RESEARCH ARTICLE

KHAI THÁC TẬP MỤC HỮU ÍCH CAO TỪ CÁC LUỒNG DỮ LIỆU DỰA TRÊN DI TRUYỀN

Lê Thị Minh Nguyện, Phạm Đức Thành, Trần Anh Duy

Khoa Công nghệ thông tin, Trường Đại học Ngoại ngữ -Tin học TP.HCM

nguyenltm@huflit.edu.vn, thanhpd@huflit.edu.vn, duyta@huflit.edu.vn

TÓM TẮT— Khai thác tập mục hữu ích cao (HUIM) từ các luồng dữ liệu với thời gian và không gian giới hạn là một nhiệm vụ đầy thách thức. Các thuật toán truyền thống thường phải quét dữ liệu nhiều lần và sử dụng các cấu trúc dữ liệu phức tạp để kết nối, lưu trữ và cập nhật thông tin. Hơn nữa, việc mất các tập mục do các thuật toán heuristic gây ra và việc đánh giá các tập mục trùng lặp được tạo ra bởi các lô dữ liệu thông thường đều góp phần làm cho thuật toán kém hiệu quả về thời gian và không gian. Để giải quyết những vấn đề này, chúng tôi đề xuất một thuật toán mới dựa trên giải thuật di truyền (GA) để khai thác các tập mục có giá trị cao từ luồng dữ liệu, được gọi là HUIM_DS_GA, nhằm giải quyết hiệu quả vấn đề không gian lưu trữ hạn chế. Thuật toán HUIM_DS_GA thiết kế một chiến lược cập nhật nhóm mới, giúp tăng tốc độ hội tụ và giảm thiểu mất mát các tập mục quan trọng. Ngoài ra, chúng tôi đề xuất chiến lược lưu trữ bảng băm để tránh việc đánh giá các tập mục trùng lặp, từ đó cải thiện hiệu quả thực thi của thuật toán. Các thử nghiệm trên tập dữ liệu thực tế và dữ liệu tổng hợp cho thấy thuật toán hoạt động hiệu quả, giảm đáng kể lượng bộ nhớ tiêu thụ mà vẫn giữ được khả năng mở rộng tốt hơn so với các phương pháp trước đây.

Từ khóa— khai thác tập mục hữu ích cao; luồng dữ liệu; bảng băm; di truyền; cửa sổ trượt.

I. GIỚI THIỆU

Khai thác tập mục hữu ích cao (HUIM) là một chủ đề nghiên cứu quan trọng trong lĩnh vực khai thác dữ liệu. HUIM mở rộng khái niệm khai thác tập mục phổ biến bằng cách không chỉ xem xét tần suất xuất hiện mà còn quan tâm đến tầm quan trọng của các tập mục, nhằm tìm ra thông tin ẩn có giá trị từ các tập dữ liệu lớn. Ví dụ, trong phân tích thị trường bán lẻ, số lượng và giá của các mặt hàng trong mỗi giao dịch được coi là độ hữu ích nội và độ hữu ích ngoại. Lợi nhuận của một mặt hàng, hay còn gọi là độ hữu ích, được tính bằng tích của độ hữu ích nội và độ hữu ích ngoại.

Công trình nghiên cứu ban đầu về HUIM liên quan đến các thuật toán hai giai đoạn [1], yêu cầu quét nhiều lần các tập dữ liệu, đặc biệt là đối với các tập dữ liệu lớn. Việc tạo ra quá nhiều tập mục ứng viên có thể làm giảm đáng kể hiệu suất của thuật toán. Để tạo ra các tập mục ứng viên một cách hiệu quả và giảm thiểu việc quét cơ sở dữ liệu, các phương pháp cây tăng trưởng mẫu, chẳng hạn như cây IHUP [2], UP-tree [3], và HUITWU-tree [4], đã chứng minh hiệu quả trong HUIM. Mặc dù có cấu trúc gọn nhẹ, hiệu suất của các thuật toán này phụ thuộc chặt chẽ vào số lượng cây điều kiện, dẫn đến yêu cầu bộ nhớ đáng kể. Liu và cộng sự [5] đã đề xuất chuyển đổi cơ sở dữ liệu gốc thành cấu trúc danh sách, giúp tránh việc tạo ứng viên và quét lại cơ sở dữ liệu, từ đó cải thiện hiệu quả khai thác. Tuy nhiên, khi kích thước dữ liệu tăng lên, việc liệt kê tất cả các tập mục hữu ích cao (HUIs) trở thành thách thức và hiệu suất của các thuật toán này cũng suy giảm theo.

Trong những năm gần đây, với sự bùng nổ của các luồng dữ liệu vô hạn từ cảm biến, dữ liệu giao dịch và mạng truyền thông, việc khai thác HUIs từ luồng dữ liệu đã thu hút được sự quan tâm đáng kể trong nghiên cứu. Không giống như các tập dữ liệu truyền thống, luồng dữ liệu có đặc điểm là dòng chảy tốc độ cao, thay đổi theo thời gian thực, có thứ tự và khối lượng lớn. Khai thác dữ liệu từ luồng dữ liệu phải đối mặt với những thách thức như bùng nổ tổ hợp, các ràng buộc không gian-thời gian, và sự thay đổi khái niệm do thay đổi dữ liệu. Để thích ứng với bản chất của luồng dữ liệu và đáp ứng các nhu cầu ứng dụng thực tế, các mô hình cửa sổ đã trở thành công nghệ quan trọng trong xử lý luồng dữ liệu. Các mô hình này xử lý các giao dịch gần đây theo lô liên tục trong một phạm vi cửa sổ được xác định. Các mô hình cửa sổ cổ điển bao gồm cửa sổ dấu mốc, cửa sổ suy giảm, cửa sổ trượt và cửa sổ giảm dần [6]. Trong HUIM, cửa sổ trượt và cửa sổ giảm dần là các kỹ thuật được sử dụng phổ biến nhất. Ví dụ, thuật toán SOHUPDS [7] sử dụng kỹ thuật chiếu cơ sở dữ liệu và cửa sổ trượt để khai thác các tập mục hữu ích cao từ luồng dữ liệu. Cấu trúc này lưu trữ độ hữu ích hiệu quả, giới hạn trên và vị trí của các mục trong cửa sổ trượt hiện tại. HUIM-Stream [8] đề xuất bảng chỉ mục vị trí Ext-list và quy trình hiệu quả để xây dựng và cập nhật, giúp giảm đáng kể độ phức tạp thời gian của thuật toán.

Thuật toán khai thác tập mục hữu ích cao trên dòng dữ liệu truyền thống thường yêu cầu quét dữ liệu nhiều lần và sử dụng các cấu trúc dữ liệu phức tạp để kết nối, lưu trữ và cập nhật dữ liệu. Mặc dù có nhiều chiến lược cắt tỉa được thiết kế nhằm cải thiện hiệu suất, các thuật toán này vẫn bị hạn chế bởi việc sử dụng bộ nhớ cao. Ngoài ra, khi dữ liệu mới đến, cửa sổ trượt loại bỏ các lô dữ liệu cũ và thêm lô mới, dẫn đến sự trùng lặp đáng kể giữa các cửa sổ, làm giảm hiệu suất thời gian. Tính ngẫu nhiên của các thuật toán heuristic cũng có thể dẫn đến mất các tập mục hữu ích cao và ảnh hưởng đến khả năng hội tụ.

Để giải quyết các vấn đề này, trong bài báo này chúng tôi giới thiệu sự kết hợp giữa các thuật toán heuristic và công nghệ cửa sổ trượt, đề xuất hai chiến lược hiệu quả để khai thác các tập mục hữu ích cao (HUIs) từ dòng dữ liệu. Sử dụng thuật toán tối ưu GA nhờ khả năng tìm kiếm toàn cục, số lượng tham số điều khiển tối thiểu và hiệu suất đã được chứng minh trong các bài toán tối ưu liên tục, tổ hợp và ràng buộc trên nhiều tiêu chuẩn và ứng dụng khác nhau.

Phần còn lại của bài báo được trình bày như sau. Phần II trình bày về các công trình liên quan. Để cung cấp nền tảng và ngữ cảnh, trong phần III trình bày các khái niệm và định nghĩa, trình bày cách nhìn tổng quan về dữ liệu luồng, độ hữu ích của một mục, độ hữu ích của tập mục, độ hữu ích trong giao dịch và ngưỡng hữu ích tối thiểu đồng thời giới thiệu các khái niệm về GA. Phần IV trình bày thuật toán đề xuất. Tiếp theo, trình bày kết quả thực nghiệm về phương pháp đề xuất ***HUIM_DS_GA*** khác biệt so với thuật toán ***HSLM*** [9] và thuật toán HUIM-stream [8]. Cuối cùng, trình bày phần kết luận và tương lai của đề tài.

II. NGHIÊN CỨU LIÊN QUAN

A. KHAI THÁC TẬP MỤC CÓ HỮU ÍCH CAO BẰNG PHƯƠNG PHÁP HEURISTIC

Với sự phát triển bùng nổ của dữ liệu, thuật toán khai thác HUIM truyền thống đã không thể đối phó với sự gia tăng theo cấp số nhân của không gian tìm kiếm. Để giải quyết vấn đề này, các thuật toán HUIM dựa trên phương pháp heuristic khác nhau đã được phát triển.

Kannimuthu [10] lần đầu áp dụng các thuật toán heuristic vào khai thác HUIM, và đề xuất hai phương pháp HUIM dựa trên thuật toán di truyền, cụ thể là thuật toán HUPEumu-GARM có ngưỡng hữu ích tối thiểu và HUPEwumu-GARM không có ngưỡng hữu ích tối thiểu. Các phương pháp này có thể khai thác hiệu quả các tập mục có hữu ích cao trong không gian tìm kiếm theo cấp số nhân, nhưng chúng dễ rơi vào tối ưu cục bộ và cho thấy sự hội tụ chậm. Thuật toán TKHUIM-GA [11] hướng dẫn quá trình tìm kiếm bằng cách xem xét hữu ích của từng mục và đề xuất một phương pháp khai thác HUIM Top-k khác mà không cần ngưỡng hữu ích tối thiểu cài đặt trước. Phương pháp này không phụ thuộc vào loại dữ liệu và giảm thời gian chạy cũng như mức tiêu thụ bộ nhớ. Thuật toán HUIM-IGA [12] là một phương pháp HUIM tiên tiến dựa trên thuật toán di truyền cải tiến. Phương pháp này giới thiệu chiến lược sửa chữa dựa trên 1-HTWUI và chiến lược tìm kiếm hàng xóm ưu tú, giúp tăng cường xác suất tiến hóa của các cá thể xuất sắc và mở rộng không gian tìm kiếm cho giải pháp tối ưu. Thuật toán PPUMGAT+ [13] tập trung vào vấn đề bảo vệ quyền riêng tư và đề xuất một thuật toán bảo vệ quyền riêng tư cho các tập mục có hữu ích cao dựa trên GA. Phương pháp này mở rộng khái niệm "pre-large" và sử dụng ngưỡng hỗ trợ trên và dưới như một vùng đệm an toàn để xóa bỏ các tập mục có hữu ích cao nhạy cảm. Ngoài ra, DcGA [14] đã trình bày một thuật toán HUIM đóng. Thuật toán sử dụng mô hình phân cụm để nhóm các giao dịch liên quan, áp dụng thuật toán di truyền nén (CGA) để xử lý các giao dịch đã được nhóm nhằm tìm ra các tập mục có hữu ích cao đóng cục bộ (CHUIs), và sau đó thực hiện các thao tác nối tiếp để thu được các CHUIs toàn cục. CGA có độ chính xác tương đương với thuật toán di truyền thông thường, nhưng giảm thời gian và mức tiêu thụ bộ nhớ. Để khai thác nhanh các CHUIs trên các tập dữ liệu lớn, MCUI-Miner [15] đã thiết kế thêm một mô hình đa mục tiêu. Mô hình này sử dụng phân nhóm k-means và khung làm việc MapReduce ba tầng dựa trên GA để tạo ra các CHUIs. Đối với các tập mục có hữu ích cao, MOBGWO [16] sử dụng các yếu tố phân rã để giải quyết các vấn đề tối ưu hóa đa mục tiêu và đề xuất một thuật toán đa mục tiêu khác. Các thí nghiệm quy mô lớn cho thấy phương pháp này hiệu quả và có ý nghĩa.

B. KHAI THÁC TẬP HỢP MỤC CÓ ĐỘ HỮU ÍCH CAO TRÊN LUỒNG DỮ LIỆU

Khác với các tập dữ liệu truyền thống, luồng dữ liệu là vô hạn, tốc độ cao, có trật tự và kịp thời, do đó việc khai thác các tập hợp mục có độ hữu ích cao (HUIs) trong thời gian và không gian hiệu quả là một thách thức. Nhắm vào chủ đề nghiên cứu này, một số phương pháp khai thác tập hợp mục có độ hữu ích cao cho luồng dữ liệu với các mô hình cửa sổ khác nhau đã được đề xuất.

Trong mô hình cửa sổ trượt, mỗi cửa sổ có kích thước cố định bao gồm một số lô dữ liệu nhất định. Khi lô dữ liệu mới đến và đạt đến kích thước cửa sổ, cửa sổ sẽ được cập nhật, tức là thêm các lô mới nhất và xóa các lô cũ nhất. Vì một số phương pháp khai thác tập hợp mục có độ hữu ích cao trên luồng dữ liệu được đề xuất yêu cầu quét cơ sở dữ liệu nhiều lần, nên quá trình này mất nhiều thời gian xử lý. Do đó, thuật toán SOHUPDS [7] sử dụng kỹ thuật chiếu cơ sở dữ liệu và cửa sổ trượt để khai thác tập hợp mục có độ hữu ích hiệu quả trên luồng dữ liệu. Trong phương pháp này, một cấu trúc dữ liệu mới, IUDataListSW, được thiết kế để lưu trữ độ hữu ích, giới hạn trên và vị trí của các mục trong cửa sổ trượt hiện tại nhằm hỗ trợ tìm kiếm các mục hiệu quả. Phương pháp này cũng sử dụng các tập hợp mục có độ hữu ích cao (HUIs) của cửa sổ cũ để cập nhật dữ liệu của cửa sổ trượt hiện tại. SOHUPDS hoạt động hiệu quả về mặt thời gian thực thi và mức sử dụng bộ nhớ. Bởi vì giá trị tham chiếu của dữ liệu lịch sử hiếm khi được xem xét trong tài liệu hiện có, thuật toán HUMHDT [17] thiết kế một kiến trúc hệ thống phân tán, có thể xây dựng và cập nhật dữ liệu lịch sử mà không làm ảnh hưởng đến thuật toán khai thác luồng dữ liệu, đồng thời tối ưu hóa thuật toán khai thác luồng dữ liệu hiện tại thông qua các bảng dữ liệu lịch sử.

Để giảm chi phí xử lý lại các lô dữ liệu chung trong các cửa sổ trượt, Reddy và cộng sự [18] đã sử dụng cấu trúc cây tập hợp mục hữu ích toàn cục mở rộng (EGUI-tree) để lưu trữ và truy xuất thông tin một cách linh hoạt. Để giải quyết sự không hiệu quả về không gian và thời gian của thuật toán do quá nhiều phép toán nối của danh sách hữu ích truyền thống và sự lặp lại của cùng một tập kết quả trong mô hình cửa sổ trượt, thuật toán HUIM-Stream [8] thiết kế một cấu trúc bảng chỉ mục vị trí Ext-list và xây dựng một quy trình xây dựng và cập nhật hiệu quả để duy trì thông tin hữu ích và vị trí của các mục trong cửa sổ trượt một cách hiệu quả, từ đó giảm đáng kể độ phức tạp thời gian của thuật toán. Thuật toán cũng đề xuất một chiến lược duy trì kết quả dựa trên cấu trúc bảng băm (HRS), giúp giảm không gian tìm kiếm của thuật toán bằng cách lưu trữ và cập nhật tập kết quả một cách hiệu quả, đồng thời nâng cao hiệu suất chạy của thuật toán.

III. CÁC ĐỊNH NGHĨA VÀ KHÁI NIỆM

A. ĐỊNH NGHĨA

Cho $I = \{i_1, i_2, \dots, i_m\}$ là một tập hợp các mục khác nhau, $Ds = \{T_1, T_2, \dots, T_n\}$ là một chuỗi giao dịch, và mỗi giao dịch $T_q \in DS$ $(1 \le q \le n)$ là một tập con của I. Hàm $q(i_j, T_q)$ đại diện cho độ hữu ích nội của mục i_j trong giao dịch T_q, và $p(i_j)$ đại diện cho độ hữu ích ngoại của mục i_j. Tập mục $X = \{i_1, i_2, \dots, i_k\}$ (1 ≤ k ≤ m) là một tập con khác rỗng của I. Trong khai thác tập mục hữu ích cao (HUIM) trên dòng dữ liệu dựa trên mô hình cửa sổ trượt, mỗi cửa sổ $W_b = \{B_{b+1}, B_{b+2}, \dots, B_{b+r}\}$ bao gồm một số lượng cố định r các lô (batch), mỗi lô $B_t = \{T_{t+1}, T_{t+2}, \dots, T_{t+s}\}$ bao gồm một số lượng cố định s các giao dịch.

Hình 1 minh họa một ví dụ đơn giản về luồng dữ liệu, trong đó kích thước cửa sổ là 2 và kích thước lô là 3. IU (Internal Utility) là độ hữu ích nội, và TU (Transaction Utility) là độ hữu ích của giao dịch cho các mục trong từng dòng giao dịch. Bảng 1 thể hiện độ hữu ích ngoại của từng mục trong luồng dữ liệu.

Data stream

W_1

W_2

BID	TID	Items	IU	TU
B_1	T_1	a, b, d, e	2, 3, 3, 5	93
	T_2	a, b, c, d, e	3, 3, 4, 5, 2	122
	T_3	b, d, e, f	3, 3, 5, 4	113
B_2	T_4	a, c, e, f, g	1, 3, 4, 5, 4	164
	T_5	c, e, g	3, 4, 5	131
	T_6	a, b, f, g	3, 2, 2, 3	87
B_3	T_7	c, d, e	1, 3, 2	43
	T_8	b, c, e	3, 1, 5	76
	T_9	b, d, e, g	3, 4, 5, 6	164

Hình 1. Dữ liệu luồng

Và bảng hữu ích ngoại (EU) như sau:

Bảng 1. Giá trị hữu ích ngoại (EU)

Item	a	B	C	D	e	f	g
EU	6	7	10	5	9	8	13

Định nghĩa 1. Độ hữu ích của một mục [8]. Mục i_j trong giao dịch T_q được biểu diễn là $u(i_j, T_q)$. Độ hữu ích này được tính bằng cách nhân hữu ích nội của mục i_j trong giao dịch T_q với hữu ích ngoại của i_j, theo công thức (1).

$$u(i_j, T_q) = q(i_j, T_q) \times p(i_j) \quad (1)$$

Ví dụ như trong Hình 1 và Bảng 1, $u(a, T_1) = q(a, T_1) \times p(a) = 2 \times 6 = 12$

Định nghĩa 2. Độ hữu ích của tập mục trong giao dịch [8]. Độ hữu ích của tập mục X trong giao dịch T_q được biểu diễn là $u(X, T_q)$, là tổng hữu ích của tất cả các mục thuộc tập X có trong giao dịch T_q. Độ hữu ích này được tính theo công thức (2).

$$u(X, T_q) = \sum_{i_j \in X \wedge X \in t_q} u(i_j, T_q) \quad (2)$$

Ví dụ: $u(\{a,b\},T_1) = u(a,T_1) + u(b,T_1) = 2 \times 6 + 3 \times 7 = 33$

Định nghĩa 3. Độ hữu ích của giao dịch [8]**.** Độ hữu ích của một giao dịch T_q được biểu diễn là $TU(T_q)$, là tổng hữu ích của tất cả các mục trong T_q, được tính theo công thức (3).

$$tu(T_q) = \sum_{i_j \in T_q} u(i_j, T_q) \quad (3)$$

Ví dụ: $TU(T_5) = u(c,T_5) + u(e,T_5) + u(g,T_5) = 3 \times 10 + 4 \times 9 + 5 \times 13 = 131$

Định nghĩa 4. Độ hữu ích trọng số giao dịch (TWU) [8]. Tập mục X trong cửa sổ W_b, Độ hữu ích trọng số giao dịch được biểu diễn là $TWU(X, W_b)$, là tổng hữu ích của tất cả các giao dịch chứa X trong cửa sổ W_b, được tính theo công thức (4)

$$TWU(X, W_b) = \sum_{X \in T_q \wedge T_q \in W_b} TU(T_q) \quad (4)$$

Ví dụ: $TWU(\{a,c\}, W_1) = TU(T_2) + TU(T_4) = 122 + 164 = 286$

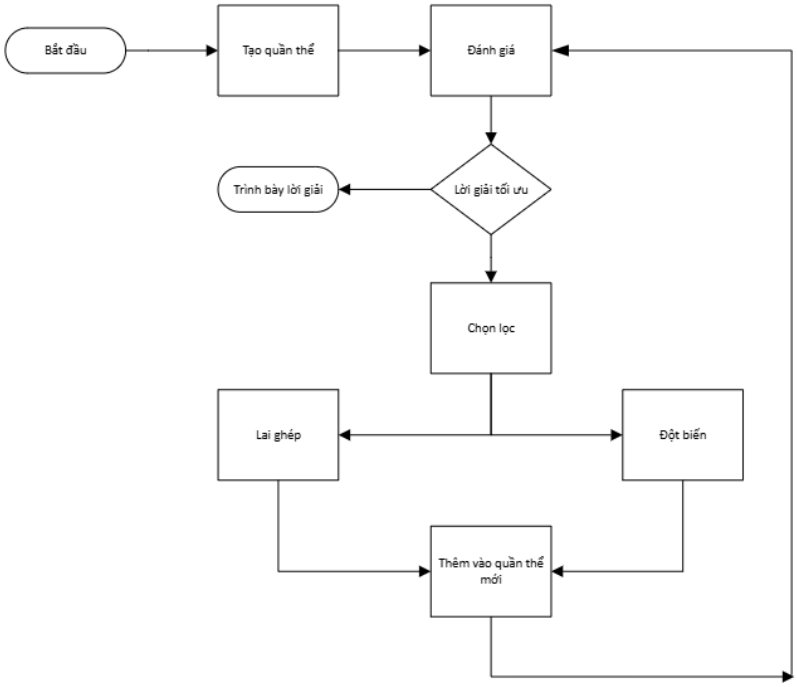
Định nghĩa 5. Ngưỡng hữu ích tối thiểu [8]. Cửa sổ W_b có ngưỡng hữu ích tối thiểu được biểu diễn là *min_util*, là tích của tổng hữu ích của tất cả các giao dịch trong cửa sổ W_b với tỷ lệ phần trăm ngưỡng hữu ích tối thiểu do người dùng định nghĩa δ, được tính theo công thức (5).

$$(\min\_util_{W_b}) = \sum_{T_q \in W_b} TU(T_q) \times \delta \quad (5)$$

Ví dụ: với độ hỗ trợ δ=0.3, ngưỡng hữu ích tối thiểu của của sổ W_1 có min_util=(93 + 122 + 113 + 164 +131 + 87) x 0.3 = 213.

B. GIỚI THIỆU SƠ LƯỢC VỀ GA

1. SƠ ĐỒ THUẬT TOÁN GA



Hình 2. Thuật toán di truyền GA

2. BIỂU DIỄN NHIỄM SẮC THỂ

Một cá thể được biểu diễn bằng 1 dãy các bit như Bảng 2 bên dưới:

Bảng 2. Biểu diễn một cá thể

Chrome	Items						
	a	b	c	d	e	f	g
C1	1	0	0	1	0	0	0

3. TẠO QUẦN THỂ

Lần lượt tạo các cá thể ta được quần thể như Bảng 3 sau:

Bảng 3. Quần thể sau khi được khởi tạo ngẫu nhiên

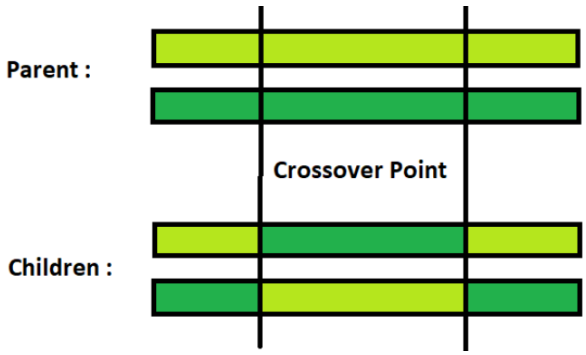
Chrome	Populations						
	a	b	c	d	e	f	g
C1	1	0	0	1	0	0	0
C2	0	1	1	0	1	1	0
C3	1	0	0	1	0	1	0
C4	0	1	0	1	1	0	0
C5	0	1	0	0	1	0	1
C6	1	0	1	0	0	0	1

4. CÁC TOÁN TỬ

a) Chọn lọc

Sau khi đánh giá, ta có thể loại bỏ các cá thể có độ thích nghi thấp, và nhân bản các cá thể có độ thích nghi cao. Việc loại bỏ hay nhân bản có thể được quyết định bằng nhiều phương pháp như: Sử dụng xác suất hoặc chọn ngẫu nhiên

b) Lai ghép

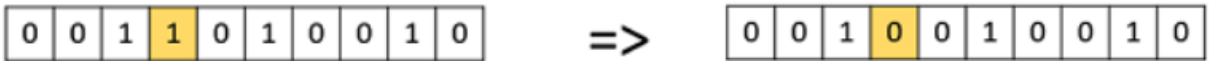


Hình 3. Lai ghép hai điểm

c) Đột biến

Đột biến lật bit

1. Chọn ngẫu nhiên 1 hay nhiều cá thể bất kỳ (mỗi cá thể có m gen).
2. Chọn ngẫu nhiên số k trong đoạn [1, m].
3. Thay đổi giá trị của gen thứ k.
4. Đưa cá thể mới tham gia vào quần thể.



Hình 4. Minh họa đột biến bằng cách lật bit

5. HÀM THÍCH NGHI (FITNESS)

Định nghĩa 6. Giá trị thích nghi của cá thể C_i so với một transaction T_q trong cửa sổ W_b, được tính như sau: Nếu $C_i \subseteq T_q$ thì $f_i(C_i, T_q) = \sum_{j \in C_j} U(j, T_q) \times bit(C_{ij})$, ngược lại bằng 0.

Ví dụ: $f_1(C_1, T_1) = 12+15 = 27$

Định nghĩa 7. Giá trị thích nghi của cá thể C_i so với một batch B_p trong cửa sổ W_b, được tính theo công thức (6) như sau:

$$f_{i\_p}(C_i, B_p) = \sum_{T_q \in B_p} f_i(C_i, T_q) \quad (6)$$

Ví dụ: $f_{1\_1}(C_1, B_p) = 27 + 43 + 0 = 70$

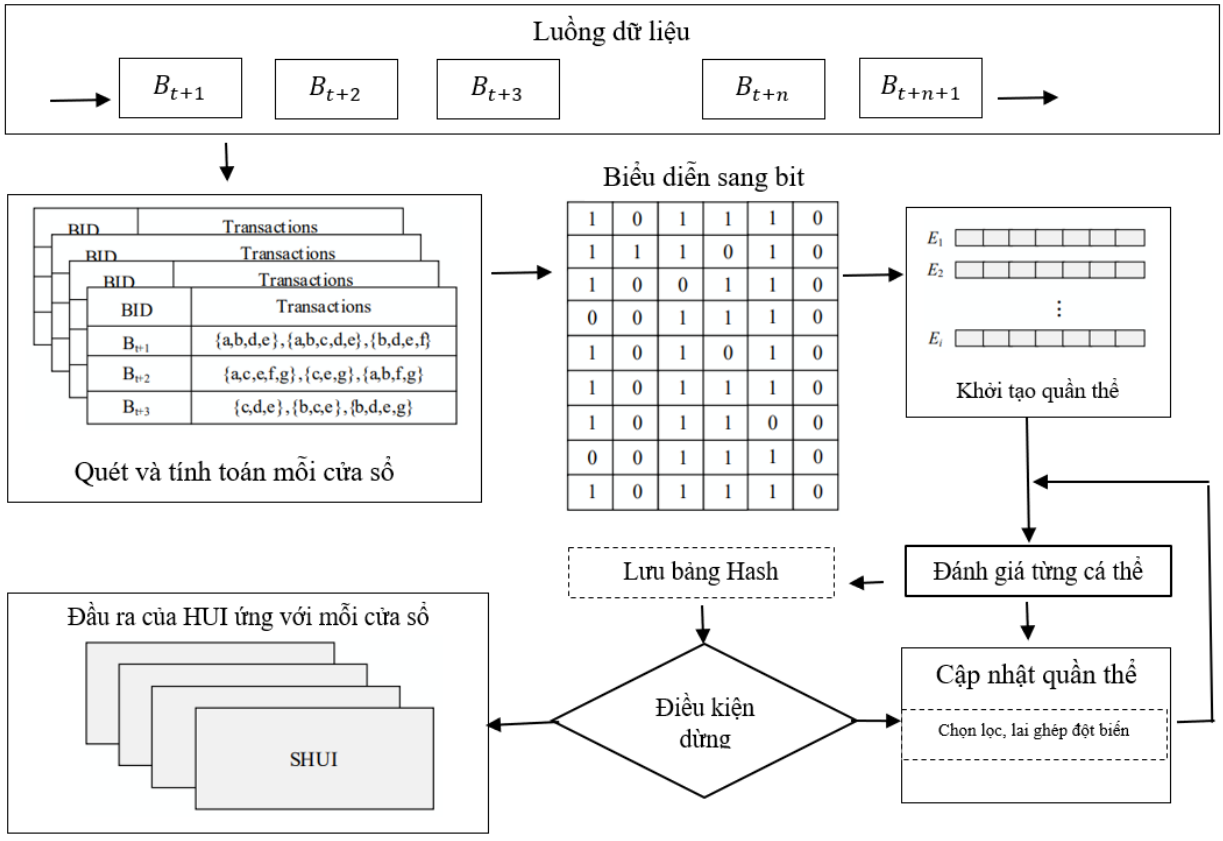
Định nghĩa 8. Giá trị thích nghi của cá thể C_i tính trên toàn bộ các transaction trong cửa sổ W_b, được tính theo công thức 7 như sau**:**

$$f_{i\_W_b}(C_i, W_b) = \sum_{B_p \in W_b} f_{i\_p}(C_i, B_p) \quad (7)$$

Ví dụ: $f_{1\_W_1}(C_1, W_1) = 70 + 0 = 70$

IV. THUẬT TOÁN ĐỀ XUẤT HUIM_DS_GA

A. CẤU TRÚC VÀ QUY TRÌNH THUẬT TOÁN HUIM_DS_GA



Hình 5. Cấu trúc và quy trình HUIM_DS_GA

(1) Đối với luồng dữ liệu DS đã cho, đầu tiên thuật toán quét dữ liệu mới đến và chèn các giao dịch vào cửa sổ trượt theo lô. Khi kích thước lô (batch_size) bằng với kích thước cửa sổ (win_size), thông tin của cửa sổ hiện tại sẽ được xử lý.

(2) Đối với dữ liệu của cửa sổ hiện tại, đầu tiên thuật toán chuyển đổi cơ sở dữ liệu đã được tổ chức lại thành bitmap.

(3) Mỗi vector vị trí của quần thể được khởi tạo và mã hóa thành các bit nhị phân.

(4) Thuật toán đánh giá mỗi vector vị trí của từng cá thể, lưu trữ giá trị độ thích nghi đã tính toán và tập hợp mục trong bảng băm, và thu được các cá thể đạt yêu cầu.

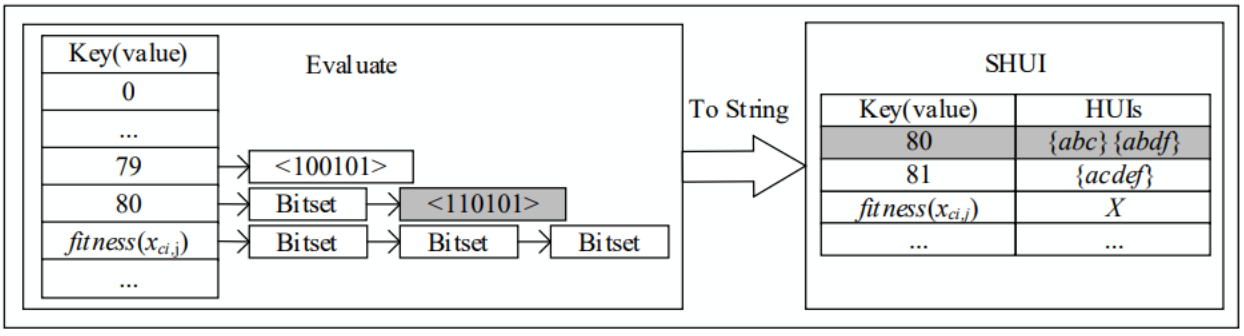
(5) Thuật toán sử dụng các toán tử lai ghép, đột biến, chọn lọc để tạo ra các cá thể mới nhằm bổ sung cho thế hệ tiếp theo. Quy trình cập nhật và đánh giá quần thể được lặp lại cho đến khi đạt số lần lặp tối đa, và các tập hợp mục có độ hữu ích cao (HUIs) của cửa sổ hiện tại được xuất ra.

(6) Khi có giao dịch mới đến, các lô cũ sẽ bị xóa, các lô mới sẽ được thêm vào cửa sổ. Thuật toán sẽ thực hiện một vòng đọc và xử lý dữ liệu mới như đã mô tả ở trên.

B. CHIẾN LƯỢC SỬ DỤNG BĂM ĐỂ LƯU TRỮ TRONG QUÁ TRÌNH ĐÁNH GIÁ VÀ QUẢN LÝ TẬP KẾT QUẢ

Để tránh việc đánh giá lặp lại các tập hợp mục, bài báo này đề xuất một chiến lược lưu trữ bảng băm cho việc đánh giá và kết quả. Chiến lược này sử dụng giá trị độ thích nghi đã tính toán $fitness(x_{ci,j})$ làm giá trị khóa của hàm băm, và lưu trữ các vector vị trí khác nhau có cùng giá trị độ thích nghi trong bảng băm dùng để đánh giá GA bằng phương pháp địa chỉ chuỗi. Nếu giá trị độ thích nghi tương ứng với vector vị trí $x_{ci,j}$ thỏa mãn ngưỡng hữu ích tối thiểu do người dùng định nghĩa, vector vị trí sẽ được chuyển đổi thành tập hợp mục tương ứng, và tập hợp mục cùng với giá trị độ thích nghi tương ứng sẽ được lưu trữ trong bảng băm tập kết quả.

Ví dụ, trong bảng băm của giai đoạn đánh giá bầy, có các vector vị trí <100101> và <110101> tương ứng với các giá trị độ thích nghi là 79 và 80. Nếu ngưỡng độ hữu dụng tối thiểu là 80, vector vị trí <110101> với giá trị độ thích nghi 80 sẽ được chuyển đổi thành tập hợp mục tương ứng <abdf> và được lưu trữ trong bảng băm của tập kết quả SHUI, như thể hiện trong Hình 6. Trong lần đánh giá tiếp theo, nếu có một vector vị trí có giá trị hữu ích nhỏ hơn ngưỡng hữu ích tối thiểu, chiến lược này trước tiên sẽ kiểm tra xem tập hợp đó có tồn tại trong bảng băm dùng để đánh giá hay không, nếu không có, sẽ tính toán giá trị hữu ích và liên kết nó với giá trị khóa tương ứng. Nếu có một vector vị trí có giá trị hữu ích lớn hơn ngưỡng hữu ích tối thiểu, sử dụng tìm kiếm nhị phân được để nhanh chóng xác định giá trị hữu ích trong bảng băm lưu trữ tập kết quả. Nếu vector vị trí không tồn tại, nó sẽ được thêm vào bảng băm của tập kết quả SHUI, và nếu đã tồn tại, vector vị trí tiếp theo sẽ được đánh giá trực tiếp.



Hình 6. Bảng Băm để lưu trữ SHUI

1. THUẬT TOÁN GA-RUN.

Algorithm 1 **genetic algorithm-Run**
Input Số thế hệ *generations*, quần thể *population*, kích thước quần thể *population_size*, bộ dữ liệu vào *databitset*, tập hữu ích cao huims_in (hashtable)
Output Các tập hữu ích cao huims_out

```
1.  huims_out ← huims_in
2.  for generation ← 0 to generations do
3.      new_population ← population[:population_size//2]
4.      while len(new_population) < population_size do
5.          if random.random() < crossover_probability then
6.              parent1, parent2 ← randomSelected(population)
7.              child1, child2 ← crossover(parent1, parent2)
8.              if huims_out.append(child1) then
9.                  insert_pos ← bisect.bisect_left([-item.fitness for item in new_population], -
                    child1.fitness)
10.                 new_population.insert(insert_pos, child1)
11.             end if
```

12. **if** huims_out.append(child2) **then**
13. insert_pos ← bisect.bisect_left([-item.fitness for item in new_population], -child2.fitness)
14. $new\_population.insert(insert\_pos, child2)$
15. **end if**
16. **end if**
17. **if** random.random() < mutation_probability **then**
18. child_to_mutate ← $randomSelected(population)$
19. mutated_child ← *mutate* (child_to_mutate)
20. **if** huims_out.append(child2) **then**
21. insert_pos ← bisect.bisect_left([-item.fitness for item in new_population], -mutated_child.fitness)
22. $new\_population.insert(insert\_pos$, mutated_child)
23. **end if**
24. **end if**
25. **end while**
26. population ← new_population
27. **end for**
28. **Return** huims_out

Dòng 1: Gán giá trị của tập hữu ích cao vào (huims_in) cho tập hữu ích cao ra (huims_out).

Dòng 2: Lặp số lượng thế hệ là $generations$.

Dòng 3: Tạo quần thể mới từ nửa đầu của quần thể đang xét population.

Dòng 4 đến 25: Lặp cho đến khi số lượng của new_population bằng với population_size.

Dòng 5: Nếu con số phát sinh ngẫu nhiên nhỏ hơn ngưỡng xác suất lai ghép crossover_probability thì.

Dòng 6: Chọn ngẫu nhiên 2 cá thể cha mẹ để lai ghép.

Dòng 7: Thực hiện việc lai ghép để tạo ra 2 cá thể con mới.

Dòng 8: Nếu cá thể con 1 đưa vào tập phổ biến huims_out được thì.

Dòng 9: Xét vị trị cần chèn thích hợp insert_pos để new_population luôn được sắp giảm.

Dòng 10: Đưa cá thể con 1 mới vừa tạo vào quần thể new_population tại vị trí cần chèn insert_pos.

Dòng 12: Nếu cá thể con 2 đưa vào tập phổ biến huims_out được thì.

Dòng 13: Xét vị trị cần chèn thích hợp insert_pos để new_population luôn được sắp giảm.

Dòng 14: Đưa cá thể con 2 mới vừa tạo vào quần thể new_population tại vị trí cần chèn insert_pos.

Dòng 17: Nếu con số phát sinh ngẫu nhiên nhỏ hơn ngưỡng xác suất đột biến mutation_probability thì.

Dòng 18: Chọn ngẫu nhiên 1 cá thể con trong quần thể.

Dòng 19: Thực hiện việc đột biến cá thể này.

Dòng 20: Nếu cá thể con vừa đột biến đưa vào tập phổ biến huims_out thì.

Dòng 21: Xét vị trị cần chèn thích hợp insert_pos để new_population luôn được sắp giảm.

Dòng 22: Đưa cá thể con mới vừa đột biến vào quần thể new_population tại vị trí cần chèn insert_pos.

Dòng 26: Sao chép quần thể cho lần lặp tiếp theo từ quần thể đã được sắp new_population với số lượng là pop_size.

Dòng 28: Trả về kết quả tìm được là tập phổ biến huims_out.

2. THUẬT TOÁN HUIM_DS_GA

Algorithm 2 ***HUIM_DS_GA (Mining High Utility Itemsets from GA-Based Data Streams)***
Input Dữ liệu $trans\_list$, $biggest\_item$, số transaction trên một batch $batch\_size$, số batch trong một cửa sổ trượt $num\_batch\_per\_window$, kích thước quần thể $population\_size$, ngưỡng hữu ích tối thiểu min_util.
Output Danh sách lưu các tập hữu ích cao của mỗi lần trượt *all_huims*

```
1.  all_huims ← ∅
2.  window_size ← batch_size * num_batch_per_window
3.  window_data ← ∅
4.  i ← 0
5.  while i < window_size do
6.          batch_data ← ∅
7.          j ← 0
8.          while j < batch_size do
9.                  batch_data.append(trans _list[i]);
10.                 i ← i + 1 ; j ← j + 1
11.         end while
12.         window_data.append(batch_data)
13. end while
14. population, huims_out ← InitPopulation_BitArray(population_size,window_data,
                                                    biggest_item, avg_len, min _util)
15. huims_out ← GA_Run()
16. all_huims ← all_huims + huims_out
17. while i <= len(items_list) − batch_size do
18.         batch_data ← ∅
19.         j ← 0
20.         while j < batch_size do
21.                 batch_data.append(trans _list[i]);
22.                 i ← i + 1; j ← j + 1
23.         end while
24.         window_data.append(batch_data)
25.         window_data.pop(0)
26.         UpdatePopulation(batch_data)
27.         huims_out ← GA_Run()
28.         all_huims ← all_huims + huims_out
29. end while
30. Return all_huims
```

Dòng 1: Khởi tạo danh sách tất cả các tập hữu ích là rỗng.

Dòng 2: Tính kích thước của cửa sổ trượt *window_size*.

Dòng 3: Khởi tạo window_data rỗng

Dòng 4: Gán i = 0, để duyệt từ đầu *trans_list*.

Dòng 5 → 16: Duyệt số batch trong 1 cửa sổ trượt, để tạo cửa sổ trượt lần đầu tiên.

Dòng 6: Khởi tạo batch_data rỗng.

Dòng 7: Gán i = j, để lặp *batch_size* lần

Dòng 8: Lặp batch_size lần để thực hiện các việc sau:

Dòng 9: Thêm từng dòng dữ liệu *trans_list*[i] vào batch_data

Dòng 10: Tăng thêm 1 cho hai biến chạy của vòng lặp i và j.

Dòng 12: Thêm batch_data vào cho window_data.

Dòng 14: Khởi tạo quần thể với mỗi cá thể là bitarray, đồng thời tính fitness của từng cá thể với phương án tính toán song song.

Dòng 15: Gọi thực thi hàm GA_Run() với hai kết quả trả về là population và huims_out.

Dòng 16: Lưu huims_out vào trong all_huims

Dòng 17 → 29: Thực hiện việc lặp lần lượt qua các cửa sổ trượt tiếp theo và thực hiện các công việc sau:

Dòng 18 → 23: Đọc dữ liệu và tạo một batch_data mới.

Dòng 24: Thêm batch_data mới vào cho window_data.

Dòng 25: Xóa batch đầu tiên (cũ) ra khỏi cửa sổ trượt (window_data) đang xét.

Dòng 26: Cập nhật lại quần thể do batch_data mới được thêm vào cửa sổ trượt.

Dòng 27: Gọi thực thi hàm GA_Run() với hai kết quả trả về là huims_out.

Dòng 28: Lưu huims_out vào trong all_huims.

Dòng 30: Trả về all_huims.

3. VÍ DỤ MINH HOẠ:

Với CSDL luồng dữ liệu ở Hình 1 và Bảng 1. Dựa theo công thức (1) để tính $u(i_j, T_q)$, ta có $u(a, T_1) = q(a, T_1) \times p(a) = 2 \times 6 = 12, u(b, T_1) = q(b, T_1) \times p(b) = 3 \times 7 = 21, u(d, T_1) = q(d, T_1) \times p(d) = 3 \times 5 = 15, u(e, T_1) = q(e, T_1) \times p(e) = 5 \times 9 = 45$. Tương tự với các Tid còn lại, ta có kết quả ở Bảng 4 như sau:

Bảng 4. Sau khi tính U(i, T)

BID	***TID***	***Items***	***TU***	***U(i, T)***
B1	1	a, b, d, e	93	12, 21, 15, 45
	2	a, b, c, d, e	122	18, 21, 40, 25, 18
	3	b, d, e, f	113	21, 15, 45, 32
B2	4	a, c, e, f, g	164	6, 30, 36, 40, 52
	5	c, e, g	131	30, 36, 65
	6	a, b, f, g	87	18, 14, 16, 39
B3	7	c, d, e	43	10, 15, 18
	8	b, c, e	76	21, 10, 45
	9	b, d, e, g	164	21, 20, 45, 78

Từ Bảng 4 ta chuyển Items sang bitarray như trong Bảng 5 sau:

Bảng 5. Sau khi chuyển Items sang bitarray

	BID	TID	Items							TU	U(i,T)						
			a	b	c	d	e	f	g		a	b	c	d	e	f	g
W1	B1	1	1	1	0	1	1	0	0	93	12	21	0	15	45	0	0
		2	1	1	1	1	1	0	0	122	18	21	40	25	18	0	0
		3	0	1	0	1	1	1	0	113	0	21	0	15	45	32	0
	B2	4	1	0	1	0	1	1	1	164	6	0	30	0	36	40	52
		5	0	0	1	0	1	0	1	131	0	0	30	0	36	0	65
		6	1	1	0	0	0	1	1	87	18	14	0	0	0	16	39
	B3	7	0	0	1	1	1	0	0	43	0	0	10	15	18	0	0
		8	0	1	1	0	1	0	0	76	0	21	10	0	45	0	0
		9	0	1	0	1	1	0	1	164	0	21	0	20	45	0	78

Với số lượng batch trên một cửa sổ trượt bằng 2, ta khởi tạo quần thể như Bảng 6 sau:

Bảng 6. Quần thể sau khi khởi tạo

Chrome (1)	Populations							FI_1 (9)	FI_2 (10)	FIT (11)
	a	b	c	d	e	f	g			
C1	1	0	0	1	0	0	0	0	0	0
C2	0	1	1	0	1	1	0	0	0	0
C3	1	0	0	1	0	1	0	0	0	0
C4	0	1	0	1	1	0	0	0	0	0
C5	0	1	0	0	1	0	1	0	0	0
C6	1	0	1	0	0	0	1	0	0	0

Với cửa sổ trượt thứ nhất W1 của Bảng 5, ta tính fitness cho từng batch cho cột (9) và (10) tương ứng cho từng cá thể trong Bảng 6. Ta xử lý song song đối với các cá thể này. Xét cá thể C1 so với B1 của W1 có FI_1(C1) được tính theo công thức (6) và (7) như sau:

Bảng 7. Tính F1_1(C1)

Chrome (1)	Populations							FI_1 (9)	FI_2 (10)	FIT (11)
	a	b	c	d	e	f	g			
C1	1	0	0	1	0	0	0	70	0	0

Tương tự, ta xét cá thể C_1 so với B_2 của W_1 có $F_{I\_2}(C_1)$, và $f_{1\_W_1}(C_1, W_1)$, thì kết quả như sau:

Bảng 8. Tính fitness cho C1 đối với W1

Chrome (1)	Populations							FI_1 (9)	FI_2 (10)	FIT (11)
	a	b	c	d	e	f	g			
C1	1	0	0	1	0	0	0	70	0	70

Lặp lại lần lượt cho các cá thể còn lại trong W_1. Ta tính được fitness của toàn bộ cá thể so với W_1. Những cá thể nào có fitness >= min $\_util_{W_b}$, lưu vào tập hữu ích cao *huims* của W_1.

Áp dụng thuật toán di truyền trên cửa sổ thứ nhất và quần thể populations, ta cũng thu hoạch được các tập hữu cao của dữ liệu trong cửa sổ này.

Tương tư như thế cho các cửa sổ còn lại. Ta sẽ được các tập hữu ích cao của toàn bộ các cửa sổ *all_huims*.

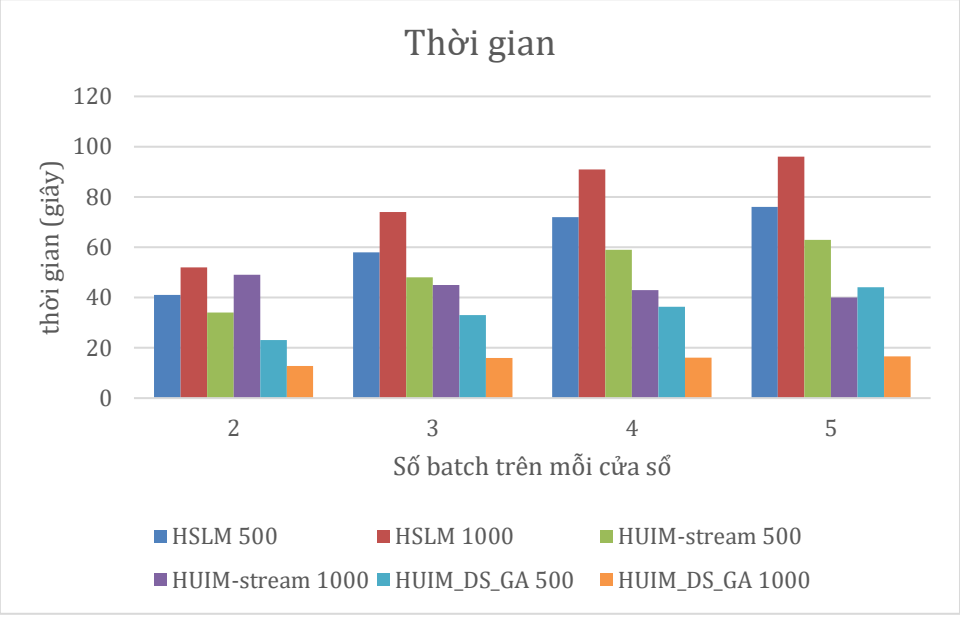
V. ĐÁNH GIÁ THỰC NGHIỆM

Các thử nghiệm được thực hiện trên máy tính CPU 2.90 GHz, 4 nhân và 32 GB bộ nhớ chạy trên hệ điều hành Microsoft Windows 10 64-bit. Chương trình được viết với ngôn ngữ lập trình Python, môi trường cài đặt IDLE (Python 3.11 64-bit). Với các bộ dữ liệu trên website: https://www.philippe-fournier-viger.com/spmf như ở Bảng 9.

Bảng 9. Tập các bộ dữ liệu

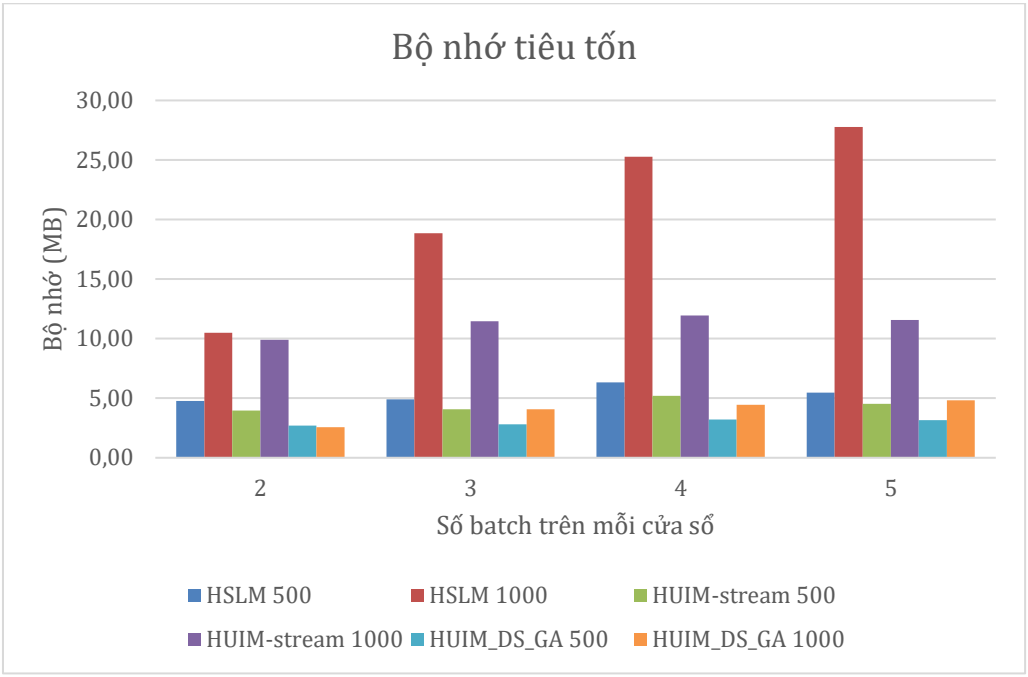
Dataset	Số lượng bit	Số item trung bình	Số dòng	Số transaction/batch
Retail	16470	10	88162	5000/10000
Chess	75	37	3196	190/380
Accidents	468	34	340183	20000/40000
Mushroom	119	23	8416	500/1000

Đầu tiên, bài báo tiến hành so sánh hiệu quả về mặt thời gian thực hiện của thuật toán đề xuất (***HUIM_DS_GA***) với thuật toán ***HSLM*** [9] và thuật toán HUIM-stream [8] trên bộ dữ liệu Mushroom với kích thước cửa sổ trượt khác nhau bằng cách thay đổi số lô trong một cửa sổ và số giao dịch trong một lô. Hình 7 cho thấy sự khác nhau về kích thước của cửa sổ với số lượng lô từ 2 đến 5. Mỗi thuật toán sẽ được thực hiện với kích thước mỗi lô tùy theo số lượng dòng của dataset thể hiện ở cột 5 của Bảng 9. Giá trị ngưỡng *min_util* được chọn là 0.25%. Cùng với các tham số cơ bản của thuật toán di truyền: số thế hệ tối đa (generations) là 20, kích thước quần thể (*population_size*) là 100, xác suất lai ghép là 0.8 và xác suất đột biến là 0.2. Kết quả thực nghiệm cho thấy rằng thuật toán HUIM_DS_GA hiệu quả hơn rất nhiều so với thuật toán HSLM và HUIM-stream do không tốn thời gian phát sinh các tập con các ứng viên. Hoạt động phát sinh mẫu được thực hiện trực tiếp khi phát sinh quần thể của thuật toán đề xuất. Kết quả thực nghiệm cũng cho thấy rằng, khi số lượng giao dịch trong một lô là 500 sẽ tốn nhiều thời gian khai thác hơn do phải thực hiện hoạt động khi trượt sang cửa sổ khác nhiều lần hơn Hình 7.



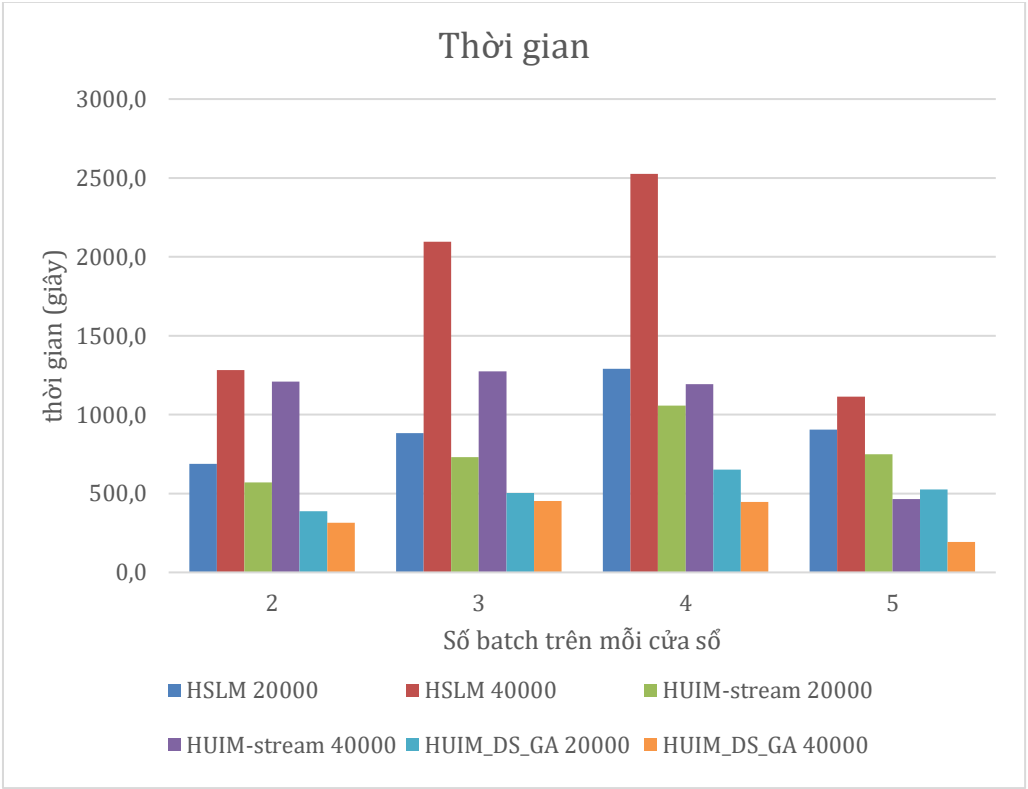
Hình 7. So sánh thời gian thực hiện ở bộ dữ liệu mushroom trên thuật toán HSLM, HUIM-stream và HUIM_DS_GA

Tiếp theo, bài báo thực nghiệm so sánh về bộ nhớ của HSLM, HUIM-stream và HUIM_DS_GA trên bộ dữ liệu của mushroom. Kết quả trong Hình 7 thể hiện GA_BSW_BP tiêu tốn ít bộ nhớ và ổn định hơn do phát sinh một số lượng xác định quần thể và số thế hệ ngay khi số lượng giao dịch trong một lô là 500 và 1000.

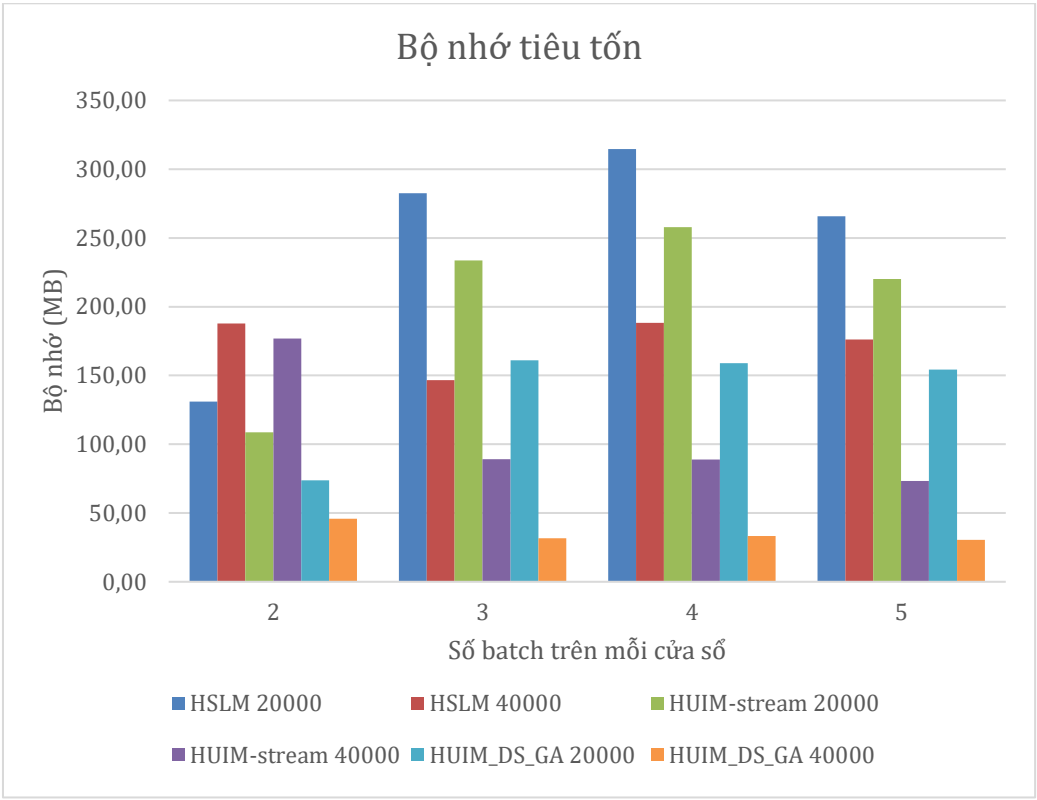


Hình 8. Biểu diễn bộ nhớ tiêu thụ ở bộ dữ liệu mushroom giữa HSLM, HUIM-stream và HUIM_DS_GA

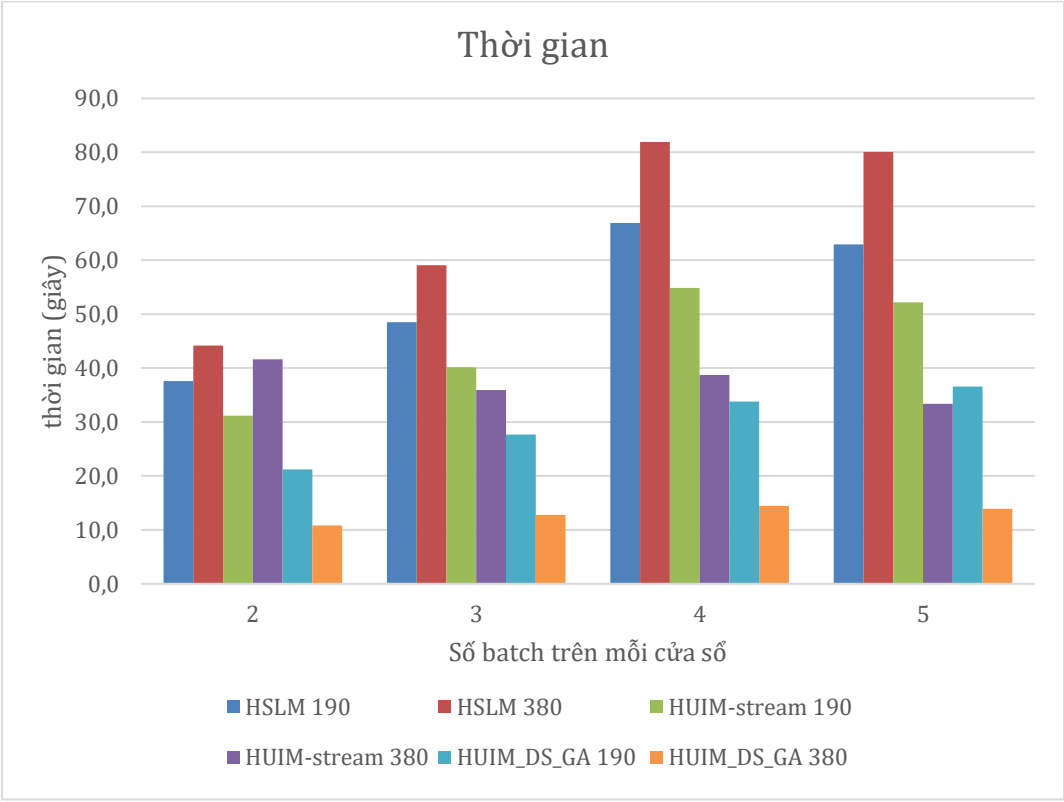
Ngoài bộ dữ liệu mushroom, bài nghiên cứu của chúng tôi còn thực hiện trên bộ dữ liệu accidents ở Hình 9 và Hình 10, chess ở Hình 11 và Hình 12, và Retail ở Hình 13 và 14 đều cho thấy tiêu tốn ít bộ nhớ và thời gian chạy của HUIM_DS_GA nhanh hơn HSLM và HUIM-stream.



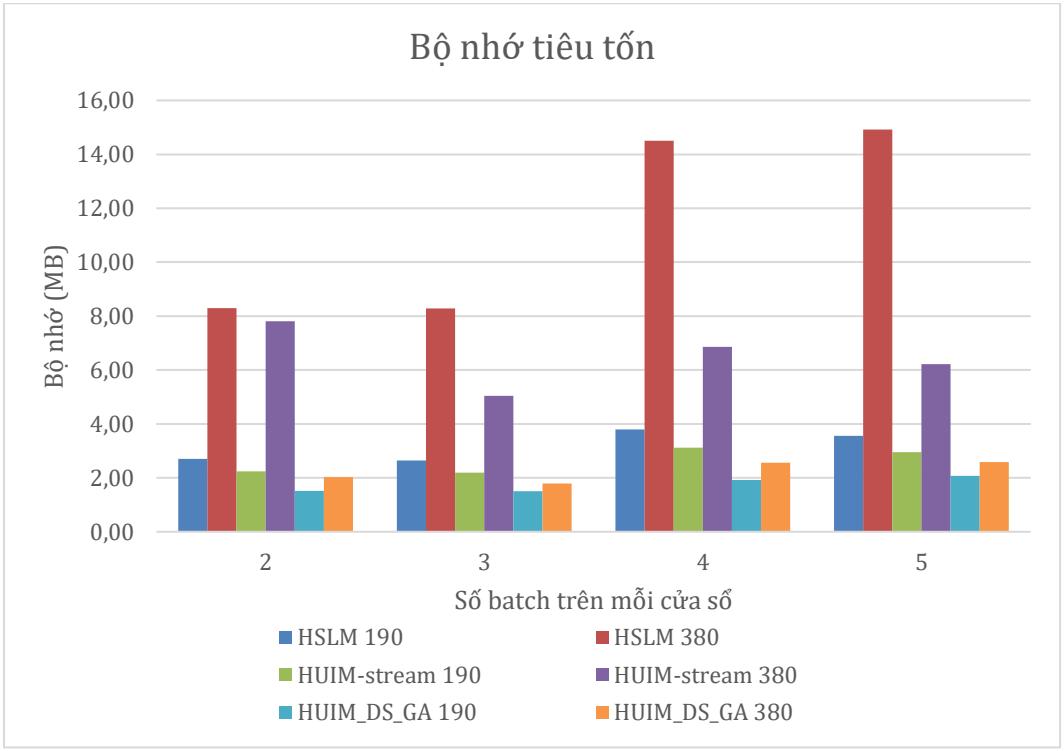
Hình 9. So sánh thời gian thực hiện ở bộ dữ liệu Accidents trên thuật toán HSLM, HUIM-stream và HUIM_DS_GA



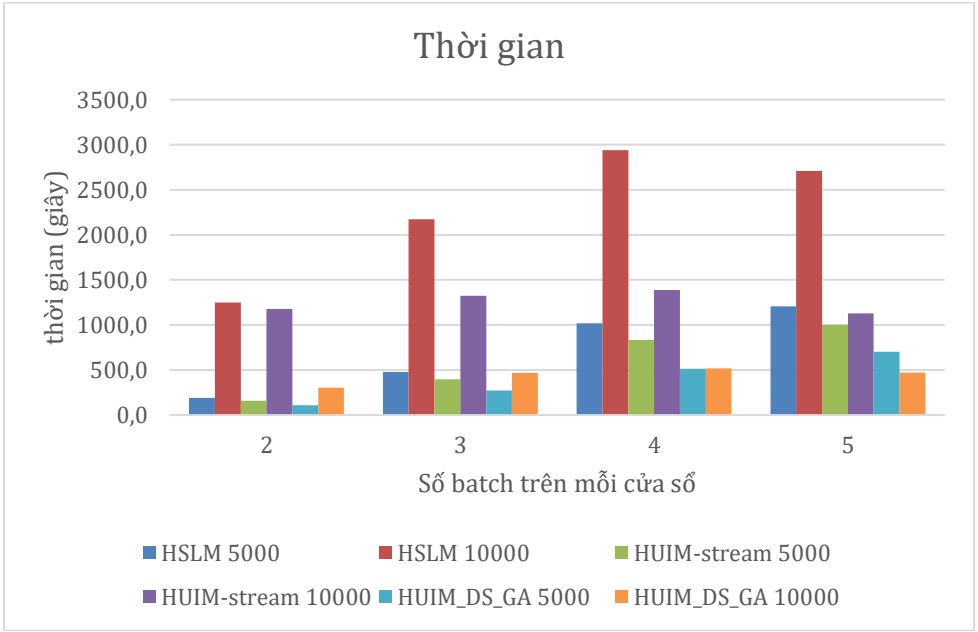
Hình 10. Biểu diễn bộ nhớ tiêu thụ ở bộ dữ liệu Accidents giữa HSLM, HUIM-stream và HUIM_DS_GA



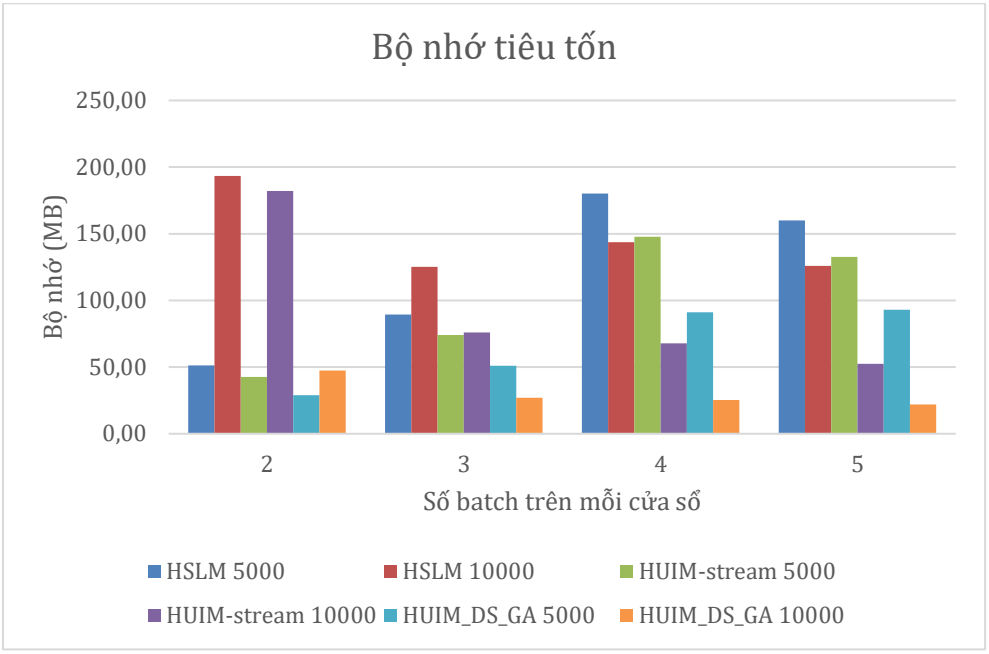
Hình 11. So sánh thời gian thực hiện ở bộ dữ liệu Chess trên thuật toán HSLM, HUIM-stream và HUIM_DS_GA



Hình 12. Biểu diễn bộ nhớ tiêu thụ ở bộ dữ liệu Chess giữa HSLM, HUIM-stream và HUIM_DS_GA



Hình 13. So sánh thời gian thực hiện ở bộ dữ liệu Retail trên thuật toán HSLM, HUIM-stream và HUIM_DS_GA



Hình 14. Biểu diễn bộ nhớ tiêu thụ ở bộ dữ liệu Retail giữa HSLM, HUIM-stream và HUIM_DS_GA

VI. KẾT LUẬN

Bài báo này trình bày một nghiên cứu về khai thác tập hữu ích cao từ dữ liệu luồng. Một phương pháp sử dụng thuật toán di truyền được trình bày và khám phá các mối quan hệ giữa kích thước cửa sổ trượt và các ràng buộc của thuật toán di truyền. **Ngưỡng hữu ích tối thiểu (*min_util*)** được trình bày để tính toán giá trị thích nghi tối thiểu để xác định tập hữu ích cao trong dữ liệu luồng sử dụng cửa sổ trượt.

Chúng tôi áp dụng mảng băm (hashtable) trong việc lưu trữ các cá thể đã được xem xét để giúp cho việc tìm kiếm nhanh chóng và tránh trùng lắp. Ngoài ra còn một cải tiến nữa là dùng ***BitWise*** để thuật toán chạy nhanh hơn cũng như sử dụng không gian bộ nhớ ít hơn.

Trong tương lai, cải tiến thuật toán để áp dụng vào tìm tập hữu ích cao phổ biến (skyline_frequent_utility - SFU) với các ràng buộc khác nhau, cũng như mở rộng khai thác trên dữ liệu có xét đến yếu tố thời gian – khai thác dữ liệu chuỗi. Đồng thời thực xử lý song song để thời gian thực thi của thuật toán nhanh hơn.

VII. TÀI LIỆU THAM KHẢO

[1] Pazhaniraja, N., Sountharrajan, S. and Suganya (2023). "Optimizing high-utility item mining using hybrid dolphin echolocation and Boolean grey wolf optimization," *Ambient Intell Human Comput 14,* p. 2327–2339.

[2] Chowdhury Farhan Ahmed, Syed Khairuzzaman Tanbeer, Byeong-Soo Jeong and Young-Koo Lee (2009). "Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases," *IEEE Transactions on Knowledge and Data Engineering,* pp. vol. 21, no. 12, pp. 1708-1721.

[3] Vincent S. Tseng, Bai-En Shie, Cheng-Wei Wu and Philip S. Yu (2013). "Efficient algorithms for mining high utility itemsets from transactional databases," *IEEE Transactions on Knowledge and Data Engineering,* pp. vol. 25, no. 8, pp. 1772-1786.

[4] Guo, SM., Gao and H. HUITWU (2016). "An Efficient Algorithm for High-Utility Itemset Mining in Transaction Databases," *Journal of Computer Science and Technology,* p. 776–786.

[5] Liu M and Qu J (2012). "Mining high utility itemsets without candidate generation," in Proceedings of the 21st ACM international conference on Information and knowledge management.

[6] WANG S F, HAN M and JIA T (2020). "Survey of high utility pattern mining over data streams," *Application Research of Computers,* pp. 2571-2578.

[7] Bijay Prasad Jaysawal and Jen-Wei Huang (2020). "SOHUPDS: a single-pass one-phase algorithm for mining high utility patterns over a data stream," in *Proceedings of the 35th annual ACM symposium on applied computing*.

[8] Han M, Li M and Chen Z (2023). "High utility pattern mining algorithm over data streams using ext-list," *Applied Intelligence,* pp. 27072-27095.

[9] Minh-Thai Tran, Anh-Duy Tran, Duc-Thanh Pham and Minh-Nguyen Le (2024). "Method for mining high-utility patterns in transaction stream data based on linked list structure," *Journal on Information Technologies & Communications*.

[10] Kannimuthu S and Premalatha K (2014). "Discovery of high utility itemsets using genetic algorithm with ranked mutation," *Applied Artificial Intelligence,* pp. 337-359..

[11] Luna J M, Kiran R U and Fournier-Viger P (2023). "Efficient mining of top-k high utility itemsets through genetic algorithms," *Information Sciences,* pp. 529-553.

[12] Q. Zhang, W. Fang, J. Sun and Q. Wang (2019). "Improved Genetic Algorithm for High-Utility Itemset Mining," *IEEE Access*.

[13] L. J. C. W, G. W and F.-V. P (2016). "High utility-itemset mining and privacy-preserving utility mining," *Perspectives in Science*.

[14] Lin J C W, Djenouri Y, Srivastava G and et al (2021). "A predictive GA-based model for closed high-utility itemset mining," *Applied Soft Computing*.

[15] Lin J C W, Djenouri Y, Srivastava G and et al (2022). "Efficient evolutionary computation model of closed high-utility itemset mining," *Applied Intelligence,* pp. 10604-10616.

[16] Pazhaniraja N, Basheer S and Thirugnanasambandam K (2023). "Multi-objective Boolean grey wolf optimization based decomposition algorithm for high-frequency and high-utility itemset mining," *AIMS Math,* pp. 18111- 18140.

[17] Chen X, Zhai P and Fang Y (2021). "High utility pattern mining based on historical data table over data streams," in *2021 4th International Conference on Data Science and Information Technology*.

[18] Amaranatha Reddy P and Hazarath Murali Krishna Prasad (2021), "High Utility Item-set Mining from retail market data stream with various discount strategies using EGUI-tree," *Journal of Ambient Intelligence and Humanized Computing,* pp. 1-12.

HIGH-UTILITY ITEMSET MINING FROM DATA STREAMS BASED ON GENETIC ALGORITHMS

Le Thi Minh Nguyen, Pham Duc Thanh, Tran Anh Duy

ABSTRACT— High-utility itemset mining (HUIM) from data streams under time and space constraints is a challenging task. Traditional algorithms often scan the data multiple times and utilize complex data structures to mange, store, and update information. Furthermore, itemset loss due to heuristic algorithms and the evaluation of duplicate itemsets generated by regular data batches contribute to the algorithm's inefficiency in time and space. To solve these problems, we propose a new algorithm based on genetic algorithms to deploy high-value itemsets from data streams, called HUIM_DS_GA, which efficiently addresses storage limitations. The HUIM_DS_GA algorithm designs a new cluster update strategy, which increases convergence and minimizes the loss of important itemsets. Additionally, we propose a storage strategy for the hash table to avoid the evaluation of duplicate itemsets, thereby enhancing the algorithm's execution efficiency. Experiments on real and aggregate datasets show that the algorithm performs effectively and significantly decreases the consumption level of memory while maintaining better expansion than previous methods.

Keywords — High Utility Itemset Mining; Data Stream; Hash Table; Genetic Algorithms; Sliding Window.

Trần Anh Duy nhận học vị thạc sĩ Khoa học máy tính Trường Đại học Khoa học tự nhiên năm 2017; hiện là giảng viên khoa Công nghệ thông tin Trường Đại học Ngoại ngữ -Tin học Thành phố Hồ Chí Minh. Lĩnh vực nghiên cứu đang quan tâm là Khai thác dữ liệu.

Lê Thị Minh Nguyện nhận học vị thạc sĩ Khoa học máy tính Đại học Quốc gia Thành phố Hồ Chi Minh năm 2007; hiện là giảng viên khoa Công nghệ thông tin Trường Đại học Ngoại ngữ-Tin học Thành phố Hồ Chí Minh. Lĩnh vực nghiên cứu đang quan tâm là-Khai thác dữ liệu.

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 Thành phố Hồ Chí Minh; lĩnh vực nghiên cứu đang quan tâm là Khai thác dữ liệu.