MỘT THUẬT TOÁN HIỆU QUẢ ĐỂ KHAI THÁC TẬP HỮU ÍCH TRUNG BÌNH CAO

Phạm Tuấn Khiêm*, Nguyễn Văn Lễ*
*Khoa Công nghệ thông tin, Trường Đại học Công nghiệp Thực phẩm TPHCM

Tóm tắt: Khai thác tập hữu ích trung bình cao (High Average Utility Itemset - HAUI) đã được nghiên cứu rộng rãi nhằm khắc phục những hạn chế của tập hữu ích cao (High Utility Itemset - HUI) trong việc đánh giá kết quả của người dùng. Tập hữu ích trung bình cao thể hiện các tập mặt hàng có độ hữu ích cao thật sự. Trong đó, yếu tố chiều dài của tập mặt hàng được xem xét, điều này đã loại bỏ được những tập hữu ích cao có chứa nhiều mặt hàng kém ý nghĩa trong kết quả phân tích kinh doanh. Gần đây, nhiều thuật toán đã được đề xuất để khai thác tập hữu ích trung bình cao, tuy nhiên hiệu suất thực thi vẫn chưa hiệu quả. Trong bài báo này, chúng tôi đề xuất thuật toán HAU-Miner để khai thác tập hữu ích trung bình cao một cách tốt hơn. Kết quả thực nghiệm trên hai nhóm cơ sở dữ liệu dày và thưa cho thấy thuật toán HAU-Miner có hiệu suất thực thi cao hơn thuật toán MHAI về số lượng ứng viên phát sinh, thời gian thực thi và bộ nhớ sử dụng.

Từ khóa: Tập hữu ích trung bình cao, khai thác dữ liệu, cơ sở dữ liệu giao dịch, chặn trên độ hữu ích trung bình, độ hữu ích trung bình.

I. GIỚI THIỆU

Khai thác tập phổ biến (FIs) trong cơ sở dữ liệu giao dịch là bài toán phổ biến và có nhiều ứng dụng trong việc khám phá tri thức trong cơ sở dữ liệu [1, 2]. Nhiều thuật toán đã được đưa ra để giải quyết vấn đề này. Trong đó cách tiếp cận thường dùng là mở rộng tập mẫu (Pattern-Growth) [3, 4]. FP-growth (Frequent Pattern Growth) ban đầu xây dựng cấu trúc FP-tree bằng cách sử dụng tập phổ biến 1 phần tử. Sau đó, trong quá trình khai thác, các cây FP-trees được tạo đệ quy và mỗi cây chứa một bảng chỉ mục được thiết kế để khai thác các tập phổ biến. Khai thác tập hợp phổ biến truyền thống chỉ xem xét tần suất xuất hiện của các tập mặt hàng trong cơ sở dữ liệu giao dịch nhưng lại bỏ qua các yếu tố quan trọng khác như số lượng, lợi nhuận và trọng lượng của các mặt hàng. Một vấn đề khác là FIs không xem xét các tập mục không phổ biến nhưng lại có thể đóng góp một lượng lớn lợi nhuận. Do đó, nếu chỉ xem xét tần suất xuất hiện là không đủ để xác định các tập phổ biến có lợi nhuận cao. Chính vì vậy, khai thác tập hữu ích cao (HUIM - high utility itemset mining) [5, 6, 7] đã được nghiên cứu trong thời gian gần đây và có những kết quả nhất định. HUIM xem xét thêm các thông tin về số lượng và lợi nhuận của các mặt hàng trong giao dịch, từ đó đánh giá tốt hơn độ hữu ích của tập mặt hàng đối với người dùng. Một mặt hàng hoặc tập các mặt hàng được gọi là tập hữu ích cao nếu độ hữu ích của chúng không nhỏ hơn ngưỡng độ hữu ích tối thiểu được xác định bởi người dùng. Một số thuật toán được đề xuất để khai thác tập mặt hàng theo cách tiếp cận này như: HUI-Miner [5], FHM [8], HUP-Miner [9].

Việc khai thác tập hữu ích cao có thể xuất hiện nhiều tập mặt hàng có số phần tử lớn là tập hữu ích cao, trong đó chỉ có một số mặt hàng có ý nghĩa thực tế về độ hữu ích cao, trong khi các mặt hàng khác trong tập có độ hữu ích thấp nên không có ý nghĩa trong thực tế. Điều này có thể gây khó khăn trong việc phân tích kinh doanh từ tập kết quả. Ví dụ tập mặt hàng gồm năm sản phẩm {A, B,C, D, E} là tập hữu ích cao do độ hữu ích lớn hơn giá trị ngưỡng cho trước. Nhưng thực tế hai phẩm A và B có độ hữu ích cao quyết định độ hữu ích cao cho cả tập, các sản phẩm C, D và E có độ hữu ích thấp. Do đó việc phân tích kết quả trên tập con {A, B} mới thực sự có ý nghĩa thay vì phân tích trên tập lớn {A,B,C,D,E}. Để giải quyết vấn đề trên, năm 2009, Hong và cộng sự [10] đề cập đến khái niệm tập hữu ích trung bình cao (High Average Utility Itemset - HAUI), trong đó có xem xét đến độ dài tập mặt hàng để tính độ hữu ích trung bình. Cũng như bài toán khai thác tập hữu ích cao, chiến lược mở rộng tập mẫu được sử dụng để khai thác tập hữu ích trung bình cao. Năm 2010, Lin và cộng sự đề xuất thuật toán HAUP-Growth [11] dựa trên cấu trúc cây để khai thác tập HAUIM nhưng chưa hiệu quả về thời gian thực thi và bộ nhớ lưu trữ. Gần đây, nhóm tác giả Unil Yun và Donggyu Kim đề xuất thuật toán MHAI [12] sử dụng cấu trúc dữ liệu Ulility List để khai thác tập HAUIM hiệu quả hơn trên cơ sở dữ liệu giao dịch. Tuy nhiên hiệu suất thực thi vẫn chưa cao, đặc biệt trên các cơ sở dữ liệu thưa. Trong bài báo này, chúng tôi đề xuất thuật toán HAU-Miner để khai thác tập hữu ích trung bình cao một cách hiệu quả.

Những đóng góp chính của bài báo:

1) Đề xuất cấu trúc dữ liệu RAU-List cải tiến từ cấu trúc Utility-List để lưu trữ dữ liệu trong quá trình khai thác tập hữu ích trung bình cao.
2) Cải tiến công thức tính độ hữu ích trung bình lớn nhất, được dùng để xác định một tập có được mở rộng hay không trong quá trình khai thác tập hữu ích trung bình cao.
3) Áp dụng chiến lược tỉa UB-Prune, MAU-Prune, EUCS, MLA-Prune để giảm không gian tìm kiếm.
4) Kết quả thực nghiệm so sánh với thuật toán MHAI cho thấy thuật toán HAU-Miner có thời gian thực hiện nhanh hơn thuật toán MHAI trên hai nhóm cơ sở dữ liệu là dày và thưa.

Cấu trúc bài báo được chia làm 6 phần. Phần 1 trình bày giới thiệu; Phần 2 trình bày các công trình liên quan;

Tác giả liên hệ: Phạm Tuấn Khiêm,
Email: khiempt@hufi.edu.vn
Đến tòa soạn: 23/4/2021, chỉnh sửa: 24/8/2021, chấp nhận đăng: 31/8/2021.

Phần 3 trình bày các định nghĩa và ký hiệu; Phần 4 trình bày thuật toán đề xuất HAU-Miner; Phần 5 trình bày kết quả thực nghiệm và đánh giá; Phần 6 trình bày kết luận và hướng phát triển.

II. CÁC CÔNG TRÌNH LIÊN QUAN

Các công trình nghiên cứu về khai thác tập hữu ích cao (HUI) đã mang lại nhiều giá trị về việc khám phá tri thức trong cơ sở dữ liệu giao dịch, đặc biệt là trong lĩnh vực kinh doanh. Năm 2005, Liu và công sự đề xuất thuật toán Two-Phase [13] để khai thác HUI trên hai pha thực thi. Pha đầu quét cơ sở dữ liệu để xây dựng tập ứng viên, Pha hai quét cơ sở dữ liệu nhiều lần để xác định tập HUI từ tập các ứng viên. Dựa trên cách tiếp cận này, một số thuật toán khác được đề xuất để nâng cao hiệu quả khai thác HUI như: TWU-Mining [14], UP-Growth [15], UP-Growth+ [16]. Tuy nhiên, việc thực hiện trên hai pha dẫn đến tốn nhiều thời gian thực thi và bộ nhớ sử dụng. Năm 2012, thuật toán HUI-Miner [5] được đề xuất cùng với cấu trúc dữ liệu Utility-List và chỉ thực hiện trên một pha đã làm tăng đáng kể hiệu suất thực thi so với các thuật toán trước đó. Năm 2014, Fournier và cộng sự đề xuất cấu trúc EUCS với chiến lược tỉa EUCP [7] làm giảm đáng kể không gian tìm kiếm so với thuật toán HUI-Miner. Năm 2017, Zida và cộng sự trình bày cơ chế chiếu và gộp trên cơ sở dữ liệu giao dịch và đề xuất thuật toán EFIM [17] cho kết quả khai thác vượt trội so với các thuật toán trước đó, đặc biệt trên cơ sở dữ liệu thưa. Năm 2018, Krishnamoorthy [18] tiếp tục phát triển chiến lược khai thác HUI bằng việc đề xuất cấu trúc dữ liệu mới là CUL kết hợp nhiều chiến lược tỉa hiệu quả làm giảm đáng kể không gian tìm kiếm, thời gian thời gian thực thi và bộ nhớ sử dụng.

Khai thác tập hữu ích trung bình cao được đề xuất lần đầu bởi Hong và cộng sự [10] vào năm 2009. Trong đó, chiều dài tập mục được sử dụng để tính độ hữu ích trung bình. Cách tiếp cận này đã loại bỏ được những tập hữu ích cao có chứa nhiều phần tử kém ý nghĩa trong phân tích, thay vào đó là những tập con gồm những phần tử có độ hữu ích cao thực sự có ý nghĩa. Độ hữu ích trung bình được đề xuất trong [19] để đánh giá giá trị của tập mục một cách tốt hơn so với khai thác tập hợp hữu ích cao. Hầu hết các tác giả đã sử dụng giá trị chặn trên độ hữu ích trung bình (auub - average utility upper-bound) để khai thác tập hữu ích trung bình cao. Nhưng các kết quả cho thấy, các thuật toán giải quyết trước đó vẫn còn chưa hiệu quả. Có thể thấy trong thuật toán TPAU [19] sử dụng hướng tiếp cận dựa trên Apriori, 2-pha quét cơ sở dữ liệu nhiều lần nên hiệu suất thực hiện vẫn chưa cao. Năm 2011, Lan và cộng sự đề xuất thuật toán PBAU [20] cải tiến dựa trên kỹ thuật chiếu trên cơ sở dữ liệu và cấu trúc chỉ mục. Tuy nhiên thuật toán vẫn thực thi trên 2-pha nên vẫn chưa hiệu quả thời gian chạy và bộ nhớ sử dụng. Năm 2016, Lin và cộng sự [21] đề xuất phương pháp khai thác HAUI dựa trên cấu trúc Utility List với thuật toán HAUI-Miner cho kết quả tốt hơn các thuật toán trước đó vì chỉ thực thi trên 1 pha. Một số thuật toán gần đây như IMHAUI được đề xuất bởi Kim và cộng sự [22] dựa trên cấu trúc cây IHAUI-Tree tối ưu hóa thông tin lưu trữ, kết quả thực nghiệm so sánh với thuật toán ITPAU [23] và HUPID [24]. Năm 2017, Yun và cộng sự đề xuất thuật toán MHAI [12] dựa trên cấu trúc Utility List với chiến lược tỉa dựa trên độ hữu ích trung bình lớn nhất cho kết quả thực thi tốt hơn các thuật toán khai thác HAUI trước đó. Năm 2018, Zhang và cộng sự đề xuất thuật toán FUP-HAUIMI [25] để khai thác tập hữu ích trung bình cao trên cơ sở dữ liệu động (Dynamic Databases). Thuật toán sử dụng cấu trúc dữ liệu AUL kết hợp với khái niệm FUP để xét mỗi tập mục ở 4 trường hợp sau khi cập nhật cơ sở dữ liệu. Năm 2019, Yildirim và cộng sự trình bày thuật toán MHAUIPNU [26] với đề xuất ngưỡng giới hạn trên $tubpn$ kết hợp với cấu trúc dữ liệu TUBPNULs để khai thác hiệu quả tập hữu ích trung bình cao trên cơ sở dữ liệu giao dịch có lợi nhuận âm. Gần đây, năm 2020, Wu và cộng sự đề xuất thuật toán APHAUIM [27] để khai thác tập hữu ích trung bình cao trên cơ sở dữ liệu tăng trưởng với khái niệm Pre-large gồm APHAUI và PAUUBI.

III. CÁC ĐỊNH NGHĨA VÀ KÝ HIỆU

Cho cơ sở dữ liệu giao dịch $D = \{T_1, T_2, \dots, T_n\}$, với n là số lượng giao dịch trong D, tập $I = \{i_1, i_2, \dots, i_m\}$ gồm m mục phân biệt trong D. $\forall T_j \in D$, $T_j = \{x_l | l = 1, 2, \dots, N_j, x_l \in I\}$, với N_j là số các mục trong giao dịch T_j. Bảng 1 biểu diễn ví dụ về cơ sở dữ liệu giao dịch D dùng để khai thác tập hữu ích trung bình cao, mỗi mục (cột 2 của Bảng 1) trong giao dịch (cột 1 của Bảng 1) có một số lượng (cột 3 của Bảng 1). Bảng 2 biểu diễn giá trị lợi nhuận của các mục.

Bảng 1. Cơ sở dữ liệu giao dịch

TID	Các mục	Số lượng
1	a, c, d, e	4, 2, 2, 1
2	a, b, c, e, f	7, 4, 7, 5, 1
3	a, b, d, f	1, 4, 1, 1
4	a, b, e, f	4, 7, 1, 1
5	b, c, e	2, 1, 2
6	a, b, c	8, 3, 8
7	c, d, e	3, 1, 1

Bảng 2. Lợi nhuận các mục

Các mục	a	b	c	d	e	f
Lợi nhuận	3	1	5	4	6	2

Mục tiêu của việc khai thác tập hữu ích trung bình cao là tìm trong cơ sở dữ liệu giao dịch D tất cả các tập có độ hữu ích trung bình không nhỏ hơn ngưỡng độ hữu ích nhỏ nhất cho trước. Các định nghĩa sau được sử dụng trong các nghiên cứu trước đây về khai thác tập hữu ích trung bình cao.

Định nghĩa 1. Độ hữu ích của mục i trong giao dịch T được định nghĩa là: $u(i,T) = p(i) * q(i,T)$. Trong đó p(i) là lợi nhuận của mục i, $q(i,T)$ là số lượng mua của mục i trong giao dịch T.

Ví dụ: Độ hữu ích của mục b trong T_2 là: $u(b,T_2) = p(b) * q(b,T_2) = 1 * 4 = 4$.

Định nghĩa 2. Độ hữu ích trung bình (Average Utility). Gọi $l(X)$ là số phần tử của tập mục X, độ hữu ích trung bình của X trong T ký hiệu là $au(X,T)$, được xác định:

$$au(X,T) = \frac{\sum_{i \in X \subseteq T} u(i,T)}{l(X)} \quad (1)$$

Ví dụ: Độ hữu ích trung bình của tập mục $\{b, c\}$ trong T_2 là:

$$au(bc,T_2) = \frac{iu(b,T_2) + iu(c,T_2)}{l(bc)} = \frac{1*4+5*7}{2} = 19.5$$

Độ hữu ích trung bình của X trong D, ký hiệu là $au(X)$, được định nghĩa:

$$au(X) = \sum_{X \subseteq T \in D} au(X,T) \quad (2)$$

Ví dụ: Độ hữu ích trung bình của tập mục $\{b,c\}$ trong Bảng 1 là

$$au(bc) = au(bc,T_2) + au(bc,T_5) + au(bc,T_6)$$
$$= \frac{1*4+5*7}{2} + \frac{1*2+5*1}{2} + \frac{1*3+5*8}{2}$$
$$= 19.5 + 3.5 + 21.5 = 44.5$$

Định nghĩa 3. Tập mục X được gọi là tập hữu ích trung bình cao (High Average Utility Itemset - HAUI) nếu độ hữu ích trung bình của X lớn hơn hoặc bằng giá trị ngưỡng độ hữu ích tối thiểu (minUtil). Với minUtil được cho trước bởi người dùng, ta có: $HAUIs = \{X \mid au(X) >= minUtil\}$ [12].

Ví dụ: Với minUtil=42. Tập $\{b,c\}$ trong Bảng 1 là tập hữu ích trung bình cao vì $au(bc) = 44.5 > 42$

Định nghĩa 4. Độ hữu ích lớn nhất (Maximum Utility) của giao dịch T, ký hiệu $maxU(T)$, là độ hữu ích lớn nhất trong số các mục có trong giao dịch, được định nghĩa:

$$maxU(T) = max(\{u(i,T) \mid i \in T\}) \quad (3)$$

Ví dụ: Độ hữu ích lớn nhất của T_4 trong Bảng 1 là:

$$maxU(T_4)$$
$$= max(u(a,T_4), u(b,T_4), u(e,T_4), u(f,T_4))$$
$$= max(12,7,6,2) = 12$$

Định nghĩa 5. Chặn trên độ hữu ích trung bình (Average-Utility Upper-Bound) của X trong D, ký hiệu $auub(X)$, là tổng của các độ hữu ích lớn nhất của các giao dịch chứa X, được định nghĩa:

$$auub(X) = \sum_{X \subseteq T \in D} maxU(T) \text{ [12] } (4)$$

Ví dụ: Với dữ liệu trong Bảng 1, $auub(f) = maxU(T_2) + maxU(T_3) + maxU(T_4) = 35 + 4 + 12 = 51$

Chiến lược tỉa 1 (UB-Prune): Nếu $auub(X) < minUtil$ thì mọi tập mục mở rộng từ X đều không là $HAUI$ [12] [23]

Bảng 3. Chặn trên độ hữu ích trung bình của tập 1 phần tử

Mục	a	b	c	d	e	f
auub	103	103	114	31	86	51

Dựa vào chiến lược tỉa 1, mục d có $auub(d) = 31 < minUtil = 42$. Do đó, mọi tập mục mở rộng từ d đều không phải là tập hữu ích trung bình cao và d bị loại bỏ khỏi cơ sở dữ liệu D trong quá trình khai thác.

Định nghĩa 6. (Thứ tự toàn phần). Thứ tự toàn phần của các mục trong cơ sở dữ liệu giao dịch D được sắp xếp tăng theo $auub$. Xét hai mục i và j, ta có $i \prec j$ nếu $auub(i) < auub(j)$. Trường hợp $auub(i) = auub(j)$ thì dựa trên thứ tự Alphabet của i và j.

Dựa vào giá trị $auub$ (Bảng 3), ta có thứ tự toàn phần của các mục trong D là: $d \prec f \prec e \prec a \prec b \prec c$. Bảng 4 trình bày cơ sở dữ liệu D được tổ chức lại sau khi loại bỏ mục d đồng thời sắp xếp các giao dịch theo thứ tự toàn phần.

Bảng 4. Cơ sở dữ liệu giao dịch D được tổ chức lại

TID	Mục	Độ hữu ích mục
1	e, a, c	6, 12, 10
2	f, e, a, b, c	2, 30, 21, 4, 35
3	f, a, b	2, 3, 4
4	f, e, a, b	2, 6, 12, 7
5	e, b, c	12, 2, 5
6	a, b, c	24, 3, 40
7	e, c	6, 15

Định nghĩa 7. Một mục i được gọi là sau tập mục X trong giao dịch T (ký hiệu là $X \prec i$) nếu $\forall j \in X, j \prec i$. Độ hữu ích lớn nhất của các mục sau tập mục X trong giao dịch T ký hiệu là $mu(X,T)$ được định nghĩa:

$$mu(X,T) = max(\{iu(i,T) \mid i \in T \wedge X \prec i\}) \quad (5)$$

Ví dụ: Xét tập mục $\{f,a\}$ thuộc giao dịch T_2 trong cơ sở dữ liệu D (Bảng 4), ta có $\{f,a\} \prec b$ và $\{f,a\} \prec c$. Độ hữu ích lớn nhất của các mục sau $\{f,a\}$ trong T_2 là $mu(\{f,a\},T_2) = max(\{iu(b,T_2), iu(c,T_2)\}) = max(4,35) = 35$.

IV. THUẬT TOÁN HAU-MINER

Chúng tôi đề xuất thuật toán khai thác tập hữu ích trung bình cao cải tiến HAU-Miner, với cấu trúc dữ liệu RAU-List biểu diễn cho mỗi tập mục trong quá trình khai thác HAUI. Mỗi danh sách RAU-List của tập mục X có cấu trúc gồm tên tập mục và 4 trường thông tin như: TID: số thứ tự giao dịch có chứa tập mục X, iu: độ hữu ích của tập mục X trong giao dịch, mu: độ hữu ích lớn nhất trong số các mục còn lại sau X, ri: số mục còn lại sau X.

Xây dựng danh sách RAU-List 1 phần tử:

Mỗi mục i chứa trong cơ sở dữ liệu D (Bảng 4) được xây dựng thành một danh sách RAU-List một phần tử tương ứng. Quá trình tạo danh sách RAU-List chứa các tập 1 phần tử được thực hiện như sau:

Xét lần lượt từng giao dịch trong cơ sở dữ liệu D, mỗi mục trong giao dịch sẽ được khởi tạo một danh sách RAU-List tương ứng nếu danh sách này chưa được khởi tạo trước đó. Trường hợp danh sách RAU-List của mục đang xét đã được khởi tạo trước đó thì chỉ cần thêm bộ mới vào danh sách. Xét giao dịch T_1 có 3 mục là e, a, c, danh sách RAU-List của mục e được khởi tạo và 1 bộ $e.T_1$ được chèn vào danh sách. Độ hữu ích của e trong T_1 là 6 ($e.T_1.iu = 6$). Số mục còn lại sau e là a và c với độ hữu ích tương ứng là 12 và 10, ta có độ hữu ích lớn nhất của các mục sau e là 12 ($e.T_1.mu = 12$) và số mục còn lại sau e là 2 ($e.T_1.ri = 2$). Xét mục kế tiếp trong giao dịch T_1 là a, một bộ mới là $a.T_1$ được chèn vào RAU-List a. Tương tự như mục e đã tính ở trên, ta cũng tính được $a.T_1.iu = 12$, $a.T_1.ri = 1$ và $a.T_1.mu = 10$. Tiếp tục xử lý mục còn lại trong giao dịch T_1 là c, ta có $c.T_1.iu = 10$, $c.T_1.mu = 0$ và $c.T_1.ri = 0$. Kết quả sau khi xử lý giao dịch T_1 được thể hiện trong Hình 1. Tiếp tục xử lý tương tự với các giao dịch tiếp theo, ta cũng tính được các giá trị iu, mu, ri của từng mục trong từng giao dịch tương ứng. Hình 2 trình bày kết quả sau khi xử lý tất cả các giao dịch của cơ sở dữ liệu D.

Tập mục	TID	iu	mu	ri
e	1	6	12	2

Tập mục	TID	iu	mu	ri
a	1	12	10	1

Tập mục	TID	iu	mu	ri
c	1	10	0	0

Hình 1. Danh sách RAU-List sau khi xử lý xong giao dịch T_1

Tập mục	TID	iu	mu	ri
f	2	2	35	4
	3	2	4	2
	4	2	12	3

Tập mục	TID	iu	mu	ri
e	1	6	12	2
	2	30	35	3
	4	6	12	2
	5	12	5	2
	7	6	15	1

Tập mục	TID	iu	mu	ri
a	1	12	10	1
	2	21	35	2
	3	3	4	1
	4	12	7	1
	6	24	40	2

Tập mục	TID	iu	mu	ri
b	2	4	35	1
	3	4	0	0
	4	7	0	0
	5	2	5	1
	6	3	40	1

Tập mục	TID	iu	mu	ri
c	1	10	0	0
	2	35	0	0
	5	5	0	0
	6	40	0	0
	7	15	0	0

Hình 2. Danh sách các tập RAU-List 1 phần tử

Tập danh sách RAU-List 1 phần tử được sử dụng để khai thác HAUI bằng giải thuật đệ quy mà không cần quét lại cơ sở dữ liệu. Trong quá trình mở rộng tập mục, các danh sách RAU-List sẽ được kết hợp với nhau để tạo ra tập danh sách RAU-List mở rộng theo thứ tự toàn phần (Định nghĩa 6). Ví dụ, trong Hình 2, RAU-List f được kết hợp với các RAU-List e, a, b và c, RAU-List e được kết hợp với RAU-List a, b và c, RAU-List a kết hợp với các RAU-List b, c, RAU-List b kết hợp với RAU-List c.

Việc kết hợp các *RAU-List* là chiến lược mở rộng tập để khai thác tất cả các HAUI có trong cơ sở dữ liệu giao dịch. Giải thuật tối ưu cần loại bỏ sớm các RAU-List dư thừa, có nghĩa là nếu việc kết hợp các tập có ít phần tử để tạo ra các tập nhiều phần tử hơn không hình thành nên tập hữu ích trung bình cao, thì chúng cần phải được loại bỏ trước khi kết hợp để làm giảm không gian tìm kiếm. Theo đó, để xác định một tập có thể mở rộng thành tập hữu ích trung bình cao hay không, chúng tôi áp dụng chiến lược tỉa cải tiến bằng cách sử dụng giá trị độ hữu ích trung bình lớn nhất (Maximum Average Utility). Giá trị này được trình bày trong Định nghĩa 8.

Định nghĩa 8. Độ hữu ích trung bình lớn nhất của X trong giao dịch T, ký hiệu $maxAU(X,T)$, được định nghĩa:

$$maxAU(X,T) = \begin{cases} \dfrac{X.T.iu + X.T.mu * X.T.ri}{l(X) + X.T.ri}, nếu\ X.T.mu > X.T.iu/l(X) \\ \dfrac{X.T.iu + X.T.mu}{l(X) + 1}, nếu\ 0 < X.T.mu \le X.T.iu/l(X) \\ 0, nếu\ X.T.mu = 0 \end{cases} \quad (6)$$

Giá trị độ hữu ích trung bình lớn nhất được đề xuất bởi Yun và cộng sự, ứng dụng trong thuật toán MHAI [12] để xác định một tập có bị loại bỏ hay không trong quá trình khai thác HAUI. Thuật toán MHAI sử dụng giá trị mn (Maximum Number of Remaining Items) để tính giá trị độ hữu ích trung bình lớn nhất của tập mục X trong giao dịch T. Giá trị mn được xác định là số phần tử còn lại lớn nhất sau X trong các giao dịch. Giá trị này sẽ áp dụng cho tất cả các giao dịch có trong cơ sở dữ liệu trong trường hợp $X.T.mu > X.T.iu/l(X)$ để tính giá trị độ hữu ích trung bình lớn nhất. Tuy nhiên trong các giao dịch chứa X sẽ có nhiều giáo dịch có số phần tử còn lại sau X nhỏ hơn mn. Do đó việc sử dụng giá trị mn để tính độ hữu ích trung bình lớn nhất là chưa tối ưu. Để giải quyết vấn đề này, chúng tôi sử dụng giá trị ri là số phần tử còn lại thực sự sau tập mục X có trong mỗi giao dịch, do $ri \le mn$ nên giá trị độ hữu ích trung bình lớn nhất theo công thức cải tiến (maxAU) luôn nhỏ hơn hoặc bằng giá trị trung bình lớn nhất (MAU) trong thuật toán MHAI. Nghĩa là ngưỡng chặn trên giá trị trung bình lớn nhất sẽ giảm dẫn đến số ứng viên bị loại bỏ nhiều hơn, từ đó làm tăng hiệu suất thực thi của thuật toán.

Độ hữu ích trung bình lớn nhất của X trong cơ sở dữ liệu giao dịch D, ký hiệu $maxAU(X)$, là tổng của các độ hữu ích trung bình lớn nhất trong các giao dịch:

$$maxAU(\mathrm{X}) = \sum_{X \subseteq T \in D} maxAU(X,T) \quad (7)$$

Chiến lược tỉa 2 (MAU-Prune). Nếu độ hữu ích trung bình lớn nhất của X nhỏ hơn minUtil thì mọi tập mở rộng từ X không phải là $HAUI$.

Trong Hình 2, độ hữu ích trung bình lớn nhất của f được tính như sau: f thuộc 3 giao dịch là T_2, T_3, T_4. Vì f.T_2.iu /1 < f.T_2.mu nên theo định nghĩa 8,

maxAU(f, T_2) = (2+35*4)/(1+4) = 28.4; tiếp tục vì f.T_3.iu /1 < f.T_3.mu nên maxAU(f, T_3) = (2+4*2)/(1+2) = 3.33; tương tự maxAU(f, T_4) = (2+12*3)/(1+3) = 9.5. Do vậy, maxAU(f) = maxAU(f, T_2) + maxAU(f, T_3) + maxAU(f, T_4) = 41.2. Vì maxAU < minUtil là 42 nên f bị loại bỏ theo chiến lược tỉa 2. Độ hữu ích trung bình của f được tính như sau: au(f) = (2 + 2 + 2)/1 = 6. Vì au(f) < minUtil nên f không phải là tập hữu ích trung bình cao.

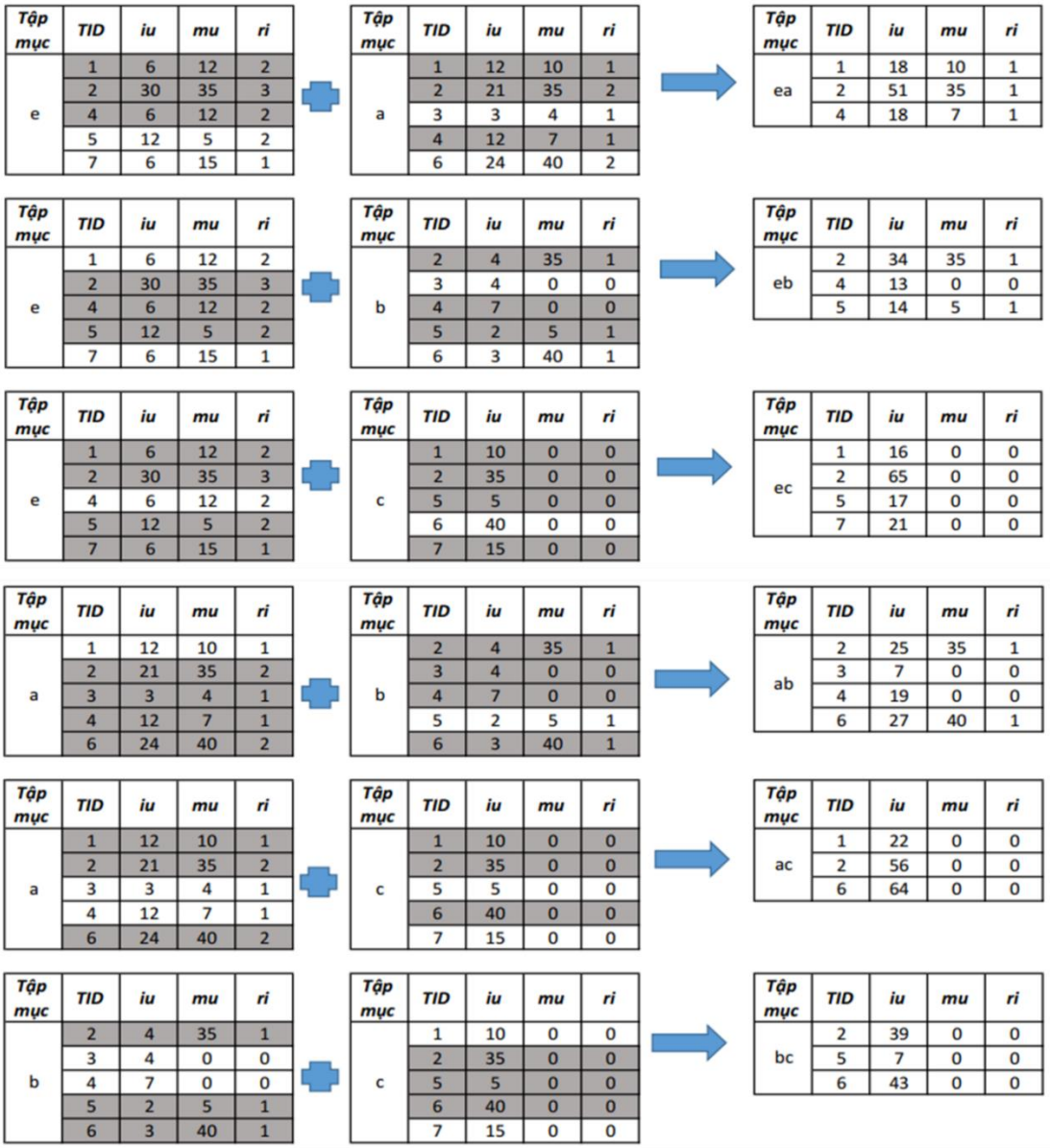
Tiếp theo, tính maxAU(e). Vì e.T_1.iu/1 < e.T_1.mu nên
maxAU(e, T_1) = (6+12*2)/(1+2)=10;
maxAU(e, T_2) = (30+35*3)/(1+3)=33.75;
maxAU(e, T_4) = (6+12*2)/(1+2)=10;
maxAU(e, T_7) = (6+15*1)/(1+1)=10.5;
vì e.T_5.iu > e.T_5.mu nên maxAU(e, T_5) = (12+5)/(1+1)=8.5. maxAU(e)= 10 + 33.75 + 10 + 8.5 + 10.5 = 72.75. Vì maxAU(e) > minUtil nên e có thể mở rộng để tiếp tục khai thác HAUI. au(e) =

(6+30+6+12+6)/1 = 60 > minUtil nên e là tập hữu ích trung bình cao. Theo cách tính tương tự, ta tính được maxAU(a)=86.66, au(a)=72; maxAU(b)=44.5, au(b)=20; maxAU(c)=0, au(c)=105. Như vậy, sau khi tính toán độ hữu ích trung bình của tất cả các tập mục 1 phần tử, ta có các tập hữu ích trung bình cao gồm: e, a, c. Sau khi tính toán độ hữu ích trung bình lớn nhất của tất cả các tập 1 phần tử, ta có các tập mục có thể mở rộng gồm: e, a, b, và các tập bị tỉa gồm: f và c.

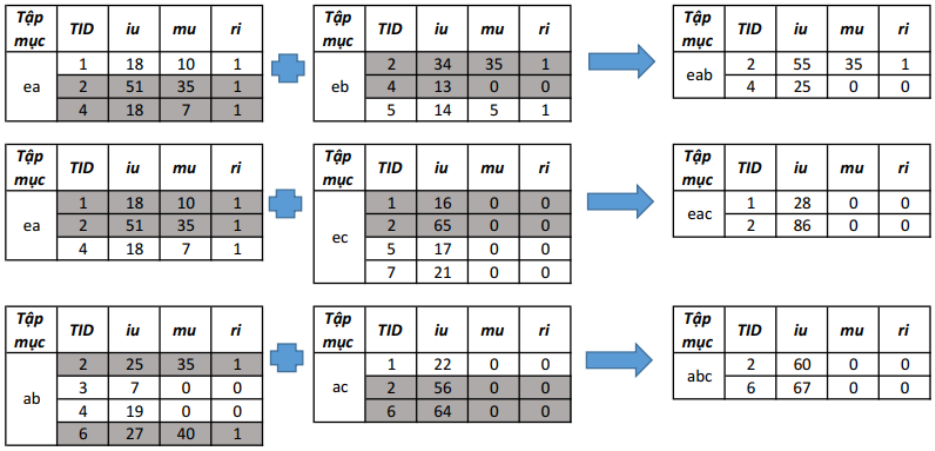
Xây dựng danh sách RAU-List 2 phần tử:

Danh sách RAU-List 2 phần tử được tạo từ danh sách RAU-List 1 phần tử theo thứ tự toàn phần (Định nghĩa 6). Vì mục f có $maxAU(f) < minUtil$ nên dừng mở rộng với f. Do đó danh sách mở rộng bắt đầu từ mục e. Ví dụ, Hình 2, e có thể kết hợp với các mục a, b, c để tạo thành các tập: ea, eb và ec. Khi e kết hợp với a, thì các giao dịch chung của chúng bao gồm: T_1, T_2 và T_4 sẽ được tính để đưa vào danh sách RAU-List ea. Các trường giá trị ri, iu và mu khi đó được tính theo các công thức sau: Giả sử X kết hợp với Y thành XY, khi đó: $XY.T.iu = X.T.iu + Y.T.iu - P.T.iu$ (với P tập mục với vai trò là tiền tố của hai tập mục X và Y, nếu P là rỗng thì $P.T.iu = 0$). Giá trị độ hữu ích lớn nhất của các mục sau XY cũng chính là độ hữu ích lớn nhất các muc sau Y, ta có $XY.T.mu = Y.T.mu$. Tương tự, số mục còn lại sau XY cũng chính là số mục còn lại sau Y, ta có $XY.T.ri = Y.T.ri$.

Tập mục e được kết hợp với a, b và c. Các giá trị ri, iu và mu của tập ea được tính như sau: e và a đều thuộc các giao dịch T_1, T_2 và T_4, nên ea.T_1.ri = a.T_1.ri = 1; ea.T_1.mu = a.T_1.mu = 10; ea.T_1.iu = e.T_1.iu + a.T_1.iu = 6+12 =18; ea.T_2.ri = a.T_2.ri = 1; ea.T_2.mu = a.T_2.mu = 35; ea.T_2.iu = e.T_2.iu + a.T_2.iu = 30+21 =51; ea.T_4.ri = a.T_4.ri = 1; ea.T_4.mu = a.T_4.mu = 7; ea.T_4.iu = e.T_4.iu + a.T_4.iu = 6+12 =18. Tương tự cho các tập kết hợp 2 phần tử còn lại, ta được danh sách kết hợp các tập *RAU-List* 2 phần tử được trình bày như trong Hình 3.



Hình 3. Danh sách RAU-List 2 phần tử



Hình 4. Danh sách *RAU-List* 3 phần tử

Từ kết quả ở hình 3 và theo Định nghĩa 2, ta tính được độ hữu ích trung bình của tập ea là au(ea) = (18+51+18)/2 = 43.5; tính tương tự cho các tập eb, ec, ab, ac và bc, kết quả được trình bày trong hình 4. Cũng từ kết quả ở hình 3 và theo định nghĩa 10, ta tính được độ hữu ích trung bình lớn nhất của tập ea là maxAU(ea) = maxAU(ea, T_1) + maxAU(ea, T_2) + max(ea, T_4) = (18+10*1)/(2+1) + (51+35*1)/(2+1) + (18+7)/(2+1) = 46.3; tính tương tự cho các tập eb, ec, ab, ac và bc. Với các tập có $mu = 0$ thì maxAU của chúng cũng bằng 0, kết quả được trình bày trong Bảng 5. Trong Bảng 5, các tập: ea, ec, ac và bc là các tập hữu ích trung bình cao vì chúng có $au > minUtil = 42$. Các tập được mở rộng bao gồm: ea và ab

Bảng 5. Độ hữu ích trung bình và độ hữu ích trung bình lớn nhất của các tập 2 phần tử

Tập mục	au	maxAU
ea	43.5	46.3
eb	30.5	29.3
ec	59.5	0
ab	39	42.3
ac	71	0
bc	44.5	0

Xây dựng danh sách RAU-List 3 phần tử:

Các tập ea và ab tiếp tục được mở rộng theo thứ tự đã được trình bày ở trên, kết quả sinh ra các tập ứng viên gồm: eab, eac và abc.

Tập eab được tạo ra từ việc kết hợp của 2 tập ea và eb. Hai tập này đều thuộc các giao dịch chung gồm T_2 và T_4. Các giá trị mu và ri của các RAU-List 3 phần tử được tính tương tự như RAU-List 2 phần tử. Ta có: eab.T_2.ri = eb.T_2.ri = 1, eab.T_2.mu = eb.T_2.mu = 35. Khi kết hợp ea và eb trên một giao dịch, nếu lấy tổng giá trị iu của ea và eb trên giao dịch đó thì thừa lượng giá trị iu của e, do đó khi kết hợp cần trừ đi một lượng iu của e trên giao dịch. Giá trị iu của tập kết hợp eab được tính: eab.T_2.iu = ea.T_2.iu + eb.T_2.iu – e.T_2.iu = 51 + 34 – 30 = 55. Với giao dịch T_4, ta có eab.T_4.ri = eb.T_4.ri = 0; eab.T_4.mu = eb.T_4.mu = 0; eab.T_4.iu = ea.T_4.iu + eb.T_4.iu – e.T_4.iu = 18 + 13 – 6 = 25. Tính tương tự cho các tập eac và abc, kết quả được trình bày trong Bảng 6.

Từ kết quả trong Bảng 6, độ hữu ích trung bình của tập eab được tính như sau: $au(eab) = (55 + 25)/3 = 26.7$, tính tương tự cho các tập ebc và abc. Theo Định nghĩa 8, độ hữu ích trung bình lớn nhất của tập eab được tính là $maxAU(eab) = maxAU(eab, T_2) + maxAU(eab, T_4) = (55 + 35 * 1)/(3 + 1) + 0 = 22.5$. Các tập ebc và abc đều có $mu = 0$ nên maxAU của chúng bằng 0. Kết quả trong Bảng 6 cho thấy chỉ có tập abc là tập hữu ích trung bình cao vì $au(abc) > minUtil = 42$. Không còn tập nào được mở rộng nữa, do đó quá trình đệ quy để khai thác kết thúc. Bảng 7 trình bày kết quả cuối cùng bao gồm tất cả các tập hữu ích trung bình cao (HAUI) khai thác trên cơ sở dữ liệu giao dịch D.

Bảng 6. Độ hữu ích trung bình và độ hữu ích trung bình lớn nhất của tập 3 phần tử

Tập mục	au	maxAU
eab	26.7	22.5
ebc	38	0
abc	42.3	0

Bảng 7. Các tập HAUI trên cơ sở dữ liệu D

Tập mục	au
e	60
a	72
c	105
ea	43.5
ec	59.5
ac	71
bc	44.5
abc	42.3

Thuật toán HAU-Miner

Thuật toán *HAU-Miner* với dữ liệu đầu vào là cơ sở dữ liệu giao dịch D và ngưỡng độ hữu ích tối thiểu minUtil. Kết quả thuật toán là các tập mục có độ hữu ích trung bình cao. Khởi đầu thuật toán quét cơ sở dữ liệu giao dịch D để tính chặn trên ngưỡng độ hữu ích trung bình (auub) cho từng mục có trong cơ sở dữ liệu (dòng 1). Tại dòng 2, thực hiện kiểm tra nếu $auub(i) \geq minUtil$ thì đưa mục i vào tập I^*, ngược lại loại mục i khỏi cơ sở dữ liệu D. Tại dòng 3, thực hiện sắp xếp các mục trong tập I^* tăng theo giá trị $auub$ và sắp xếp các mục trong các giao dịch của cơ sở dữ liệu D tăng theo I^*. Tiếp theo quét cơ sở dữ liệu D lần 2 để tạo danh sách *RAU-List* cho mỗi phần tử $i \notin I^*$ ở dòng 4 và khởi tạo cấu trúc *EUCS* để chuẩn bị áp dụng chiến lược tỉa này trong thuật toán kế tiếp. Cuối cùng gọi thực hiện thuật toán 2 là *MiningHAU* để khai thác tập $HAUI$.

Thuật toán 1: (Thuật toán chính - HAU-Miner)

Vào: D: Cơ sở dữ liệu giao dịch, $minUtil$: Ngưỡng độ hữu ích tối thiểu.
Ra: Các tập mục độ hữu ích trung bình cao
1. Quét cơ sở dữ liệu D để tính $auub(i)$ cho mỗi mục i có trong I.
2. Tính $I^* = \{i \in I \mid auub(i) \geq minUtil\}$ và loại bỏ các mục $i \notin I^*$ khỏi cơ sở dữ liệu D.
3. Sắp xếp I^* tăng theo $auub$, sắp xếp các mục trong D theo thứ tự của I^*.
4. Quét cơ sở dữ liệu D để tạo danh sách *RAU-List* cho mỗi phần tử $i \notin I^*$
5. Khởi tạo cấu trúc $EUCS$
6. *MiningHAUI(∅, RAU-List, minUtil, EUCS)*

Chiến lược tỉa 3 (EUCS-Prune) [8]**:** Cấu trúc EUCS áp dụng trong thuật toán HAU-Miner sử dụng giá trị $auub$ để kiểm tra khả năng mở rộng của tập mục. Mỗi phần tử trong cấu trúc EUCS được gán bởi giá trị $auub$ tương ứng với tập mục đang xét. Xét hai tập mục X và Y, ta có $EUCS(X,Y) = auub(XY)$. Nếu $EUCS(X,Y) < minUtil$ thì tập mục XY không phải là HAUI và mọi tập mở rộng từ XY cũng không phải HAUI. Khi đó dừng mở rộng với tập mục XY.

Thuật toán *MiningHAUI* được thiết kế thực hiện đệ quy để tìm ra tập mục có độ hữu ích trung bình cao. Đầu tiên, thuật toán duyệt qua từng phần tử *RAU-List X* có trong danh sách *RLs* ở dòng 1. Tại dòng 2 và 3, kiểm tra nếu $au(X) \geq minUtil$ thì X là một tập có độ hữu ích trung bình cao và đưa X vào tập kết quả $HAUIs$. Dòng 5 kiểm tra nếu $maxAU(X) \geq minUtil$ thì những tập mở rộng từ X có khả năng là $HAUI$. Khi đó khởi tạo tập các *RAU-List* mở rộng từ X ($exRLs = \emptyset$), duyệt qua các *RAU-List Y* sau X thuộc danh sách *RLs* để áp dùng Chiến lược tỉa 3 (EUCS-Prune) với cấu trúc EUCS ở dòng 8. Nếu thỏa điều kiện $EUCS(X,Y) \geq minUtil$ thì gọi hàm HAUConstruct để kết hợp 2 *RAU-List X* và Y thành *RAU-List* mở rộng XY. Kết quả thực hiện từ dòng 7 đến 11 được danh sách *RAU-List* mở rộng là $exRLs$. Tại dòng 12 thực hiện gọi đệ quy thuật toán $MiningHAUI$ với danh sách *RAU-List* mở rộng vừa tìm được.

Thuật toán 2: $MiningHAUI$

Vào: P: *RAU-List* với vai trò là tiền tố; RLs: Danh sách các *RAU-List* có tiền tố là *RAU-List* P, $minUtil$: Ngưỡng độ hữu ích tối thiểu, $EUCS$: Cấu trúc EUCS
Ra: Các tập mục độ hữu ích trung bình cao.
1. **for** each $X \in RLs$ **do**
2. **if** $au(X) \geq minUtil$ **then**
3. $HAUIs \leftarrow X$
4. **end if**
5. **if** $(maxAU(X) \geq minUtil)$ //MAU-Prune
6. $exRLs = \emptyset$ //Khởi tạo danh sách *RAU-List* mở rộng từ X
7. **for** each Y in $RLs \wedge X \prec Y$ **do**
8. **if** $EUCS(X,Y) \geq minUtil$ **then** //Áp dụng chiến lược tỉa EUCP
9. $exRLs \leftarrow HAU\text{Construct}(P, X, Y)$;
10. **end if**
11. **end for**
12. $MiningHAUI(X, exRLs, minUtil, EUCS)$;
13. **end if**
14. **end for**

Chiến lược tỉa 4 (MLA-Prune) [9] **:** Xét 2 tập X và Y, nếu $maxAU(X) - \sum_{X \subseteq T_j \in D \wedge Y \not\subset T_j} maxAU(X, T_j) < minUtil$ thì tập mục XY không phải là HAUI và mọi tập mở rộng từ XY cũng không phải là HAUI. Khi đó dừng mở rộng với tập XY.

Thuật toán *HAUConstruct* thực hiện kết hợp 2 *RAU-List* Px và Py thành *RAU-List* mở rộng Pxy. Dòng 1 khởi tạo giá trị ban đầu cho MLA bằng $maxAU(Px)$. Từ dòng 2 đến dòng 16, thuật toán duyệt qua từng bộ ex trong Px và kiểm tra có tồn tại bộ ey thuộc Py sao cho $ex.TID = ey.TID$ (dòng 3). Nếu có, thì một bộ mới exy được tạo ra dựa vào một trong hai trường hợp: Trường hợp *RAU-List* tiền tố $P \neq \emptyset$ thì *RAU-List* Pxy được tạo có từ 3 mục trở lên. Ngược lại thi Pxy có 2 mục. Dòng 10 chèn bộ exy vào danh sách Pxy. Từ dòng 11 đến 16, áp dụng chiến lược tỉa 4 (MLA-Prune) khi không tìm thấy bộ ey nào trong Py có cùng TID với ex. Khi đó giá trị MLA sẽ được thiết lập giảm: $MLA = MLA - maxAU(ex)$ Dòng 13, 14 kiểm tra nếu $MLA < minUtil$ thì loại bỏ so sớm phần tử không phải *HAUI.* Dòng 18 trả về kết quả *RAU-List XY* được kết hợp từ hai *RAU-List X* và Y.

Thuật toán 3: ($HAUConstruct$)

Vào: P: *RAU-List* với vai trò là tiền tố; Px, Py: Hai *RAU-List* cần kết hợp; $minUtil$: Ngưỡng độ hữu ích tối thiểu.
Ra: Pxy: *RAU-List* sau khi kết hợp Px và Py.
1. $Set\ MLA = maxAU(Px)$;
2. **for** each $ex \in Px$ **then** //*duyệt qua từng bộ ex trong RAU-List Px*
3. **if** $\exists\ ey \in Py \wedge ex.TID = ey.TID$ **then**
4. **if** $P \neq \emptyset$ **then**
5. Tìm bộ $e \in P$ sao cho $e.TID = ex.TID$;
6. $exy =< ex.TID,\ ex.iu + ey.iu - e.iu, ey.mu, ey.ri >$;
7. **else**
8. $exy =< ex.TID,\ ex.iu + ey.iu,\ ey.mu, ey.ri >$;
9. **end if**
10. $Pxy \leftarrow exy$;
11. **else**
12. $MLA = MLA - maxAU(ex)$;
13. **if** $MLA < minUtil$ **then** // áp dụng chiến lược tỉa MLA-Prune
14. return null;
15. **end if**
16. **end if**
17. **end for**
18. return Pxy;

V. THỰC NGHIỆM

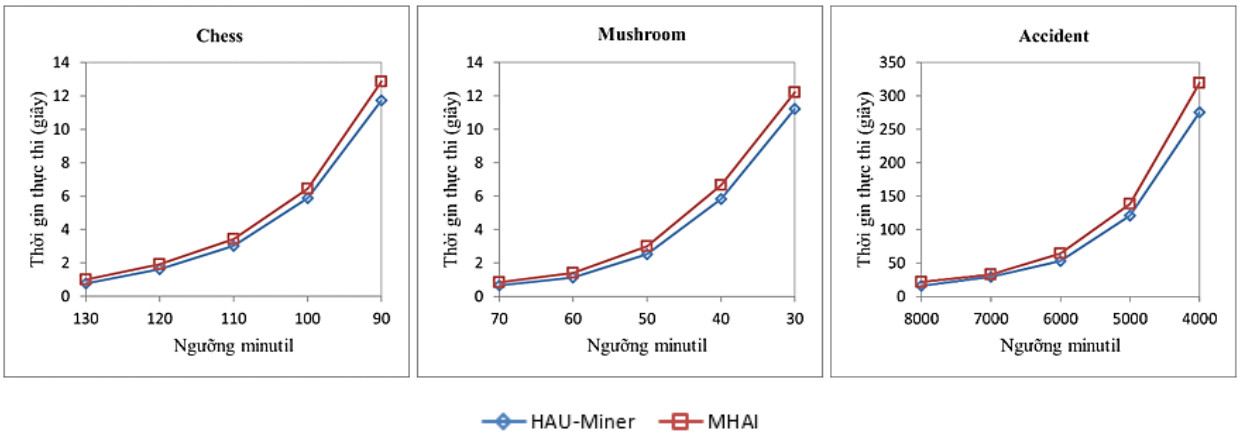
Thuật toán HAU-Miner được cài đặt bằng ngôn ngữ lập trình Java, chạy thử nghiệm trên máy tính Dell Precision Tower 3620, Intel Core i7-7800X CPU @3.5GHz, bộ nhớ RAM 32GB trên hệ điều hành Windows 10. Các cơ sở dữ liệu thử nghiệm được tải từ thư viện SPMF [28] là các cơ sở dữ liệu giao dịch bao gồm: Chess, Mushroom, Accidents, Retail, Kosarak Chainstore. Trong đó cơ sở dữ liệu Chess có độ dày cao nhất là 49.3333, kế đến là cơ sở dữ liệu Mushroom và Accident có độ dày vừa lần lượt là 19.3277 và 7.2222. Các cơ sở dữ liệu còn lại thuộc loại cơ sở dữ liệu thưa gồm Retail, Kosarak và Chainstore. Xét về độ lớn thì cơ sở dữ liệu lớn nhất là Chainstore với 1,112,949 giao dịch, kế đến là cơ sở dữ liệu Kosarak và Accident, các cơ sở dữ

liệu còn lại có số lượng giao dịch nhỏ hơn. Chi tiết các cơ sở dữ liệu trình bày trong Bảng 8. Thực nghiệm của thuật toán HAU-Miner được so sánh với thuật toán MHAI [12]. Kết quả thực nghiệm được đánh giá dựa trên thời gian thực thi và dung lượng bộ nhớ sử dụng.

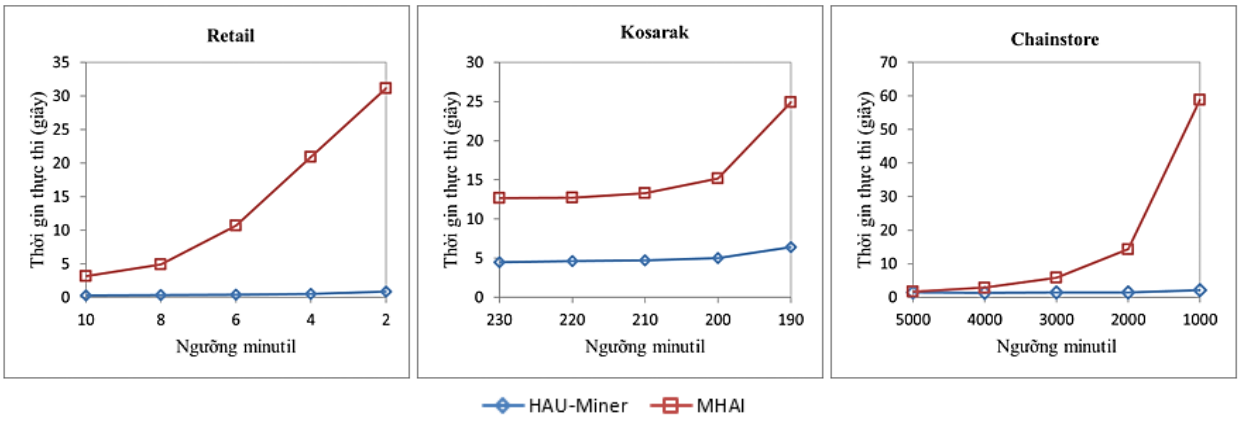
Hình 5, 6, 7 và 8 trình bày kết quả thực nghiệm so sánh giữa thuật toán HAU-Miner và thuật toán MHAI về thời gian thực thi và bộ nhớ sử dụng. Kết quả thực nghiệm cho thấy thuật toán HAU-Miner có thời gian thực thi hiệu quả hơn thuật toán MHAI trên tất cả các cơ sở dữ liệu từ rất dày như Chess đến cơ sở dữ liệu rất thưa như Chainstore. Tuy nhiên bộ nhớ sử dụng của thuật toán HAU-Miner chỉ có hiệu quả hơn thuật toán MHAI ở cơ sở dữ liệu thưa.

Bảng 8. Đặc điểm các cơ sở dữ liệu thực nghiệm

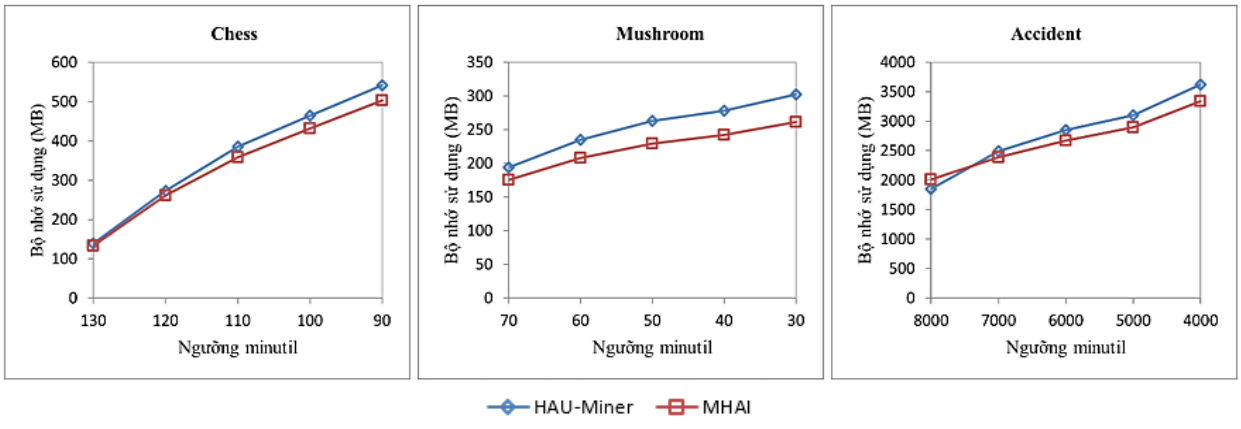
Cơ sở dữ liệu	Tổng số giao dịch	Số mục (I)	Độ dài trung bình giao dịch (A)	Độ dày (A/I) %
Chess	3,196	75	37	49.3333
Mushroom	8,124	119	23	19.3277
Accidents	340,183	468	33.8	7.2222
Retail	88,162	16,470	10.3	0.0625
Kosarak	990,002	41,270	8.1	0.0196
Chainstore	1,112,949	46,086	7.3	0.0158



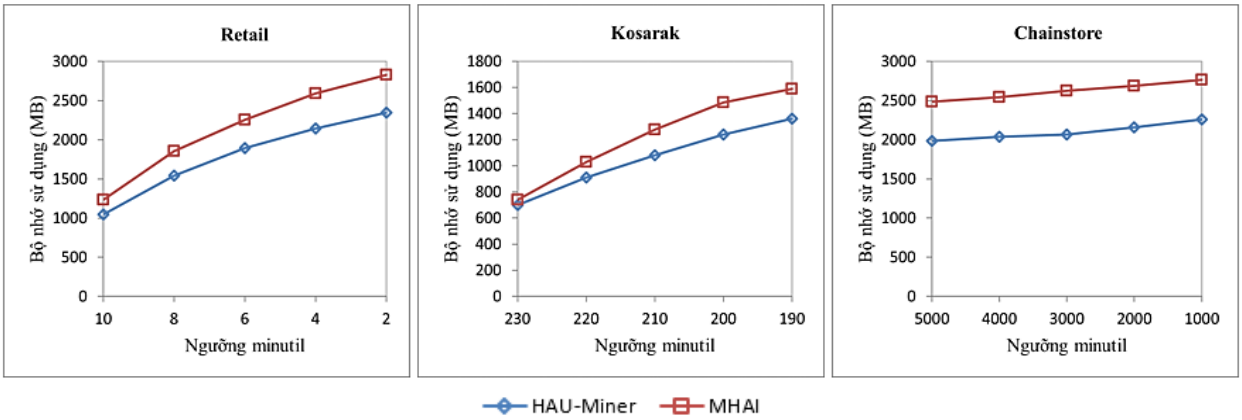
Hình 5. So sánh thời gian thực thi trên cơ sở dữ liệu dày



Hình 6. *So sánh thời gian thực thi trên cơ sở dữ liệu thưa*



Hình 7. *So sánh bộ nhớ sử dụng trên cơ sở dữ liệu dày*



Hình 8. *So sánh bộ nhớ sử dụng trên cơ sở dữ liệu thưa*

Hình 5 so sánh thời gian thực thi của 2 thuật toán trên cơ sở dữ liệu dày. Với cơ sở dữ liệu rất dày là Chess, khi ngưỡng minUtil là 130,000, thời gian chạy của thuật toán HAU-Miner là 0.77s; còn thời gian chạy của thuật toán MHAI là 0.993s không chênh lệch nhiều so với thuật toán HAU-Miner. Khi ngưỡng minUtil là 90,000, thì thời gian chạy của thuật toán HAU-Miner cũng nhanh hơn so với MHAI nhưng không đáng kể. Với cơ sở dữ liệu dày trung bình là Mushroom và Accident, kết quả cũng cho thấy thời gian thực thi của thuật toán HAU-Miner nhanh hơn thời gian thực thi của thuật toán MHAI.

Hình 6 so sánh thời gian thực thi của 2 thuật toán trên các cơ sở dữ liệu thưa. Từ cơ sở dữ liệu thưa ít là Retail cho đến cơ sở dữ liệu rất thưa là Chainstore, kết quả cho thấy thời gian thực thi của thuật toán HAU-Miner nhanh hơn đáng kể so với thuật toán MHAI. Cụ thể, với cơ sở dữ liệu Retail, tại ngưỡng minUtil=10,000, thời gian thực thi của HAU-Miner nhanh khoảng 10 lần so với thuật toán MHAI. Khi ngưỡng minUtil giảm xuống còn 2,000 thì thời gian thực thi của thuật toán HAU-Miner càng nhanh hơn vượt trội so với thuật toán MHAI khoảng 37 lần. Với cơ sở dữ liệu thưa vừa là Kosarak, tại ngưỡng minUtil=230,000, thời gian thực thi của HAU-Miner nhanh hơn của MHAI khoảng 2,8 lần, khi ngưỡng minUtil=190,000, thời gian thực thi của HAU-Miner nhanh hơn MHAI tới 3,8 lần. Với cơ sở dữ liệu rất thưa Chainstore, tại ngưỡng minUtil=5,000,000, thời gian thực thi của HAU-Miner nhanh hơn của MHAI khoảng 1,19 lần, khi ngưỡng minUtil=1,000,000, thời gian thực thi của HAU-Miner nhanh hơn MHAI tới 28,1 lần. Kết quả này cho thấy các chiến lược tỉa sử dụng trong thuật toán HAU-Miner tỏ ra hiệu quả trên các cơ sở dữ liệu thưa.

Hình 7 và 8 lần lượt so sánh bộ nhớ sử dụng của 2 thuật toán trên các nhóm cơ sở dữ liệu dày và thưa. Với nhóm cơ sở dữ liệu dày như Chess, Mushroom và Accident, bộ nhớ sử dụng của thuật toán MHAI tốt hơn hơn thuật toán HAU-Miner nhưng không đáng kể. Với nhóm cơ sở dữ liệu thưa như Retail, Chainstore và Kosarak thì thuật toán HAU-Miner sử dụng bộ nhớ ít hơn thuật toán MHAI tại tất cả các ngưỡng thực nghiệm. Cụ thể, với cơ sở dữ liệu rất dày là Chess, tại ngưỡng minUtil=130,000, bộ nhớ sử dụng của thuật toán HAU-Miner là 139MB, còn thuật toán MHAI là 133MB. Với cơ sở dữ liệu dày Mushroom, thuật toán HAU-Miner sử dụng bộ nhớ nhiều hơn MHAI từ 19 đến 41MB tại các ngưỡng thực nghiệm. Cơ sở dữ liệu dày vừa Accident cũng cho kết quả tương tự. Ngược lại, với cơ sở dữ liệu thưa như Retail, tai ngưỡng minUtil=10,000, thuật toán MHAI sử dụng bộ nhớ là 1233MB trong khi thuật toán HAU-Miner sử dụng 1047MB, ít hơn 186MB. Khi ngưỡng minUtil giảm xuống còn 2,000 thì thuật toán HAU-Miner sử dụng bộ nhớ ít hơn 480MB. Với cơ sở dữ liệu Kosarak, thuật toán HAU-Miner cũng sử dụng bộ nhớ ít hơn thuật toán MHAI từ 40 đến 229MB trên các ngưỡng minUtil thực nghiệm. Trên cơ sở dữ liệu rất thưa Chainstore, thuật toán HAU-Miner cũng sử dụng bộ nhớ hiệu quả hơn thuật toán MHAI tại tất cả các ngưỡng. Kết quả này cho thấy khi ngưỡng minUtil càng giảm thì thuật toán HAU-Miner sử dụng bộ nhớ càng hiệu quả hơn thuật toán MHAI trên các cơ sở dữ liệu thưa. Điều này chứng tỏ việc cải tiến công thức tính giá trị maxAU cùng các chiến lược tỉa sử dụng trong thuật toán đã cắt giảm đáng kể không gian tìm kiếm làm để tăng hiệu quả tính toán và giảm bộ nhớ sử dụng.

Bảng 9. Số ứng viên phát sinh trên cơ sở dữ liệu Chess

MinUtil	130,000	120,000	110,000	100,000	90,000
MHAI	3,643	7,444	14,680	31,305	67,476
HAU-Miner	2,945	6,042	12,494	26,740	58,764

Bảng 10. Số ứng viên phát sinh trên cơ sở dữ liệu Mushroom

MinUtil	70,000	60,000	50,000	40,000	30,000
MHAI	4,061	7,709	18,128	43,946	96,151
HAU-Miner	2,335	4,684	12,829	34,094	75,108

Bảng 11. Số ứng viên phát sinh trên cơ sở dữ liệu Accident

MinUtil	8,000,000	7,000,000	6,000,000	5,000,000	4,000,000
MHAI	805	1,414	2,948	7,440	19,929
HAU-Miner	443	919	2,067	5,270	15,431

Bảng 12. Số ứng viên phát sinh trên cơ sở dữ liệu Retail

MinUtil	10,000	8,000	6,000	4,000	2,000
MHAI	119,505	302,158	722,077	2,100,233	8,095,358
HAU-Miner	1,208	1,744	2,787	5,136	12,518

Bảng 13. Số ứng viên phát sinh trên cơ sở dữ liệu Kosarak

MinUtil	230,000	220,000	210,000	200,000	190,000
MHAI	16,359	18,113	21,585	27,064	32,977
HAU-Miner	999	1,083	1,196	1,330	1,493

Bảng 14. Số ứng viên phát sinh trên cơ sở dữ liệu Accident

MinUtil	5,000,000	4,000,000	3,000,000	2,000,000	1,000,000
MHAI	355	1,436	5,389	22,081	232,086
HAU-Miner	119	169	241	415	1,167

Bảng 9 đến Bảng 14 trình bày chi tiết kết quả so sánh số lượng ứng viên phát sinh trong quá trình khai thác HAUI trên các cơ sở dữ liệu từ dày đến thưa. Mỗi ứng viên là một danh sách RAU-List phát sinh trong quá trình khai thác và mở rộng tập. Với cơ sở dữ liệu rất dày như Chess. Tại ngưỡng minUtil=130,000, thuật toán HAU-Miner phát sinh 2,945 ứng viên trong khi thuật toán MHAI phát sinh 3,643 ứng viên. Khi ngưỡng minUtil giảm xuống còn 90,000 thì thuật toán HAU-Miner phát sinh 58,764, thuật toán MHAI phát sinh đến 67,476, nhiều hơn 8,712 ứng viên. Với cơ sở dữ liệu Mushroom và Accident, thuật toán HAU-Miner cũng phát sinh số lượng ứng viên ít hơn thuật toán MHAI trung bình từ 1.2 đến 1.8 lần. Đặt biệt với cơ sở dữ liệu thưa như Retail, tại ngưỡng minUtil = 10,000, thuật toán HAU-Miner chỉ phát sinh 1,208 ứng viên, trong khi thuật toán MHAI phát sinh 119,505 ứng viên, nhiều gấp 98 lần so với thuật toán HAU-Miner. Khi giảm ngưỡng minUtil xuống còn 2000 thì thuật toán HAU-Miner càng hiệu quả với số ứng viên phát sinh chỉ là 12,518 trong khi thuật toán MHAI phát sinh số lượng ứng viên rất lớn lên đến 8,095,358, nhiều gấp 647 lần so với thuật toán HAU-Miner. Tương tự với cơ sở dữ liệu Kosarak và Chainstore, số ứng viên phát sinh của thuật toán HAU-Miner cũng ít hơn nhiều so với thuật toán MHAI ở tất cả các ngưỡng. Kết quả này chứng tỏ rằng việc cải tiến công thức tính giá trị maxAU kết hợp với các chiến lược tỉa EUCS, MLA-Prune, UB-Prune đã loại bỏ được nhiều ứng viên không phải là HAUI, từ đó cắt giảm đáng kể không gian tìm kiếm và mở rộng tập trong quá trình khai thác HAUI.

VI. KẾT LUẬN

Bài báo này đề xuất thuật toán *HAU-Miner* với cấu trúc dữ liệu *RAU-List* cải tiến từ cấu trúc *Utility-List* để khai thác tập hữu ích trung bình cao trên cơ sở dữ liệu giao dịch. Tối ưu giá trị độ hữu ích trung bình lớn nhất *(maxAU)* áp dụng trong chiến lược tỉa MAU-Prune để nâng cao hiệu suất thực thi. Ngoài ra, thuật toán còn áp dụng các chiến lược tỉa như *UB-Prune, EUCS, MLA-Prune* để cắt giảm không gian tìm kiếm một cách hiệu quả. Kết quả thực nghiệm cho thấy thuật toán đề xuất *HAU-Miner* có thời gian thực hiện nhanh hơn thuật toán *MHAI* trên tất cả các cơ sở dữ liệu được thử nghiệm. Bộ nhớ sử dụng của thuật toán *HAU-Miner* tốt hơn thuật toán *MHAI* ở các cơ sở dữ liệu thưa.

Hướng phát triển tiếp theo là nghiên cứu cải tiến thuật toán để thực nghiệm khai thác hiệu quả tập hữu ích trung bình cao trên cơ sở dữ liệu tăng trưởng.

TÀI LIỆU THAM KHẢO

[1] R. Agrawal, T. Imielinski and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," *ACM SIGMOD Record*, p. 207–216, 1993.

[2] M.-S. Chen, J. Han and P. S. Yu, "Data mining: an overview from a database perspective," *IEEE TRANSACTIONS ON KNOWLEDGEAND DATA ENGIJEERING*, vol. 8, no. 6, p. 866–883, 1996.

[3] C.-W. Lin, T.-P. Hong and W.-H. Lu, "The Pre-FUFP algorithm for incremental mining," *Expert Systems with Applications*, vol. 36, p. 9498–9505, 2009.

[4] J. HAN, J. PEI, Y. YIN and R. MAO, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach," *Data Mining and Knowledge Discovery*, vol. 8, p. 53–87, 2004.

[5] M. Liu and J. Qu, "Mining High Utility Itemsets without Candidate Generation," *ACM International Conference on Information and Knowledge Management*, p. 55–64, 2012.

[6] G.-C. Lan, T.-P. Hong and V. S. Tseng, "An efficient projection-based indexing approach for mining high utility itemsets," *Knowl. Inform. Syst.*, vol. 8, p. 85–107, 2013.

[7] H. Yao, H. J. Hamilton and C. J. Butz, "A Foundational Approach to Mining Itemset Utilities from Databases," *SIAM International Conference on Data Mining*, p. 215–221, 2004.

[8] P. Fournier-Viger, C.-W. Wu, S. Zida and V. S. Tseng, "FHM: Faster High-Utility Itemset Mining Using Estimated Utility Co-occurrence Pruning," *Foundations of Intelligent Systems*, vol. 8502, pp. 83-92, 2014.

[9] S. Krishnamoorthy, "Pruning strategies for mining high utility itemsets," *Expert Systems with Applications*, vol. 42, pp. 2371-2381, 2015.

[10] T.-P. Hong, C.-H. Lee and S.-L. Wang, "Mining High Average-Utility Itemsets," *Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics*, 2009.

[11] C.-W. Lin, T.-P. Hong and W.-H. Lu, "Efficiently Mining High Average Utility Itemsets with a Tree Structure," *Lect. Notes Comput. Sci.* , vol. 5990, p. 131–139, 2010.

[12] U. Yun and . K. Donggyu, "Mining of high average-utility itemsets using novel list structure and pruning strategy," *Future Generation Computer Systems*, vol. 68, pp. 346-360, 2017.

[13] Y. Liu, W.-k. Liao and A. Choudhary, "A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets," *Lect. Notes Comput. Sci.*, vol. 3518, p. 689–695, 2005.

[14] B. Le, H. Nguyen, T. A. Cao and B. Vo, "A Novel Algorithm for Mining High Utility Itemsets," *First Asian Conference on Intelligent Information and Database Systems*, pp. 13-17, 2009.

[15] V. S. Tseng, C.-W. Wu, B.-E. Shie and P. S. Yu, "UP-Growth: An Efficient Algorithm for High Utility Itemset Mining," *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 253-262, 2010.

[16] V. S. Tseng, B.-E. Shie, C.-W. Wu and P. S. Yu, "Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases," *IEEE transactions on knowledge and data engineering*, pp. 1772-1786, 2012.

[17] S. Zida, P. Fournier-Viger, J. C.-W. Lin, C.-W. Wu and V. S. Tseng, "EFIM: A Fast and Memory Efficient Algorithm for High-Utility Itemset Mining," *Knowledge and Information Systems*, vol. 51, no. 2, pp. 595-625, 2017.

[18] S. Krishnamoorthy, "HMiner: Efficiently mining high utility itemsets," *Expert Systems With Applications*, vol. 90, p. 168–183, 2017.

[19] T.-P. Hong, C.-H. Lee and S.-L. Wang, "Effective utility mining with the measure of average utility," *Expert Systems with Applications*, vol. 38, p. 8259–8265, 2011.

[20] G.-C. LAN, T.-P. HONG and V. S. TSENG, "A Projection-Based Approach for Discovering High Average-Utility Itemsets," *Journal of Information Science and Engineering* , vol. 28, p. 193–209, 2012.

[21] J. C.-W. Lin, T. Li, P. Fournier-Viger, T.-P. Hong, J. Zhan and M. Voznak, "An efficient algorithm to mine high average-utility itemsets," *Advanced Engineering Informatics*, vol. 30, p. 233–243, 2016.

[22] D. Kim and U. Yun, "Efficient algorithm for mining high average-utility itemsets in incremental transaction databases," *Applied Intelligence*, vol. 47, no. 1, pp. 114-

131, 2017.

[23] T.-P. Hong, C.-H. Lee and S.-L. Wang, "An incremental mining algorithm for high average-utility itemsets," *In 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, pp. 421-425, 2009.

[24] U. Yun and H. Ryang, "Incremental high utility pattern mining with static and dynamic databases," *Applied intelligence*, vol. 42, no. 2, pp. 323-352, 2015.

[25] B. Zhang, J. Chun-Wei Lin, Y. Shao, P. Fournier-Viger and Y. Djenouri, "Maintenance of Discovered High Average-Utility Itemsets in Dynamic Databases," *Applied Sciences*, vol. 8, p. 769, 2018.

[26] I. Yildirim and M. Celik, "Mining High-Average Utility Itemsets with Positive and Negative External Utilities," *New Generation Computing*, vol. 38.1, pp. 153-186, 2020.

[27] J. Ming-Tai Wu, Q. Teng, J. Chun-Wei Lin, U. Yun and H.-C. Chen, "Updating high average-utility itemsets with pre-large concept," *Journal of Intelligent & Fuzzy Systems*, vol. 38, no. 5, pp. 5831-5840, 2020.

[28] P. Fournier-Viger, J. Chun-Wei Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng and H. Thanh Lam, "The SPMF open-source data mining library version 2," *Joint European conference on machine learning and knowledge discovery in databases*, pp. 36-40, 2016.

[29] J. Chun-Wei Lin, J. Ming-Tai Wu, P. Fournier-Viger, T.-P. Hong and T. Li, "Efficient Mining of High Average-Utility Sequential Patterns from Uncertain Databases," *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2019.

AN EFFECTIVE ALGORITHM FOR MINING HIGH AVERAGE-UTILITY ITEMSETS

Abstract: High average-utility itemsets mining (HAUI) is popularly researched to overcome the limitations of High Utility Itemset (HUI) in evaluating user results. HAUIs shows faithfully highly useful items. In which, the length of the itemset is considered, which eliminates the HUIs containing many items of poor significance in the business analysis results. Recently, many algorithms have been proposed for mining high average-utility itemset, but the execution performance is still ineffective. In this paper, we propose the HAU-Miner algorithm to mine HAUI in a better way. Experimental results on two groups of dense and sparse databases show that the HAU-Miner algorithm has higher performance than the MHAI algorithm in terms of the number of generated candidates, execution time and memory usage.

Keywords: High average-utility itemsets, Data mining, Transactional databases, Average-Utility Upper-Bound, Average-utility.

Phạm Tuấn Khiêm, nhận bằng Đại học tại Trường ĐH Sư phạm Tp.HCM năm 2006, chuyên ngành Sư phạm Tin học; nhận bằng Thạc sỹ tại Trường ĐH Công nghệ Thông tin Tp.HCM năm 2016, chuyên ngành Khoa học máy tính. Hiện công tác tại khoa Công nghệ thông tin, trường Đại học Công nghiệp Thực phẩm TPHCM. Hướng nghiên cứu: Khai thác tập hữu ích-trung bình cao trên cơ sở dữ liệu giao dịch, gom cụm văn bản.

Email: khiempt@hufi.edu.vn

Nguyễn Văn Lễ, nhận học vị Thạc sỹ năm 2011, chuyên ngành Truyền dữ liệu & Mạng máy tính tại Học viện Công nghệ Bưu chính Viễn Thông TPHCM. Hiện công tác tại khoa Công nghệ thông tin, trường Đại học Công nghiệp Thực phẩm TPHCM. Hướng nghiên cứu: Khai thác luật kết hợp, tập hữu ích cao trên cơ sở dữ liệu giao dịch, phân lớp văn bản.

Email: lenv@hufi.edu.vn