hỗ trợ của một dãy được ký hiệu $sequence: support$. Trong đó, sequence thể hiện dãy và support thể hiện giá trị hỗ trợ. Ví dụ, một dãy ⟨AB⟩ với hỗ trợ là 3 thì được thể hiện là ⟨AB⟩:3.

Cho một ngưỡng hỗ trợ tối thiểu minSup được xác định bởi người dùng, một dãy S_a là dãy phổ biến (hay còn gọi là mẫu tuần tự phổ biến) trên D nếu $sup^D(S_a) \geq$ minSup. Nếu dãy S_a là phổ biến và không tồn tại dãy cha thích hợp S_b của S_a với cùng hỗ trợ thì S_a được gọi là dãy phổ biến đóng (hay còn gọi là mẫu tuần tự phổ biến đóng), nghĩa là, không tồn tại dãy S_b sao cho $S_a \subseteq S_b$ và $sup^D(S_a) = sup^D(S_b)$. Bài toán khai thác dãy phổ biến đóng là cố gắng tìm một tập hoàn chỉnh các dãy phổ biến đóng cho một tập dữ liệu tuần tự D đầu vào với một giá trị ngưỡng hỗ trợ tối thiểu minSup cho trước.

Ví dụ 1. Xét CSDL tuần tự D trong Bảng 1. Dữ liệu có ba item phân biệt I = {A, B, C} và bốn giao dịch, nghĩa là, |D| = 4. Giả sử rằng minSup = 2 (50% nếu tính theo tỉ lệ phần trăm). Nếu tất cả các dãy phổ biến của D được khai thác với minSup được cho thì sẽ có 32 dãy sau thu được gồm: S_{FS} = {⟨A⟩:4, ⟨AA⟩:4, ⟨AB⟩:3, ⟨AC⟩:4, ⟨(AC)⟩:2, ⟨AAB⟩:2, ⟨AAC⟩:2, ⟨A(AC)⟩:2, ⟨ABA⟩:3, ⟨ABB⟩:3, ⟨ABC⟩:3, ⟨A(BC)⟩:3, ⟨ACA⟩:2, ⟨ACB⟩:2, ⟨ABAB⟩:2, ⟨AB(BC)⟩:2, ⟨A(BC)A⟩:2, ⟨A(BC)B⟩:2, ⟨B:3⟩, ⟨BA⟩:3, ⟨BB⟩:3, ⟨BC⟩:3, ⟨(BC)⟩:3, ⟨BAB⟩:2, ⟨B(BC)⟩:2, ⟨(BC)A⟩:2, ⟨(BC)B⟩:2, ⟨C⟩:4, ⟨CA⟩:3, ⟨CB⟩:2, ⟨CC⟩:2, ⟨CAC⟩:2}.

Trong khi đó, nếu khai thác dãy phổ biến đóng thì kết quả thu được gồm: S_{FCS} = {⟨AA⟩:4, ⟨AC⟩:4, ⟨AAC⟩:2, ⟨A(AC)⟩:2, ⟨ABA⟩:3, ⟨ABB⟩:3, ⟨ABC⟩:3, ⟨A(BC)⟩:3, ⟨ABAB⟩:2, ⟨AB(BC)⟩:2, ⟨A(BC)A⟩:2, ⟨A(BC)B⟩:2, ⟨C⟩:4, ⟨CA⟩:3, ⟨CAC⟩:2}, mà chỉ gồm 15 dãy chiếm tỉ lệ 46.88% so với dãy phổ biến tổng quát.

Rõ ràng, dãy phổ biến đóng S_{FCS} thu được ít hơn nhiều so với dãy phổ biến tổng quát S_{FS}. Điều này là do dãy con S_a có cùng hỗ trợ như là của dãy cha S_b sẽ bị loại trừ bởi S_b mà không ảnh hưởng đến kết quả khai thác. Chẳng hạn như, dãy phổ biến ⟨(BC)A⟩:2 bị loại trừ bởi dãy phổ biến ⟨A(BC)A⟩:2 bởi vì ⟨(BC)A⟩ ⊆ ⟨A(BC)⟩ và $sup^D(\langle(BC)A\rangle) = sup^D(\langle A(BC)A\rangle) = 2$.

Định nghĩa 1 (dãy con của một dãy). Gọi S là một dãy tuần tự. $sub_{i,j}(S)$ (i ≤ j) được định nghĩa là dãy con có chiều dài (j-i +1) từ vị trí i đến vị trí j của S. Ví dụ, $sub_{1,3}(\langle BABC\rangle)$ là ⟨BAB⟩ và $sub_{4,4}(\langle BABC\rangle)$ là ⟨C⟩.

Định nghĩa 2 (mở rộng dãy từ dãy tuần tự 1-sequence). Gọi α và β là hai dãy 1-sequence phổ biến, $t_\alpha.p_\alpha$ và $t_\beta.p_\beta$ là các giao dịch và vị trí tương ứng của chuỗi α và β. Có thể có hai hình thức mở rộng của dãy (2.1) Mở rộng dạng Itemset: $< (\alpha\beta) > \{t_\beta.p_\beta\}$, nếu $(\alpha < \beta) \wedge (t_\alpha = t_\beta) \wedge (p_\alpha = p_\beta)$. (2.2) Mở rộng dạng Sequence: $< \alpha\beta > \{t_\beta.p_\beta\}$, nếu $(t_\alpha = t_\beta) \wedge (p_\alpha < p_\beta)$.

Định nghĩa 3 (mở rộng dãy từ dãy tuần tự k-sequence). Gọi α và β là hai dãy k-sequences phổ biến (k>1), $sub_{k,k}(\alpha) = u$, và $sub_{k,k}(\beta) = v$. $t_\alpha.p_\alpha$ và $t_\beta.p_\beta$ là các giao dịch và vị trí tương ứng của dãy α và β. Có thể có hai hình thức mở rộng dãy. (3.1) Mở rộng dạng Itemset: $\alpha +_i (v)\{t_\beta.p_\beta\}$, nếu

$(u < v) \wedge (t_\alpha = t_\beta) \wedge (p_\alpha = p_\beta)$. (3.2) Mở rộng dạng Sequence: $\alpha +_s (v)\{t_\beta . p_\beta\}$, nếu $(t_\alpha = t_\beta) \wedge (p_\alpha < p_\beta)$.

Định nghĩa 4 Một item e' có thể được thêm vào dãy mở rộng một trong ba vị trí: (4.1) $S' = s_1 s_2 \cdots s_n e' \wedge (sup^D(S') = sup^D(S))$; (4.2) $\exists i (1 \leq i < n) \mid S' = s_1 s_2 \cdots s_i e' \cdots s_n \wedge (sup^D(S') = sup^D(S))$; và (4.3) $S' = e' s_1 s_2 \cdots s_n \wedge (sup^D(S') = sup^D(S))$. Trong (4.1), item e' xuất hiện sau s_n, vì thế e' được gọi là forward-extension và S' được gọi là dãy forward-extension. Ví dụ, dãy ⟨AC⟩:4 được gọi là dãy forward-extension của dãy ⟨A⟩:4 bởi vì dãy ⟨C⟩ được mở rộng vào sau dãy ⟨A⟩ và hỗ trợ của chúng là 4. Trong (4.2) và (4.3), item e' xuất hiện trước s_n, vì thế item e' được gọi là backward-extension và S' được gọi là dãy backward-extension. Ví dụ, dãy ⟨CAC⟩:2 là backward-extension của dãy ⟨CC⟩:2 bởi vì ⟨A⟩ được mở rộng ở vị trí giữa của dãy ⟨CC⟩ và hỗ trợ của chúng là 2.

Hệ quả 1 (kiểm tra tính đóng của dãy). Nếu tồn tại một dãy S_b mà là forward-extension hay backward-extension của dãy S_a, do vậy S_a không là dãy đóng và S_a có thể được loại trừ một cách an toàn bởi dãy S_b.

Xét ví dụ 1, giả sử rằng S_a = ⟨CC⟩:2 và S_b = ⟨CAC⟩:2 thì ⟨CC⟩:2 sẽ bị loại trừ bởi ⟨CAC⟩ bởi vì $CC \subseteq CAC$ và $sup^D(\langle CC \rangle) = sup^D(\langle CAC \rangle) = 2$.

Định nghĩa 5. Gọi $S = s_1 s_2 \cdots s_n$. Vị trí bắt đầu của dãy S là vị trí xuất hiện đầu tiên của itemset s_1. Ví dụ, trong dãy ⟨AB(ABC)CB⟩, vị trí bắt đầu của dãy ⟨(ABC)⟩ là 3, và của dãy ⟨ABB⟩ là 1.

Hệ quả 2 (tỉa tiền tố). Xét tiền tố n-sequence $S_p = s_1 s_2 \cdots s_n$. Nếu tồn tại một item e trước vị trí bắt đầu của tiền tố S_p trong mỗi giao dịch của S_p trong D, mở rộng cho tiền tố S_p có thể được bỏ qua mà không ảnh hưởng đến kết quả khai thác.

Chứng minh. Giả sử rằng tồn tại một item e trước tất cả các vị trí bắt đầu trong các giao dịch có chứa tiền tố S_p, sinh ra một dãy mới $S'_p = eS_p$. Điều này nghĩa là $S_p \subseteq S'_p$ và $sup^D(S'_p) = sup^D(S_p)$). Do vậy, dãy mở rộng của tiền tố S_p sẽ bị loại trừ do dãy mở rộng của tiền tố S'_p (theo Hệ quả 1). Ví dụ, xét D trong Bảng 1. Không cần mở rộng cho tiền tố ⟨B⟩ bởi vì có tồn tại một dãy ⟨A⟩ mà xuất hiện trước ⟨B⟩ trong mỗi giao dịch mà có chứa tiền tố ⟨B⟩. Nếu chúng ta mở rộng cho tiền tố ⟨B⟩, kết quả thu được cũng sẽ bị loại trừ do mở rộng của tiền tố ⟨A⟩ đã chứa ⟨B⟩ và có cùng hỗ trợ.

3. Các công trình nghiên cứu liên quan

Khai thác dãy phổ biến được đề xuất lần đầu tiên vào năm 1995 bởi Agrawal và Srikant với thuật toán AprioriAll [1], thuật toán này dựa vào tính chất Apriori. Sau đó, cũng chính nhóm tác giả này mở rộng thêm bài toán khai thác dãy một cách tổng quát hơn với thuật toán GSP [2]. Kể từ đó, nhiều thuật toán khai thác dãy phổ biến được đề xuất nhằm cải tiến hiệu quả khai thác. Các thuật toán dùng nhiều cách tiếp cận khác nhau cho việc tổ chức dữ liệu và lưu trữ thông tin khai thác. Các thuật toán tiêu biểu gồm có SPADE [3], PrefixSpan [4], SPAM [5], và LAPIN-SPAM [6]. Thuật toán SPAM tổ chức dữ liệu theo dạng bitmap dọc và dùng cấu trúc cây từ điển để lưu trữ thông tin khai thác. PrefixSpan thì dùng phép chiếu cơ sở dữ liệu cho việc mở rộng dãy nhằm giảm không gian tìm kiếm với cách biểu diễn dữ liệu ngang. Thuật toán LAPIN-SPAM dùng danh sách để lưu trữ vị trí cuối cùng của các item và tập các vị trí biên của tiền tố để giảm phạm vi của không gian tìm kiếm.

Do khai thác toàn bộ dãy phổ biến sẽ tồn tại những dãy dư thừa, nên một số tác giả đã đề xuất khai thác dãy phổ biến không dư thừa nhằm giảm yêu cầu không gian lưu trữ và thời gian thực thi cho việc khai thác luật tuần tự. Một số thuật toán khai thác tập itemset phổ biến đóng và dãy phổ biến đóng nhằm giải quyết vấn đề dư thừa mẫu. Trong số này, có thể kể đến các thuật toán như CHARM [7], and CLOSET+ [8]. Phần lớn các thuật toán vừa liệt kê đều duy trì mẫu khai thác được để phục vụ cho việc kiểm tra tính đóng của mẫu rồi sau đó mới loại trừ nếu không thỏa yêu cầu. Điều này đòi hỏi nhiều bộ nhớ trong khai thác. Mặc dù, CLOSET+ dùng cấu trúc hash-index hai mức và cấu trúc cây cho việc lưu trữ itemset để giảm bộ nhớ và thời gian thực thi. Hay thuật toán CloSpan [9] sử dụng phương pháp duy trì và test mẫu kết hợp với cấu trúc hash-index cho việc lưu trữ chuỗi. Thuật toán này tỉa mẫu dùng các kỹ thuật như là CommomPrefix và Backward Sub-Pattern nhằm cải thiện vấn đề tìm kiếm. Tuy nhiên, chính vì chưa loại trừ sớm mẫu không tiềm năng, việc duy trì mẫu ứng viên làm gia tăng bộ nhớ sử dụng, và khi số lương mẫu ứng viên lớn thì đây là vấn đề chính cần phải được xem xét.

Nhằm giải quyết các vấn đề này, thuật toán BIDE [10] có sự cải tiến đáng kể là không lưu vết bất kỳ mẫu ứng viên nào cho việc kiểm tra dãy đóng mới được tạo ra hay không. Thay vào đó, thuật toán sử dụng kỹ thuật mở rộng hai chiều để kiểm tra mẫu phổ biến đóng hay không từ mẫu ứng viên trước khi tiến hành mở rộng mẫu. Hơn nữa, thuật toán sử dụng tiến trình xử lý BackScan để xác định ứng viên có thể không cần mở rộng nhằm giảm thời gian và không gian khai thác. Thuật toán này dùng kỹ thuật chiếu giả trên CSDL đã giảm đáng kể không gian lưu trữ và hiệu quả cho những ngưỡng hỗ trợ thấp. Tuy nhiên, việc chiếu và quét dữ liệu chiếu nhiều lần cho mỗi tiền tố cũng ảnh hưởng không nhỏ đến hiệu quả trong quá trình khai thác.

4. Đề xuất thuật toán

Trong phần này, chúng tôi trình bày một thuật toán đề xuất, với tên gọi CSPM-DBV, sử dụng cấu trúc vector bit động (Dynamic Bit Vector - DBV) kết hợp với các thông tin vị trí của giao dịch (CSPM-DBVPattern). Thông qua các chiến lược kiểm tra mẫu đóng và tỉa sớm ứng viên không tiềm năng giúp thuật toán CSPM-DBV thực hiện việc khai thác dãy phổ biến đóng một cách hiệu quả.

4.1. Cấu trúc dữ liệu DBV

Cấu trúc vector bit có thể được biểu diễn cho các itemset trong các giao dịch của dữ liệu chuỗi. Bit '1' chỉ thị rằng item xuất hiện trong giao dịch và bit '0' chỉ thị trường hợp ngược lại. Tuy nhiên, sẽ có nhiều bit '0' tồn tại trong vector bit này trong quá trình xử lý. Vì thế, nó sẽ lưu nhiều bit '0' không cần thiết làm gia tăng bộ nhớ và thời gian xử lý. Do vậy, để giải quyết vấn đề này, cấu trúc vector bit động (DBV) được sử dụng [11]. Mỗi DBV bao gồm 2 phần: (1) Start bit: vị trí xuất hiện đầu tiên của bit '1' và (2) Bit vector: chuỗi bit từ bit '1' xuất hiện đầu tiên đến bit '1' xuất hiện cuối cùng. Cấu trúc DBV được lưu trữ theo định dạng dọc. Hỗ trợ của dãy có thể được tính một cách dễ dàng bằng cách đếm số lượng bit '1' (Bảng 2).

Bảng 2: *Chuyển đổi D trong Bảng 1 thành định dạng DBV*

Item	Dat	Bit-vector		Start bit	Bit-vector	Giá trị
A	1, 2, 3, 4	1 1 1 1	Chuyển đổi sang DBV	1	1 1 1 1	15
B	2, 3, 4	1 1 1 0		2	1 1 1	7
C	1, 2, 3, 4	1 1 1 1		1	1 1 1 1	15

4.2. Cấu trúc dữ liệu CSPM-DBVPattern

Cấu trúc CSPM-DBVPattern kết hợp cấu trúc DBV với thể hiện thông tin dãy. Mỗi CSPM-DBVPattern gồm 2 phần: (1) Sequence: Dãy tuần tự và (2) BlockInfo: DBV và danh sách vị trí xuất hiện dãy trong các giao dịch. Danh sách vị trí của mỗi giao dịch được thể hiện theo dạng startPos: {list positions}. Trong đó, startPos là vị trí xuất hiện đầu tiên của dãy trong mỗi giao dịch và list positions là danh sách các vị trí tương ứng của dãy.

Ví dụ 2. Xét D (Bảng 1), chuỗi ⟨A⟩ tồn tại trong giao dịch 1, 2, 3, và 4. Với giao dịch thứ nhất, dãy ⟨A⟩ xuất hiện tại các vị trí {2, 3, 4}. Và vị trí bắt đầu là 2, và vì thế 2: {2, 3, 4} được lưu. Tương tự, dãy ⟨A⟩ xuất hiện tại vị trí {1, 3}, vị trí bắt đầu là 1 nên sẽ lưu 1:{1, 3}.

4.3. Thuật toán CSPM-DBV

Thuật toán CSPM-DBV gồm bốn pha chính: (1) chuyển đổi CSDL tuần tự thành cấu trúc CSPM-DBVPattern, (2) kiểm tra tính đóng của dãy phổ biến, (3) tỉa tiền tố, và (4) mở rộng dãy.

Bảng 3: *Các môđun trong thuật toán CSPM-DBV: (a) CSPM-DBV, (b) DBV-Build-Pattern, and (c) DBV-Pattern-Extension*

Method: CSPM-DBV (D, minSup)
Input: CSDL tuần tự D và ngưỡng hỗ trợ minSup
Output: Tập chuỗi phổ biến đóng: FCS

1. $FCS_1 = \{i_{CloFS-DBVPattern(i)} | i \in I \wedge \sup(i) \geq minSup\};$
2. Sort (FCS_1) increase order by item i;
3. listPrefix = Count(FCS_1);
4. DBV-Build-Pattern (listPrefix, minSup, FCS_1);

```
5.  return FCS;
```

(a) CloFS-DBV

```
Method: DBV-Build-Pattern (listPrefix, minSup, FCS_level)
Input: Danh sách số lượng chuỗi với cùng tiền tố listPrefix, minSup và FCS
Output: FCS
6.  i=1, newList = ∅, newPattern = ∅;
7.  For (each k in listPrefix) do
8.      j = i + k;
9.      list_ij = list_i to list_j;
10.          newPattern = DBV-Pattern-Extension (list_ij, minSup, FCS_level, newList)
11.      i = j+1;
12.      If (newPattern = ∅)
13.           return FCS_level;
14.      FCS_level+1 = FCS_level ∩ newPattern;
15.      DBV-Build-Pattern (newList, minSup, FCS_level+1);
```

(b) DBV-Build-Pattern

```
Method: DBV-Pattern-Extension (listSeq, minSup, FCS_level, listPrefix)
Input: Danh sách tiền tố listSeq, minSup, FCS và listPrefix
Output: Tập chuỗi phổ biến đóng newPattern
16. newPattern = ∅;
17. For (each S_p in listSeq) do
18.   d=0;
19.   If (!Pruning a Prefix (S_p, FCS_level))
20.   For (each a in listSeq) do
21.        s = SExtension(S_p, a);
22.        If (sup(s)≥minSup∧!B-SExt (s, FCS_level)∧!F-SExt (s, FCS_level))
23.            newPattern = newPattern ∩ s;
24.            d=d+1;
25.   For (each b after S_p in listSeq) do
26.        s = IExtension(S_p, b);
27.        If (sup(s)≥minSup )∧!F_IExt (s, FCS_level))
28.            newPattern = newPattern ∩ s;
29.            d=d+1;
30.   If (d>0)
31.        listPrefix = listPrefix ∩ d;
32.        If(B-SExt(S_p,FCS_level)∨F-SExt(S_p,FCS_level)∨F-SExt(S_p,FCS_level))
33.                Remove S_p;
34. return newPattern;
```

(c) DBV-Pattern-Extension

Bảng 3 trình bày thuật toán CSPM-DBV được đề xuất. Đầu tiên, thuật toán duyệt CSDL D để tìm các dãy 1-sequence phổ biến và lưu trữ chúng trong FCS_1 trong cấu trúc CSPM-DBVPattern (dòng 1). Sau đó, các item trong FCS1 được sắp xếp theo thứ tự tăng dần (dòng 2) nhằm làm giảm các bước trong pha mở rộng dãy. Tại dòng 4, thuật toán thực hiện mở rộng dãy cho các dãy 2-sequence dựa vào các itemset ứng viên trong FCS1.

Mở rộng dãy được thực hiện theo nhóm các ứng viên với cùng tiền tố tại cùng mức; các số lượng tiền tố trong mỗi nhóm được lưu trữ trong listPrefix. Đầu tiên, listPrefix chứa số lượng các ứng viên trong FCS1 (dòng 3). Từ dòng 7 đến dòng 11 được dùng để mở rộng dãy tương ứng trong cùng nhóm tiền tố. Dòng 17 đến dòng 33 thể hiện mở rộng dãy trong hai hình thức: Mở rộng sequence (dòng 21) và mở rộng itemset (dòng 26). Trước khi mở rộng dãy, thuật toán tiến hành kiểm tra và loại trừ sớm tiền tố mà không có khả năng mở rộng thành dãy phổ biến đóng bằng cách dùng Hệ quả 2 (dòng 19). Nếu dãy thu được là phổ biến đóng thông qua thực hiện thao tác backward-extension và

forward-extension (dòng 22 và dòng 27), chúng ta sẽ lưu trong newPattern. Sau khi mở rộng cho tiền tố hoàn tất, số lượng tiền tố trong listPrefix được cập nhật (dòng 31). Quá trình được lặp lại (dòng 15) cho đến khi không còn dãy phổ biến đóng nào được sinh ra (dòng 13). Để giảm chi phí tính toán, chúng tôi chỉ đánh dấu dãy không phổ biến đóng và xóa chúng sau khi hoàn tất mở rộng ở mức kế tiếp.

Bảng 4: *Item ⟨A⟩, ⟨B⟩, và ⟨C⟩ được chuyển đổi thành CSPM-DBVPattern*

Sequence	⟨A⟩				⟨B⟩			⟨C⟩			
Start bit	1				2			1			
Value	15				7			15			
Index	4	3	2	1	4	3	2	4	3	2	1
Position	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}	2:{2,3}	2:{2,4}	2:{2,3,4}	3:{3}	2:{2,5}	3:{3}	1:{1,4}

Ví dụ 3. Trong ví dụ này minh họa trường hợp mở rộng dãy cho thuật toán CSPM-DBV với dữ liệu D trong Bảng 1 và minSup = 2 (50%). Sau khi thực hiện dòng 2 và 3 các dãy phổ biến 1-sequence được lưu trữ, nghĩa là, FCS_1 = {⟨A⟩: 4, ⟨B⟩: 3, ⟨C⟩: 4} (Bảng 4).

Bảng 5: *Ví dụ mở rộng sequence cho tiền tố ⟨A⟩ với ⟨A⟩ tạo thành dãy ⟨AA⟩*

Sequence	⟨AA⟩			
Start bit	1			
Value	15 & 15 = 15			
Index	4	3	2	1
Position ⟨A⟩	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}
Position ⟨A⟩	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}
Position ⟨AA⟩	1:{4}	1:{3}	1:{3}	2:{3,4}

Bảng 6: *Ví dụ mở rộng itemset cho tiền tố ⟨A⟩ với ⟨B⟩ tạo thành dãy ⟨(AB)⟩*

Sequence	⟨(AB)⟩			
Start bit	1			
Value	15 & 7 = 7			
Index	4	3	2	1
Position ⟨(A)⟩	1:{1,4}	1:{1,3}	1:{1,3}	2:{2,3,4}
Position ⟨(B)⟩	2:{2,3}	2:{2,4}	2:{2,3,4}	
Position ⟨(AB)⟩	∅	∅	3:{3}	

Trong ví dụ này, tiền tố ⟨A⟩ không là dãy phổ biến đóng sau khi xử lý backward-extension, và tiền tố ⟨B⟩ bị tỉa sau khi xử lý kiểm tra tiền tố. Thuật toán thực hiện mở rộng dãy để tạo các dãy phổ biến đóng 2-sequence. Bắt đầu với tiền tố ⟨A⟩, xử lý mở rộng với dãy ⟨A⟩, ⟨B⟩, và ⟨C⟩ trong hình thức mở rộng sequence (Bảng 5) và mở rộng itemset (Bảng 6). Vị trí 3 và 4 của itemset ⟨(AB)⟩ là rỗng, do đó bit tương ứng cho những vị trí này được gán là '0'và itemset này bị xóa ($\sup^D(\langle(AB)\rangle) = 1 <$ minSup).

5. Kết quả thực nghiệm

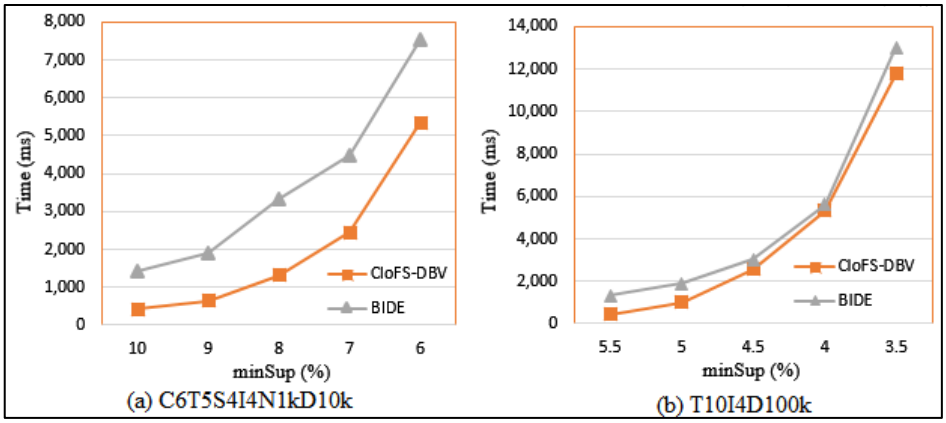
Thực nghiệm được thực hiện để đánh giá thuật toán được đề xuất. Tất cả các thuật toán được cài đặt trên máy tính CPU Intel Core Duo 2.0-GHz với bộ nhớ 4 GB chạy hệ điều hành Windows 8.1. Thuật toán BIDE được sử dụng trong quá trình so sánh. Các dữ liệu được dùng trong thực nghiệm

được phát sinh bởi công cụ sinh dữ liệu tổng hợp bởi IBM. Định nghĩa các tham số dùng để sinh dữ liệu tổng hợp được mô tả trong Bảng 7. Tập mẫu được sinh ra của hai thuật toán hoàn toàn giống nhau.

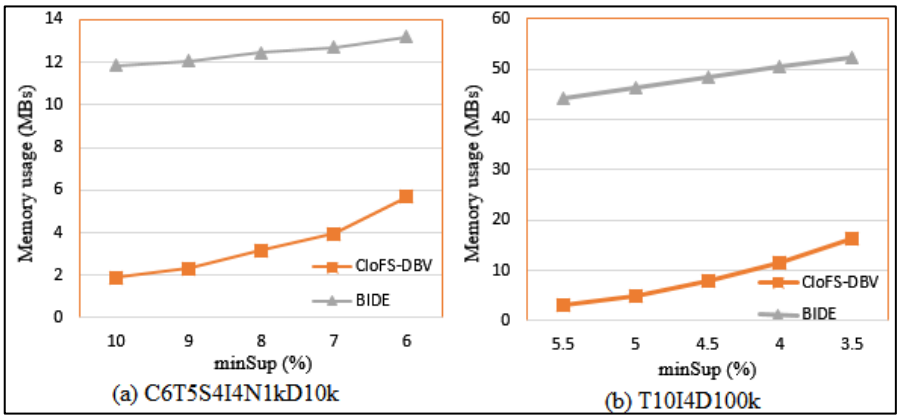
Bảng 7: *Các tham số được dùng trong sinh dữ liệu từ IBM*

C	Số lượng trung bình itemset trên mỗi dãy
T	Số lượng trung bình item trên mỗi itemset
S	Số lượng trung bình itemset trong dãy cực đại
I	Số lượng trung bình item trong dãy cực đại
N	Số lượng item phân biệt
D	Số lượng dãy

So sánh thời gian thực thi (được tính theo đơn vị milli giây) được thực hiện trên dữ liệu C6T5S4I4N1kD10k và T10I4D100k. Hình 1a thể hiện thời gian thực thi cho giá trị minSup từ 6% đến 10%, và Hình 1b thể hiện thời gian thực thi cho giá trị minSup từ 3.5% đến 5.5%. Thời gian thực thi cho các hai thuật toán tăng với các giá trị minSup giảm. Trong đó, thuật toán CSPM-DBV thực hiện nhanh hơn trong đa số các trường hợp. Điều này có được là do CSPM-DBV sử dụng cấu trúc DBV và các thao tác mở rộng dãy hoàn toàn dựa vào các phép toán trên bit. Hơn nữa, CSPM-DBV áp dụng các chiến lược tỉa sớm và kiểm tra mẫu ứng viên phù hợp trong quá trình khai thác mẫu.



Hình 1: *So sánh thời gian thực thi cho nhiều giá trị minSup khác nhau đối với các CSDL (a) C6T5S4I4N1kD10k và (b) T10I4D100k*



Hình 2: *So sánh bộ nhớ sử dụng cho nhiều giá trị minSup khác nhau đối với CSDL (a) C6T5S4I4N1kD10k và (b) T10I4D100k*

Kết quả thể hiện trong Hình 2 cho thấy bộ nhớ sử dụng (được tính theo đơn vị MB) của hai thuật toán với các giá trị minSup khác nhau. Một cách tương tự, với giá trị minSup giảm, số lượng các ứng viên được sinh ra sẽ gia tăng cho nên đòi hỏi nhiều bộ nhớ yêu cầu cũng nhiều hơn trong cả hai

thuật toán. Tuy nhiên, thuật toán CSPM-DBV yêu cầu không gian lưu trữ ít hơn nhiều so với thuật toán BIDE do CSPM-DBV dùng cấu trúc dữ liệu nén.

6. Kết luận và hướng nghiên cứu tiếp theo

Bài báo này trình bày một cách tiếp cận khai thác mẫu tuần tự đóng trên CSDL tuần tự với thuật toán được đề xuất là CSPM-DBV. Thuật toán CSPM-DBV được chia thành hai bước chính: (1) Dữ liệu ban đầu được chuyển đổi sang biểu diễn định dạng dọc được gọi là CSPM-DBVPattern, trong đó mỗi CSPM-DBVPattern lưu trữ vị trí của các dãy phổ biến đóng dưới dạng cấu trúc DBV; (2) Các dãy phổ biến đóng được kiểm tra và phát sinh, tỉa sớm các tiền tố. Thuật toán CSPM-DBV chỉ duyệt CSDL một lần duy nhất và tính toán hỗ trợ của dãy thông qua thông tin vector bit đã được lưu trữ. Do sử dụng cấu trúc dữ liệu nén và kết hợp các chiến lược tỉa và kiểm tra tính đóng của dãy đơn giản, thuật toán CSPM-DBV hiệu quả hơn thuật toán BIDE về mặt bộ nhớ sử dụng và thời gian thực thi.

Bên cạnh những điểm mạnh, thuật toán CSPM-DBV có một số giới hạn cần được phát triển và mở rộng thêm trong tương lai. Chẳng hạn như, các vị trí xuất hiện của dãy cần được biểu diễn theo cấu trúc DBV, thực hiện khai thác song song trên dữ liệu lớn phân tán, v.v… Hơn nữa, CSPM-DBV chỉ mới tập trung khai thác theo từng giao dịch đơn. Do vậy, vấn đề khai thác mẫu liên giao dịch cần quan tâm nghiên cứu mở rộng do bài toán này có nhiều ứng dụng trong thực tế, nhất là trong lĩnh vực phân tích hành vi khách hàng.

TÀI LIỆU THAM KHẢO

[1] R. Agrawal, R. Srikant. Mining sequential patterns, *Proceedings of IEEE International Conference on Data Engineering*, 3–14, 1995.

[2] R. Srikant, R. Agrawal. Mining sequential patterns: Generalizations and performance improvements, *In Intl. Conf. Extending Database Technology*, 3-17, 1996.

[3] M. Zaki, SPADE: An efficient algorithm for mining frequent sequences, *Machine Learning*, **42**:31–60, 2001.

[4] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu, PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth, *Int'l Conf. Data Engineering*, 215-224, 2001.

[5] J. Ayres, J. Gehrke, T. Yiu, J. Flannick., Sequential pattern mining using a bitmap representation, *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Edmonton, Alberta, Canadapp*, 429-435, 2002.

[6] Zhenglu Yang, Masaru Kitsuregawa, LAPIN-SPAM: An Improved Algorithm for Mining Sequential Pattern, *ICDE Workshops*, 2004.

[7] M. Zaki, C. Hsiao, CHARM: An Efficient Algorithm for Closed Itemset Mining, *Proc. SIAM Int'l Conf. Data Mining* (SDM'02), 457-473, 2002.

[8] J. Wang, J. Han, and J. Pei, CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets, *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining* (SIGKDD'03), 236-245, 2003.

[9] X. Yan, J. Han, R. Afshar, CloSpan: mining closed sequential patterns in large datasets, *Proceedings of SIAM International Conference on Data Mining*, 166-177, 2003.

[10] J. Wang, J. Han, BIDE: efficient mining of frequent closed sequences, *Proceedings of IEEE International Conference on Data Engineering*, 79–90, 2007.

[11] B. Vo, T.P. Hong, B. Le, DBV-Miner: A Dynamic Bit-Vector approach for fast mining frequent itemsets, *Expert Systems with Applications*, **39**:8:7196-7206, 2012.

CSPM-DBV: MINING CLOSED SEQUENTIAL PATTERNS EFFICIENTLY BASED ON DYNAMIC BIT VECTORS

Tran Minh Thai[1], Pham Duc Thanh[2]

(1) (2) Department of Information Technology, HUFLIT

minhthai@huflit.edu.vn, phamducthanh@huflit.edu.vn

Abstract: Frequent sequence mining is being extensively studied because they have broad applications. Most studies on sequence data have focused on finding all possible frequent sequences.

This will produce redundant results, increasing unnecessary storage space and execution time. Meanwhile, mining frequent closed sequences can help maintain a smaller number of patterns while providing sufficient information when extracting rules. Therefore, the algorithms to mine frequent closed sequences were proposed. These algorithms often use a candidate maintenance-and-test paradigm that is not effective on large datasets. To overcome the above problem, this paper proposes an algorithm, called CSPM-DBV, to effectively mine the frequent closed sequences through dynamic bit vector structure. Various methods are employed to reduce memory usage and run time. Experimental results show that CSPM-DBV is more efficient than BIDE algorithm in terms of execution time and memory usage.

Keywords: *Dynamic bit vector, Sequential Pattern, Frequent closed sequence, Bit vector.*

Trần Minh Thái tốt nghiệp cử nhân ngành Công nghệ Phần mềm vào năm 2001 và thạc sỹ Tin học vào năm 2006 tại trường Đại học Khoa học Tự nhiên thành phố Hồ Chí Minh, nhận bằng tiến sỹ Khoa học Máy tính vào năm 2017 do Đại học Quốc gia thành phố Hồ Chí Minh cấp. Anh ta từng là giảng viên và quản lý khoa Công nghệ Thông tin trường Cao đẳng Công nghệ Thông tin thành phố Hồ Chí Minh từ năm 2002 đến 2015. Từ năm 2015 đến hiện tại, anh ta là giảng viên và là trưởng bộ môn Hệ thống Thông tin thuộc khoa Công nghệ Thông tin trường Đại học Ngoại ngữ Tin học thành phố Hồ Chí Minh. Lĩnh vực nghiên cứu chính của anh ta liên quan đến vấn đề khai thác dữ liệu và xử lý dữ liệu lớn.

Phạm Đức Thành nhận học vị Thạc sĩ năm 2006 tại Đại học Quốc gia Thành phố Hồ Chí Minh; hiện đang là Giảng viên công tác tại khoa Công nghệ Thông tin trường Đại học Ngoại ngữ Tin học Tp. Hồ Chí Minh; lĩnh vực nghiên cứu đang quan tâm là: khai thác dữ liệu.