RESEARCH ARTICLE

KHAI THÁC TẬP PHỔ BIẾN TỪ DỮ LIỆU LUỒNG DỰA TRÊN THUẬT TOÁN DI TRUYỀN SỬ DỤNG BIT VÀ XỬ LÝ SONG SONG

Phạm Đức Thành , Lê Thị Minh Nguyện

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

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

TÓM TẮT— Trong thời đại của dữ liệu lớn, khả năng khai thác thông tin có ý nghĩa từ dữ liệu luồng là vô cùng quan trọng cho các ứng dụng như phân tích thời gian thực, phát hiện bất thường và quá trình ra quyết định. Bài báo này đề xuất một phương pháp mới để khai thác tập phổ biến từ dữ liệu luồng sử dụng thuật toán di truyền kết hợp với các phép toán bit và xử lý song song. Cốt lõi của phương pháp này sử dụng ThreadPoolExecutor từ Python để xử lý song song, tăng tốc độ tính toán đáng kể và cho phép xử lý các luồng dữ liệu lớn một cách hiệu quả. Thuật toán đề xuất sử dụng kỹ thuật cửa sổ trượt để duy trì và cập nhật các tập phổ biến một cách linh hoạt khi dữ liệu mới đến. Phương pháp này đảm bảo rằng phân tích luôn phù hợp với dữ liệu mới nhất, giải quyết các thách thức do tính chất thoáng qua của dữ liệu luồng gây ra. Bằng cách tích hợp các phép toán bit trong thuật toán di truyền, phương pháp này tối ưu hóa việc biểu diễn và thao tác các tập phổ biến, dẫn đến giảm chi phí tính toán và cải thiện hiệu suất. Xử lý song song được thực hiện bằng cách sử dụng ThreadPoolExecutor, cho phép nhiều phân đoạn của luồng dữ liệu được xử lý đồng thời. Điều này không chỉ tăng tốc độ của thuật toán mà còn đảm bảo khả năng mở rộng, làm cho nó phù hợp với môi trường dữ liệu có thông lượng cao. Kết quả thực nghiệm cho thấy phương pháp đề xuất vượt trội so với các kỹ thuật khai thác tập phổ biến truyền thống về cả tốc độ và độ chính xác, đặc biệt trong các tình huống liên quan đến các luồng dữ liệu lớn và liên tục. Chi tiết về triển khai, bao gồm thiết kế của thuật toán di truyền, các phép toán bit và khung xử lý song song, được thảo luận kỹ lưỡng. Ngoài ra, bài báo cung cấp phân tích toàn diện về hiệu suất của thuật toán, nhấn mạnh hiệu quả và tính khả dụng của nó trong các kịch bản dữ liệu luồng thực tế. Phương pháp đề xuất mở ra các hướng đi mới cho việc khai thác dữ liệu luồng thời gian thực, mang lại giải pháp mạnh mẽ và có khả năng mở rộng để trích xuất những thông tin giá trị từ các luồng dữ liệu liên tục.

Từ khóa— Tập phổ biến, dữ liệu luồng, cửa sổ trượt, khai thác luật kết hợp, thuật toán di truyền, khai thác dữ liệu, số giao dịch mỗi lần trượt (batch).

I. GIỚI THIỆU

Khái niệm dữ liệu lớn được đề cập lần đầu tiên trong một bài báo xuất bản năm 1997 [1]. Các tác giả gọi vấn đề xử lý các tập dữ liệu lớn là "vấn đề của dữ liệu lớn". Những tập dữ liệu lớn này có đặc điểm là không vừa với bộ nhớ chính, khiến việc phân tích và trực quan chúng trở nên khó khăn hoặc thậm chí không thể thực hiện được. Ngay cả 10 năm sau, hầu hết các máy tính vẫn không thể tải 100 GB vào bộ nhớ chứ đừng nói đến việc xử lý nó [2]. Trong thời đại dữ liệu được tạo ra với tốc độ cao như hiện nay, thông tin có vai trò quyết định và hầu hết máy tính không thể xử lý lượng dữ liệu khổng lồ; do đó, cần phải tạo ra những cách mới để xử lý dữ liệu. Dữ liệu luồng (data stream) là một chuỗi dữ liệu không ngừng phát sinh và cập nhật theo thời gian thực, chẳng hạn như các bản ghi giao dịch, dữ liệu cảm biến, hoặc các sự kiện mạng xã hội. Đặc điểm của dữ liệu luồng: (i) tốc độ cao: dữ liệu được tạo ra liên tục và với tốc độ cao; (ii) dung lượng lớn: tổng dung lượng dữ liệu có thể rất lớn, vượt quá khả năng lưu trữ truyền thống. (iii) thời gian thực: yêu cầu xử lý và phân tích dữ liệu trong thời gian thực. Trong bài báo này chúng tôi phát triển trên bài nghiên cứu [3] khai thác dữ liệu luồng dựa trên thuật toán di truyền, nhưng ở đây chúng tôi cải tiến dùng kỹ thuật bit và xử lý song song để khai thác.

Khai thác tập phổ biến là một kỹ thuật quan trọng trong khai phá dữ liệu, nhằm tìm ra các tập hợp con của các mục xuất hiện thường xuyên trong một tập dữ liệu. Trong bối cảnh dữ liệu luồng, việc khai thác tập phổ biến trở nên phức tạp hơn do dữ liệu liên tục cập nhật và có dung lượng lớn [4] [5] [6] [7]. Việc khai thác tập phổ biến trong dữ liệu luồng đòi hỏi các kỹ thuật và phương pháp đặc biệt để xử lý đặc điểm tốc độ cao và dung lượng lớn của dữ liệu. Trong môi trường dữ liệu luồng, các thuật toán truyền thống không thể áp dụng trực tiếp do hạn chế về bộ nhớ và yêu cầu xử lý thời gian thực. Do đó, các kỹ thuật như thuật toán trượt cửa sổ (Sliding Window) [9], thuật toán tổng hợp (Synopsis) [8], và thuật toán di truyền (Genetic Algorithm - GA) [9] [10] đã được phát triển để khai thác tập phổ biến hiệu quả từ dữ liệu luồng. Các phương pháp này cho phép tóm tắt và cập nhật liên tục các tập phổ biến, giúp phát hiện kịp thời các mẫu quan trọng. Điều này có ứng dụng rộng rãi trong nhiều lĩnh vực, từ phân tích thị trường và giám sát mạng đến phân tích dữ liệu cảm biến trong các hệ thống IoT. Khai thác tập phổ biến từ dữ liệu luồng không chỉ giúp tiết kiệm tài nguyên mà còn tăng cường khả năng ra quyết định dựa trên dữ liệu thời gian thực.

Trong nghiên cứu này, chúng tôi dựa trên thuật toán di truyền sử dụng bit và xử lý song song. Mỗi bit trong chuỗi đại diện cho sự hiện diện (1) hoặc vắng mặt (0) của một mục trong tập hợp con. Quần thể ban đầu được khởi tạo ngẫu nhiên để đảm bảo tính đa dạng. Độ thích nghi của mỗi cá thể được đánh giá dựa trên tần suất xuất hiện của tập hợp con trong dữ liệu luồng. Việc đánh giá này có thể được thực hiện song song trên các cá thể khác nhau để tăng tốc độ xử lý. Các thao tác chọn lọc, lai ghép và đột biến được thực hiện song song trên các cá thể. Chọn lọc giúp chọn ra các cá thể có độ thích nghi cao để lai ghép, lai ghép kết hợp các cặp cá thể để tạo ra con cái mới, và đột biến giúp duy trì đa dạng di truyền bằng cách thay đổi ngẫu nhiên một số bit trong chuỗi. Quá trình tiến hóa này được lặp lại qua nhiều thế hệ, với các cá thể được tiến hóa song song để tăng tốc độ và hiệu quả.

Sử dụng GA với mã hóa bit và xử lý song song giúp giảm thiểu thời gian tính toán và quản lý hiệu quả bộ nhớ. Các thuật toán song song đảm bảo rằng các kết quả được cập nhật liên tục và chính xác trong thời gian thực, phù hợp với đặc tính của dữ liệu luồng. Phương pháp này cũng có khả năng mở rộng, có thể áp dụng trên các hệ thống đa lõi hoặc hệ thống phân tán để xử lý các tập dữ liệu luồng lớn và phức tạp.

Phần còn lại của bài báo được trình bày như sau. Phần hai trình bày về các công trình liên quan. Để cung cấp nền tảng và ngữ cảnh, trong phần 3 trình bày các khái niệm và định nghĩa, trình bày cách nhìn tổng quan về dữ liệu luồng trong cửa sổ trượt, các hàm thích nghi, lai ghép và đột biến trong di truyền bằng phương pháp bit và xử lý song song. Phần 4 trình bày phương pháp đề xuất. Tiếp theo, trình bày kết quả thực nghiệm về phương pháp đề xuất khác biệt so với thuật toán GA_BSW [3] và thuật toán T_Apriori [11]. Cuối cùng, trình bày phần kết luận và tương lai của đề tài.

II. CÔNG TRÌNH LIÊN QUAN

Việc khai thác tập phổ biến từ dữ liệu luồng là một lĩnh vực nghiên cứu quan trọng và phức tạp, đặc biệt khi kết hợp với các thuật toán di truyền và xử lý song song. Các nghiên cứu gần đây đã tập trung vào việc tối ưu hóa và cải tiến các thuật toán để nâng cao hiệu suất và độ chính xác của quá trình khai thác. Một số nghiên cứu đã phát triển các mô hình và thuật toán mới nhằm xử lý dữ liệu luồng một cách hiệu quả. Ví dụ, nghiên cứu của Tseng và cộng sự [12] đã giới thiệu thuật toán UP-Growth (Utility Pattern Growth) để khai thác các tập mục hữu ích cao (HUIs) từ cơ sở dữ liệu giao dịch. Thuật toán này sử dụng nhiều chiến lược để giảm thiểu không gian tìm kiếm và tối ưu hóa quá trình khai thác HUIs, trong đó bao gồm cả việc sử dụng bit-vector để tối ưu hóa, S. Almeida e Aguiar và cộng sự [13] tối ưu hóa quá trình khai thác bằng cách sử dụng các chiến lược tối ưu hóa bit-vector. Liu và cộng sự [14] đã phát triển một thuật toán hai giai đoạn để nhanh chóng phát hiện các tập mục hữu ích cao từ dữ liệu luồng. Thuật toán này sử dụng các chiến lược tối ưu hóa để giảm thiểu thời gian tính toán và tăng độ chính xác của kết quả. Lin và cộng sự đã áp dụng tối ưu hóa đàn hạt (Particle Swarm Optimization - PSO) [15] để khai thác các tập mục hữu ích cao. Phương pháp này được kết hợp với thuật toán di truyền để nâng cao khả năng tìm kiếm các giải pháp tối ưu trong không gian tìm kiếm rộng lớn. Sự kết hợp giữa PSO và GA đã mang lại kết quả vượt trội về hiệu suất và khả năng khai thác các tập mục hữu ích từ dữ liệu luồng, đặc biệt trong các môi trường dữ liệu phức tạp và biến đổi liên tục. Nghiên cứu của Kannimuthu và Premalatha [16] đã đề xuất việc sử dụng thuật toán di truyền với chiến lược đột biến xếp hạng để khám phá các tập mục hữu ích cao. Naik và Pawar [17] đã phát triển một thuật toán nhanh cho việc khai thác các tập phổ biến đóng từ dữ liệu luồng theo cách thức tăng dần. Thuật toán này sử dụng phương pháp mã hóa bit để tối ưu hóa quá trình khai thác. Công trình này đã cải thiện hiệu quả xử lý và khả năng thích ứng với dữ liệu luồng liên tục thay đổi, đồng thời tiết kiệm bộ nhớ và tài nguyên tính toán.

Các công trình trên không chỉ mở rộng khả năng ứng dụng trong thực tiễn mà còn đặt nền tảng cho các nghiên cứu tiếp theo trong lĩnh vực khai phá dữ liệu. Từ những nghiên cứu này chúng tôi tiếp tục phát triển khai thác tập phổ biến từ dữ liệu luồng dựa trên thuật toán di truyền kết hợp với mã hóa bit và xử lý song song giúp cải thiện hiệu suất và độ chính xác.

III. KHÁI NIỆM VÀ ĐỊNH NGHĨA

A. KHÁI NIỆM

Một CSDL giao dịch D chứa các giao dịch, D = {T_1, T_2, …, T_n}, trong đó n là tổng số các giao dịch trong CSDL. Bảng 1 thể hiện ví dụ về CSDL giao dịch, Bảng 2 thể hiện ánh xạ từ Bảng 1 sang dạng bit. Ví dụ: T_2 = {2, 4, 1, 5} thì ở giao dịch T_2 ở Bảng 2 tại những vị trí {1, 2, 4, 5} bậc lên là 1, còn tại ví trí {3, 6} là 0, cụ thể T_2= {1, 1, 0, 1, 1, 0}.

Bảng 1. *CSDL giao dịch*

T_{id}	Itemset
T_1	2, 4, 6

Bảng 2. *CSDL dạng bit*

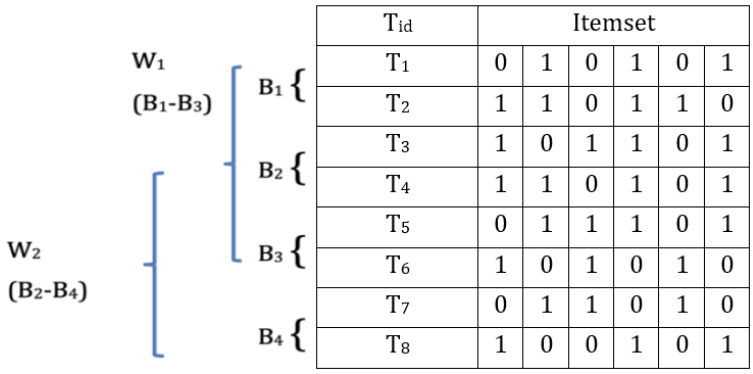
T_{id}	Itemset					
T_1	0	1	0	1	0	1

T_2	2, 4, 1, 5
T_3	6, 4, 1, 3
T_4	2, 6, 4, 1
T_5	2, 6, 4, 3
T_6	1, 3, 5
T_7	2, 3, 5
T_8	1, 4, 6

T_2	1	1	0	1	1	0
T_3	1	0	1	1	0	1
T_4	1	1	0	1	0	1
T_5	0	1	1	1	0	1
T_6	1	0	1	0	1	0
T_7	0	1	1	0	1	0
T_8	1	0	0	1	0	1

1. DỮ LIỆU LUỒNG TRONG GIAO DỊCH

Dữ liệu luồng giao dịch là loại dữ liệu phổ biến trong thực tế. Dữ liệu dạng này có thể thấy trong dữ liệu bán hàng, dữ liệu về các thao tác trên trang web, dữ liệu thời tiết, … Trong đó, số lượng dữ liệu là không giới hạn và có thể tăng trưởng theo thời gian. Một lô (batch) giao dịch gồm nhiều giao dịch. Một cửa sổ (window) bao gồm nhiều lô giao dịch. Hình 1 là một ví dụ minh hoạ về dữ liệu luồng giao dịch. Trong đó, CSDL được chia làm 4 lô có kích thước bằng nhau và 2 cửa sổ. Mỗi lô bao gồm 2 giao dịch. Mỗi cửa sổ gồm 3 lô. Cửa sổ W_1 bao gồm lô B_1, B_2 và B_3. Cửa sổ W_2 bao gồm lô B_2, B_3 và B_4.



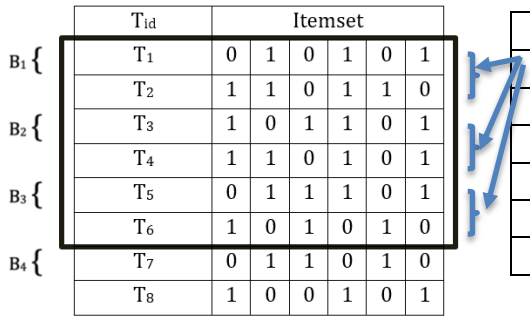
T_{id}	Itemset					
T_1	0	1	0	1	0	1
T_2	1	1	0	1	1	0
T_3	1	0	1	1	0	1
T_4	1	1	0	1	0	1
T_5	0	1	1	1	0	1
T_6	1	0	1	0	1	0
T_7	0	1	1	0	1	0
T_8	1	0	0	1	0	1

Hình 1. *Cửa sổ luồng giao dịch*

2. TÍNH HÀM THÍCH NGHI

Khởi tạo một tập quần thể ngẫu nhiên ban đầu bao gồm nhiều cá thể. Mỗi cá thể là một chuỗi các gene đại diện C_i. Đánh giá chất lượng của mỗi cá thể trong quần thể bằng cách sử dụng một hàm thích nghi. Tính hàm thích nghi F_b_1, F_b_2, F_b_3 của 3 lô trong cửa sổ W_1. Cụ thể tính F_b_1 của lô B_1 bằng tổng của 2 giao dịch trong một lô theo thứ tự các bit {T_1 {0, 1, 0, 1, 0, 1} and C_1 {0, 1, 0, 0 ,1, 0} = 0} + {T_2 {1, 1, 0, 1, 1, 0} and C_1 {0, 1, 0, 0 ,1, 0}=1} =1; F_b2 = {T_3 {1, 0, 1, 1, 0, 1} and C_1 {0, 1, 0, 0 ,1, 0} = 0} + {T_4 {1, 1, 0, 1, 0, 1} and C_1 {0, 1, 0, 0 ,1, 0}=0} =0; F_b3 = {T_5 {0, 1, 1, 1, 0, 1} and C_1 {0, 1, 0, 0 ,1, 0} = 0} + {T_6 {1, 0, 1, 0, 1, 0} and C_1 {0, 1, 0, 0 ,1, 0}=0} =0. Hàm thích nghi Fit của C_1 =Fb_1 + Fb_2 + Fb_3 = 1 + 0 + 0 = 1, tương tự tính các cá thể còn lại từ C_2, C_3, …, C_6 như ở Bảng 3. Sau đó sắp xếp giảm dần theo thứ tự của hàm thích nghi ở Bảng 4. Khi tính hàm thích nghi của 3 lô trong cửa sổ W_1 chúng tôi thực hiện tính song song F_b_1, F_b_2, F_b_3 của 3 lô trong cửa sổ W_1. Cụ thể là tính song song giữa C_1 với B_1, C_1 với B_2 và C_1 với B_3. Tương tự như vậy cho mỗi cá thể C_2, C_3…, C_6.

Bảng 3. *CSDL tính hàm thích nghi*



T_{id}	Itemset					
T_1	0	1	0	1	0	1
T_2	1	1	0	1	1	0
T_3	1	0	1	1	0	1
T_4	1	1	0	1	0	1
T_5	0	1	1	1	0	1
T_6	1	0	1	0	1	0
T_7	0	1	1	0	1	0
T_8	1	0	0	1	0	1

C_i	Population						F_b_1	F_b_2	F_b_3	Fit
C_1	0	1	0	0	1	0	1	0	0	1
C_2	1	0	0	0	1	0	1	0	1	2
C_3	0	0	1	0	0	0	0	1	2	3
C_4	0	1	0	1	0	0	2	1	1	4
C_5	0	0	0	1	0	1	1	2	1	4
C_6	0	0	1	1	1	0	0	0	0	0

Hình 2. *Luồng giao dịch với cửa sổ W1*

Bảng 4. *Hàm thích nghi sau khi sắp xếp giảm dần*

C_i	Population						F_b1	F_b2	F_b3	Fit
C_1	0	1	0	1	0	0	2	1	1	4
C_2	0	0	0	1	0	1	1	2	1	4
C_3	0	0	1	0	0	0	0	1	2	3
C_4	1	0	0	0	1	0	1	0	1	2
C_5	0	1	0	0	1	0	1	0	0	1
C_6	0	0	1	1	1	0	0	0	0	0

Cho tập nhiễm sắc thể chro_exists = {C_1, C_2, C_3, C_4, C_5, C_6} và tập phổ biến có độ hỗ trợ bằng 2 thì tập phổ biến frequent_set = {C_1, C_2, C_3, C_4} theo Bảng 4.

3. LAI GHÉP

Lai ghép giữa hai nhiễm sắc thể liên quan đến sự kết hợp các gen từ hai nguồn khác nhau để tạo ra một bộ nhiễm sắc thể mới với các đặc điểm mong muốn. Cụ thể lấy cặp nhiễm sắc thể C_2 và C_5 lai ghép với nhau như Hình 3. Kết quả sau khi lai ghép ở Hình 4 bằng cách đảo 3bit đánh dấu của C_2 xuống C_5 và ngược lại. Tập nhiễm sắc thể cập nhật mới là chro_exists = {C_1, C_2, C_3, C_4, C_5, C_6, C_{25}, C_{52}} và frequen_set = {C_1, C_2, C_3, C_4, C_{25}, C_{52}} thể hiện ở Bảng 5.

C_2	0	0	0	1	0	1
C_5	0	1	0	0	1	0

Hình 3. *Bộ nhiễm sắc thể trước khi lai ghép*

C_{25}	0	0	0	0	1	0
C_{52}	0	1	0	1	0	1

Hình 4. *Bộ nhiễm sắc thể sau khi lai ghép*

Bảng 5. *CSDL tính hàm thích nghi sau khi lai ghép giữa C_{25} và C_{52}*

C_i	Population						F_b1	F_b2	F_b3	Fit
C_1	0	1	0	1	0	0	2	1	1	4
C_2	0	0	0	1	0	1	1	2	1	4
C_3	0	0	1	0	0	0	0	1	2	3
C_4	1	0	0	0	1	0	1	0	1	2
C_5	0	1	0	0	1	0	1	0	0	1
C_6	0	0	1	1	1	0	0	0	0	0
C_{25}	0	0	0	0	1	0	1	0	1	2
C_{52}	0	1	0	1	0	1	1	1	1	3

4. ĐỘT BIẾN

Đột biến là quá trình thay đổi 1 hoặc nhiều gene trong một nhiễm sắc thể để tạo ra các giải pháp mới và đa dạng hơn, cụ thể ở đây chúng tôi đột biến nhiễm sắc thể C_4. Trước khi đột biến C_4 = {1, 0, 0, 0, 1, 0}, nếu đột biến vị trí thứ 3 tính từ trái sang nghĩa là thay đổi giá trị 0 thành giá trị 1, vậy nhiễm sắc thể C_{41} sẽ là {1, 0, 1, 0, 1, 0} Bảng 6. Tập nhiễm sắc thể cập nhật mới cho chro_exists = {C_1, C_2, C_3, C_4, C_5, C_6, C_{25}, C_{52}, C_{41}} và tập phổ biến frequen_set = {C_1, C_2, C_3, C_4, C_{25}, C_{52}} thể hiện ở Bảng 6.

Bảng 6. *CSDL tính hàm mục tiêu sau khi đột biến*

C_i	Population						F_b1	F_b2	F_b3	Fit
C_1	0	1	0	1	0	0	2	1	1	4
C_2	0	0	0	1	0	1	1	2	1	4
C_3	0	0	1	0	0	0	0	1	2	3
C_4	1	0	0	0	1	0	1	0	1	2
C_5	0	1	0	0	1	0	1	0	0	1
C_6	0	0	1	1	1	0	0	0	0	0
C_{25}	0	0	0	0	1	0	1	0	1	2
C_{52}	0	1	0	1	0	1	1	1	1	3
C_{41}	1	0	1	0	1	0	0	0	1	1

Chọn lọc quần thể mới với pop_size = 6 thể hiện ở Bảng 7. Trượt sang cửa sổ (W_2) thứ 2 Hình 5, xóa lô cũ B_1, thêm vào lô mới B_4, cập nhật lại bộ nhiễm sắc thể chro_exists = {C_1, C_2, C_3, C_4, C_5, C_6} và tập phổ biến frequen_set = {C_1, C_2, C_3, C_4, C_5, C_6}. Tiếp tục thao tác như cửa sổ (W_1)

	T_{id}	Itemset					
B_1	T_1	0	1	0	1	0	1
B_1	T_2	1	1	0	1	1	0
B_2	T_3	1	0	1	1	0	1
B_2	T_4	1	1	0	1	0	1
B_3	T_5	0	1	1	1	0	1
B_3	T_6	1	0	1	0	1	0
B_4	T_7	0	1	1	0	1	0
B_4	T_8	1	0	0	1	0	1

Hình 5. *Luồng giao dịch với cửa sổ W_2*

Bảng 7. *Quần thể mới với độ hỗ trợ bằng 2*

C_i	Population						F_b1	F_b2	F_b3	Fit
C_1	0	1	0	1	0	0	2	1	1	4
C_2	0	0	0	1	0	1	1	2	1	4
C_3	0	0	1	0	0	0	0	1	2	3
C_4	0	1	0	1	0	1	1	1	1	3
C_5	1	0	0	0	1	0	1	0	1	2
C_6	0	0	0	0	1	0	1	0	1	2

B. ĐỊNH NGHĨA [3]

Gọi $I = \{a_1, a_2, \dots, a_m\}$ là một tập các hạng mục, và cơ sở dữ liệu (CSDL) giao dịch $DB = \langle T_1, T_2, \dots, T_n \rangle$, trong đó T_i ($i \in (1..n)$ là một giao dịch chứa một tập các hạng mục trong I.

Định nghĩa 1. Độ hỗ trợ được định nghĩa là các hạng mục được tìm thấy với tần suất tối thiểu trong toàn bộ tập dữ liệu. Độ hỗ trợ (hoặc tần suất xuất hiện) của một mẫu A, trong đó A là một tập con của I, là số lượng giao dịch chứa A trong DB. Công thức (1) tính độ *support* như sau:

$$support(A) = \frac{n(A)}{|DB|} \quad (1)$$

Trong đó: $n(A)$ là số lượng giao dịch chứa A trong DB

Định nghĩa 2. Một tập phổ biến là một tập mục đáp ứng ngưỡng hỗ trợ tối thiểu. Một mẫu A được coi là phổ biến nếu độ hỗ trợ của A không nhỏ hơn một ngưỡng hỗ trợ tối thiểu *min_sup* được xác định trước, ξ. Công thức (2) tính tập phổ biến như sau:

$$support(A) \geq \xi \text{ x } |DB| \quad (2)$$

Định nghĩa 3. Ta có một tập mẫu A là tập con của I. Đối với dữ liệu luồng, sử dụng cửa sổ trượt, với kích thước cửa sổ trượt là *win_size*, ta có ***support_count*** để xác định tập phổ biến với công thức tính (3) là:

$$support\_count(A) = (win\_size * support(A)) \quad (3)$$

Giá trị support_count của thuật toán di truyền được tính theo công thức (4) như sau:

$$support\_count\_GA(A) = (win\_size * support(A) * confidence(A)) \quad (4)$$

IV. KHAI THÁC CÁC TẬP PHỔ BIẾN DỮ LIỆU LUỒNG VỚI THUẬT GIẢI DI TRUYỀN

A. THUẬT TOÁN ĐỀ XUẤT:

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 $databitset$, độ hỗ trợ $support\_count$, tập phổ biến frequent_itemsets_in
Output Các tập phổ biến frequent_itemsets_out

```
1.  frequent_itemsets_out ← frequent_itemsets_in
2.  for generation ← 0 to generations do
3.      j ← 0
4.      while j < population_size/3 do
5.          if random.random() < crossover_probability then
6.              parent1, parent2 ← randomSelected(population)
7.              child1, child2 ← crossover(parent1, parent2, frequent_itemsets_out)
8.              population. append(child1); population.append(child2)
9.          end if
10.         if random.random() < mutation_probability then
11.             child_to_mutate ← randomSelected(population)
12.             mutated_child ← mutate (child_to_mutate, frequent_itemsets_out)
13.             population.append(mutated_child)
14.         end if
15.         j ← j + 1
16.     end while
17.     sorted_population ← Sorted(population)
18.     population ← sorted_population[:population_size]
19. end for
20. Return frequent_itemsets_out
```

Dòng 1: Gán giá trị của tập phổ biến vào (frequent_itemsets_in) cho tập phổ biến ra (frequent_itemsets_out).

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

Dòng 3 đến 16: Lặp với số lần n= (population_size/3).

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, nếu 2 cá thể con là phổ biến thì đưa vào tập phổ biến frequent_itemsets_out.

Dòng 8: Đưa 2 cá thể con mới vừa tạo vào quần thể population.

Dòng 10: Nếu con số phát sinh ngẫu nhiên nhỏ hơn ngưỡng xác suất đột biến mutation_probability thì.

Dòng 11: Chọn ngẫu nhiên 1 cá thể con trong quần thể.

Dòng 12: Thực hiện việc đột biến cá thể này. Nếu cá thể con vừa đột biến là phổ biến thì đưa vào tập phổ biến frequent_itemsets_out.

Dòng 13: Đưa cá thể con vừa đột biến vào quần thể population.

Dòng 17: Sắp xếp quần thể theo độ thích nghi fitness của từng cá thể giảm dần.

Dòng 18: Sao chép quần thể cho lần lặp tiếp theo từ quần thể đã được sắp sorted_population với số lượng là pop_size.

Dòng 20: Trả về kết quả tìm được là tập phổ biến frequent_itemsets_out.

2. THUẬT TOÁN GA_BSW_BP (GENETIC ALGORITHM WITH BATCH SLIDING WINDOW BIT PARALLEL).

Algorithm 2 ***GA_BSW_BP***
Input Dữ liệu $databitset$, kích thước quần thể $population\_size$, số batch của một cửa sổ trượt $batchNumberPerwindow$, độ hỗ trợ $support\_count$, kích thước của một lô $batch\_size$.

Output Danh sách lưu các tập phổ biến của mỗi lần trượt *all_frequent_itemsets*

```
1.  all_frequent_itemsets ← ∅
2.  window_size ← batch_size * batchNumberPerwindow
3.  window_data ← ∅
4.  i ← 0
5.  for j ← 0 to batchNumberPerwindow - 1 do
6.          batch_data ← databitset[i:i+batch_size]
7.          i ← i + batch_size
8.          window_data ← window_data + batch_data
9.  end for
10. population, frequen_sets ← InitPopulation_BitArray(population_size,window_data, support_count)
11. population, frequen_sets ← GA_obj.GA_Run()
12. all_frequent_itemsets ← all_frequent_itemsets + frequen_sets
13. while i <= len(databitset) − batch_size do
14.         batch_data ← databitset[i:i+batch_size]
15.         i ← i + batch_size
16.         window_data.pop(0)
17.         window_data ← window_data + batch_data
18.         GA_obj.UpdatePopulation(batch_data)
19.         population, frequen_sets ← GA_obj.GA_Run()
20.         all_frequent_itemsets ← all_frequent_itemsets + frequen_sets
21. end while
22. Return all_frequent_itemsets
```

Dòng 1: Khởi tạo danh sách tất cả các tập phổ biến 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 databitset.

Dòng 5 -> 9: Duyệt số batch trong 1 cửa sổ trượt, để tạo cửa sổ trượt lần đầu tiên.

Dòng 6: Lấy 1 lần số lượng transaction (batch_size) gán cho batch_data.

Dòng 7: tăng i thêm một số lượng là batch_size.

Dòng 8: Thêm batch_data vào cho window_data.

Dòng 10: 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 11: Gọi thực thi hàm GA_Run() với hai kết quả trả về là population và frequen_sets.

Dòng 12: Lưu frequen_sets vào trong all_frequent_itemsets

Dòng 13 -> 22: Thực hiện việc lặp lần lượt qua các cửa sổ trượt tiếp theo và thực hiện các công việc sau:

Dòng 14: Đọc một batch_data mới.

Dòng 15: Tăng thêm batc_size cho i.

Dòng 16: Xóa batch đầu tiên (cũ) ra khỏi cửa sổ trượt (window_data) đang xét.

Dòng 17: Đưa batch mới thêm vào sau của cửa sổ trượt.

Dòng 18: Cập nhật lại quần thể do batch_data mới được thêm vào cửa sổ trượt.

Dòng 19: Gọi thực thi hàm GA_Run() với hai kết quả trả về là population và frequen_sets.

Dòng 20: Lưu frequen_sets vào trong all_frequent_itemsets

Dòng 22: Trả về all_frequent_itemsets

B. VÍ DỤ MINH HOẠ:

[1] Cho một CSDL giao dịch:

Bảng 8. *CSDL giao dịch với lần trượt thứ nhất*

T_{id}	Itemset
T_1	2, 4, 6
T_2	2, 4, 1, 5
T_3	6, 4, 1, 3
T_4	2, 6, 4, 1
T_5	2, 6, 4, 3
T_6	1, 3, 5
T_7	2, 3, 5
T_8	1, 4, 6

Giải thuật di truyền được đề xuất sử dụng ngưỡng bao gồm tập phổ biến của giải thuật di truyền là support_count = (kích thước cửa sổ trượt * support * confidence). Cho kích thước cửa sổ trượt win_size=5, support=0.5 và confidence=0.6. Xét cửa sổ trượt đầu tiên từ T_1 đến T_5. Ta có: support_count là: 5 * 0.5 * 0.6=1.5.

Khi từng giao dịch được xử lý, sẽ được ghi nhận số lần xuất hiện của mỗi hạng mục (item) trong các giao dịch, và những hạng mục không đáp ứng ngưỡng support_count tối thiểu sẽ bị loại bỏ. Trong ví dụ này, Mục 5 bị loại bỏ không đưa vào trong việc khởi tạo gene. Với số lượng hạng mục (item) ứng viên ánh xạ trong bộ dữ liệu của cửa sổ trượt này là năm, bước đầu tiên là tạo một quần thể ngẫu nhiên gồm $2^{(5-1)}$ hoặc 16 cá thể, mỗi cá thể có 5 gene và mỗi gene ánh xạ tới một trong các mục có thể có.

Một số cá thể được trình bày trong Bảng 9, với các ứng viên ánh xạ [1, 2, 3, 4, 6] cho lần trượt thứ nhất như hình ở Bảng 8.

Bảng 9. *Ví dụ về thuật toán di truyền*

Cá thể (1)	**Gồm ứng viên ánh xạ (2)**	**Kiểm tra sự thích nghi (3)**	**Chọn lọc để lai ghép (4)**	**Cá thể mới (5)**	**Đột biến cá thể mới (6)**	**Gồm (7)**	**Độ thích nghi (8)**
[01101]	[2, 3, 6]	±	Cá thể 2,3	[0**100**1] Con 1	[010***0***1]	[2, 6]	3
[00011]	[4, 6]	4	Cá thể 2,3	[1**001**0] Con 2	[10010] Không có gen đột biến	[1, 4]	3
[11000]	[1, 2]	2	Cá thể 2,4	[00011] Không lai ghép. Xác suất không vượt quá ngưỡng, cá thể 2 được chuyển sang thế hệ tiếp theo	[0***0***011]	[4, 6]	4
[01010]	[2, 4]	4	Cá thể 2,4	[01010] Không lai ghép. Xác suất không vượt quá ngưỡng, cá thể 4 được chuyển sang thế hệ tiếp theo	[01010] Không có gen đột biến	[2, 4]	4

Trong Bảng 9, cột 1 (cá thể) biểu diễn quần thể ban đầu được khởi tạo gồm 4 cá thể {01101, 00011, 11000, 01010} bằng cách phát sinh ngẫu nhiên 1 hoặc 0, với 1 là mục ánh xạ tương ứng có xuất hiện, 0 thì ngược lại. Cột 2 gồm các mục tương ứng với số 1 bên cột 1. Cột 3 tính ***support*** của từng cá thể ở cột 1 với lần lượt các giá trị là: 1, 4, 2, 4. Với support_count = 1.5 vừa tính ở trên thì cá thể đầu tiên bị loại. Ta còn lại 3 cá thể thứ hai, ba và tư. Cột 4 sau khi chọn lọc 3 cá thể còn lại, ta lai ghép các cặp như sau: 2-3, 2-4. Việc lai ghép cũng được chọn ngẫu nhiên và theo xác suất lai ghép cho trước. Nếu không vượt ngưỡng xác suất thì không lai ghép. Cột (5) biểu diễn các cá thể con đã lai ghép hoặc các quần thể trước đó. Cột (6) biểu diễn việc đột biến cá thể mới với xác suất đột biến cho trước và thực hiện tương tự như lai ghép. Cột (7) thể hiện các mục ánh xạ tương ứng như cột 2. Cột (8) tính support như cột (3).

Quá trình trên được lặp lại cho đến khi đạt được một điều kiện dừng nào đó. Điều kiện dừng này bao gồm số thế hệ tối đa hoặc một điều kiện về sự thích nghi. Thường sẽ xác định trước số lần lặp (số thế hệ). Ta ghi nhận được các tập phổ biến cho lần trượt thứ nhất này.

Tiếp theo, ta trượt cửa sổ cho lần trượt thứ hai với batch = 3. Bảng 10 minh hoạ cửa sổ trượt lần hai như sau:

Bảng 10. *CSDL với lần trượt thứ hai*

T_{id}	Itemset
T_1	2, 4, 6
T_2	2, 4, 1, 5
T_3	6, 4, 1, 3
T_4	2, 6, 4, 1
T_5	2, 6, 4, 3
T_6	1, 3, 5
T_7	2, 3, 5
T_8	1, 4, 6

Thực hiện thuật toán di truyền tương tự như lần trượt cửa sổ thứ nhất, ta cũng thu hoạch được các tập phổ biến của dữ liệu trong cửa sổ này.

Bài báo này, chúng tôi chỉ có tạo ra các tập phổ biến, không hiển thị các luật kết hợp.

V. ĐÁNH GIÁ THỰC NGHIỆM

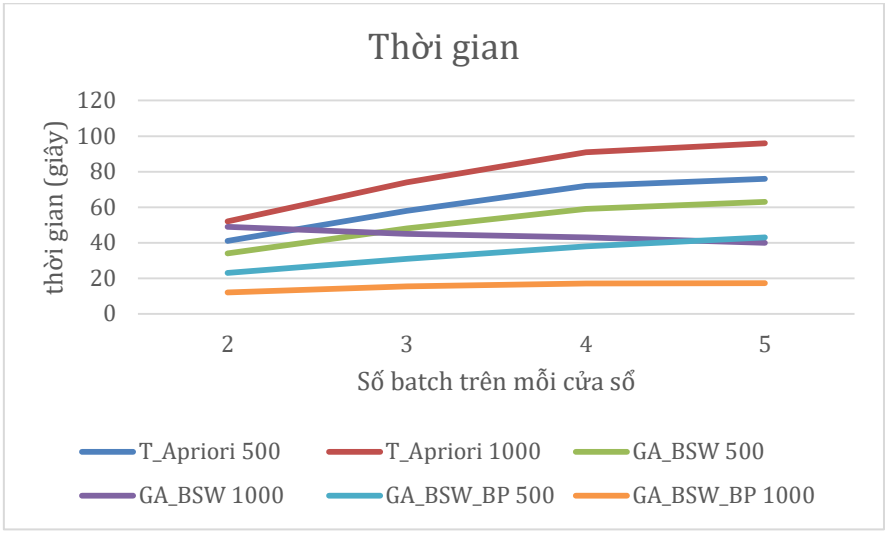
Các thử nghiệm được thực hiện trên máy tính CPU 2.90 GHz, 4 nhân và 32 GB bộ nhớ chạy trên hệ điều hành Microsoft Windows 10 64-bit. Chương trình được viết với ngôn ngữ lập trình Python, môi trường cài đặt IDLE (Python 3.11 64-bit). Với các bộ dữ liệu ở bảng 11.

Bảng 11. Tập các bộ dữ liệu

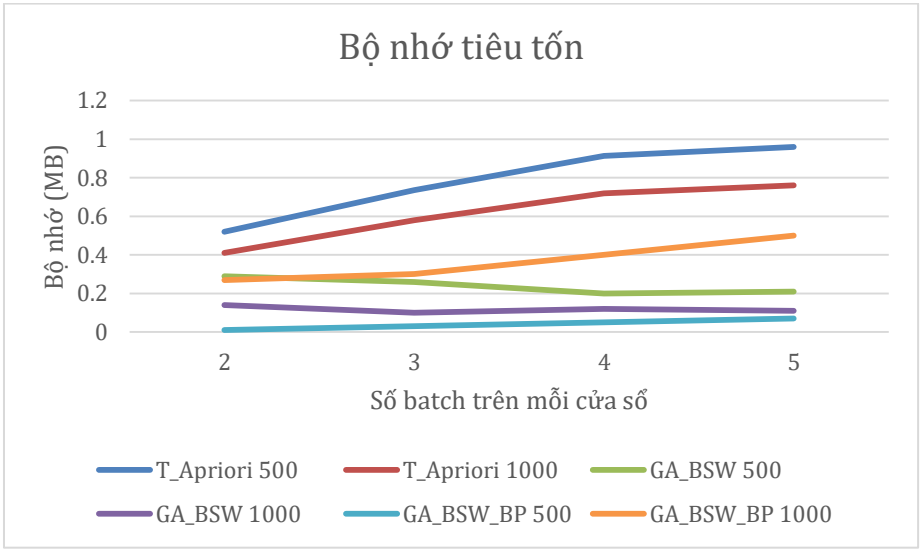
Dataset	Số lượng bit	Số item trung bình	Số dòng	Số transaction/batch
OnlineRetail	2603	4	540455	32000/64000
Chess	75	37	3196	190/380
Accidents	468	33	340183	20000/40000
Mushroom	128	23	8416	500/1000

Đầu tiên, bài báo tiến hành so sánh hiệu quả về mặt thời gian thực hiện của thuật toán đề xuất (***GA_BSW_BP***) với thuật toán ***GA_BSW*** [3] và thuật toán T_Apriori [11] 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 6 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 11. Giá trị ngưỡng support_count 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 GA_BSW_BP hiệu quả hơn rất nhiều so với thuật toán T_Apriori và GA_BSW 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 6.

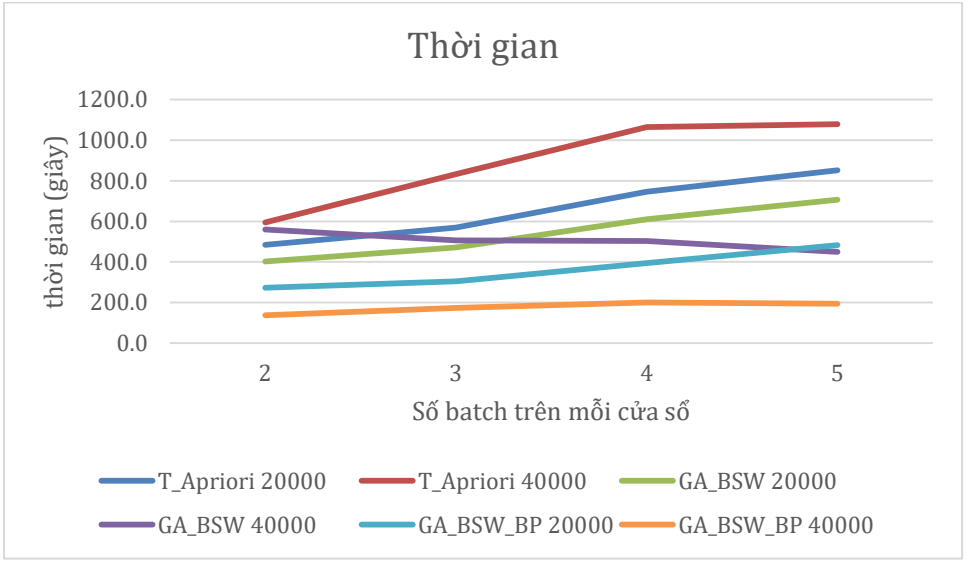
Tiếp theo, bài báo thực nghiệm so sánh về bộ nhớ của GA_BSW_BP, GA_BSW và T_Apriori trên bộ dữ liệu của mushroom. Kết quả trong Hình 7 thể hiện GA_BSW_BP tiêu tốn ít bộ nhớ và ổn định hơn do phát sinh một số lượng xác định quần thể và số thế hệ ngay khi số lượng giao dịch trong một lô là 500 và 1000.



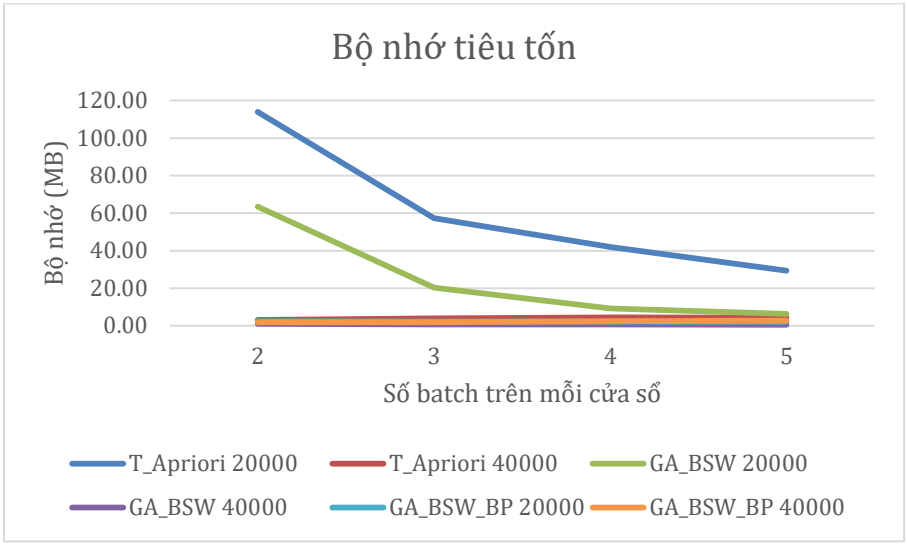
Hình 6. *So sánh thời gian thực hiện ở bộ dữ liệu mushroom trên thuật toán T_Apriori, GA_BSW và GA_BSW_BP*



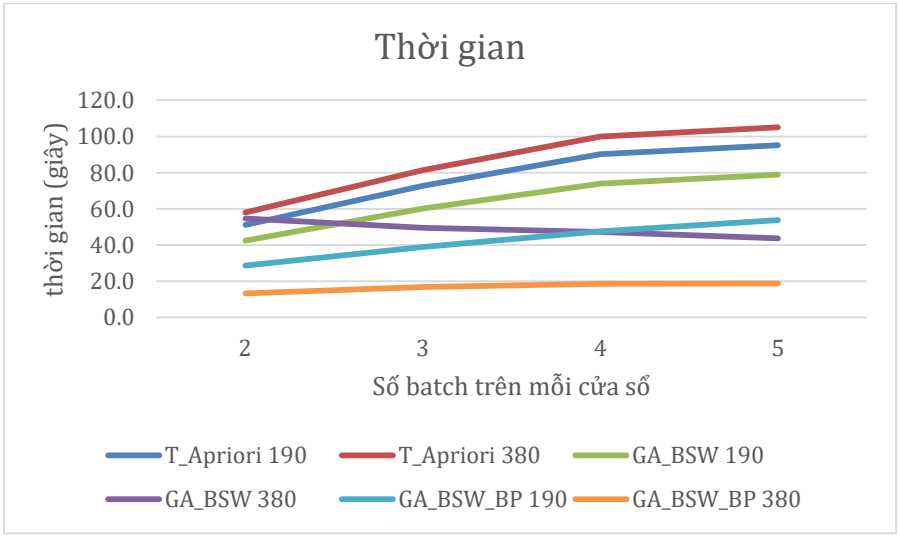
Hình 7. *Biểu diễn bộ nhớ tiêu thụ ở bộ dữ liệu mushroom giữa T_Apriori, GA_BSW và GA_BSW_BP*



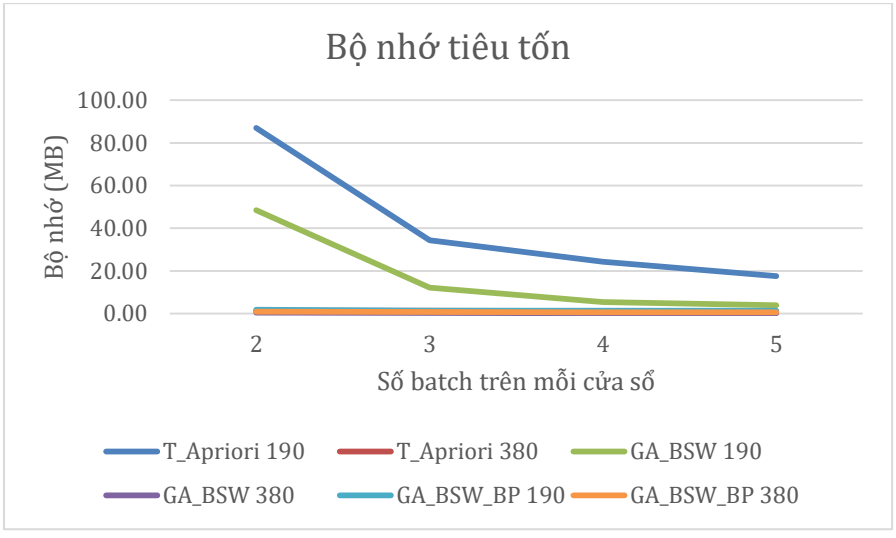
Hình 8. *So sánh thời gian thực hiện ở bộ dữ liệu Accidents trên thuật toán T_Apriori, GA_BSW và GA_BSW_BP*



Hình 9. *Biểu diễn bộ nhớ tiêu thụ ở bộ dữ liệu Accidents giữa T_Apriori, GA_BSW và GA_BSW_BP*

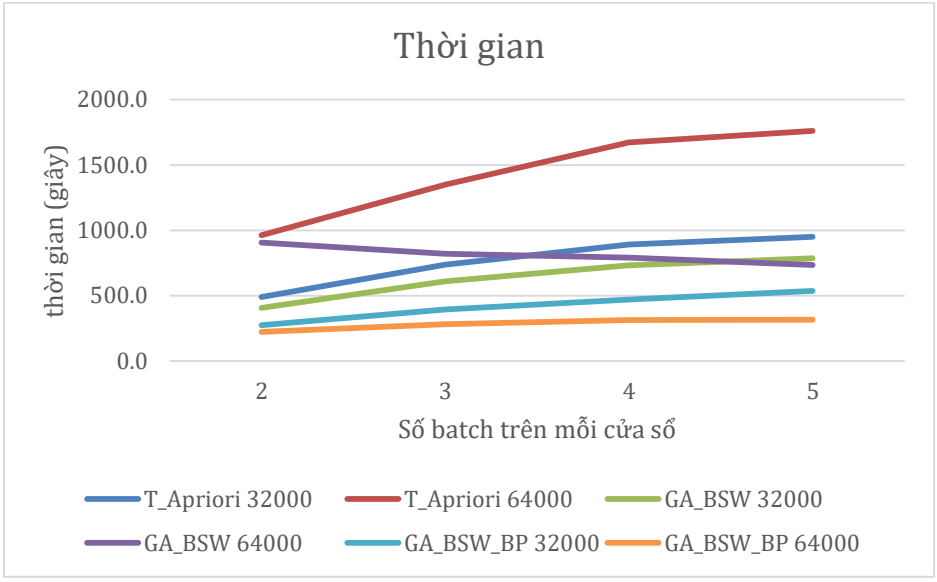


Hình 10. *So sánh thời gian thực hiện ở bộ dữ liệu Chess trên thuật toán T_Apriori, GA_BSW và GA_BSW_BP*



Hình 11. *Biểu diễn bộ nhớ tiêu thụ ở bộ dữ liệu Chess giữa T_Apriori, GA_BSW và GA_BSW_BP*

Ngoài bộ dữ liệu mushroom, bài nghiên cứu của chúng tôi còn thực hiện trên bộ trên bộ dữ liệu accidents ở Hình 8 và Hình 9, chess ở Hình 10 và Hình 11, và OnlineRetail ở Hình 12 và 13 đều cho thấy tiêu tốn ít bộ nhớ và thời gian chạy của GA_BSW_BP nhanh hơn T_Apriori và GA_BSW.



Hình 12. *So sánh thời gian thực hiện ở bộ dữ liệu OnlineRetail trên thuật toán T_Apriori, GA_BSW và GA_BSW_BP*

Hình 13. *Biểu diễn bộ nhớ tiêu thụ ở bộ dữ liệu OnlineRetail giữa T_Apriori, GA_BSW và GA_BSW_BP*

VI. KẾT LUẬN

Bài báo này trình bày một nghiên cứu về khai thác tập phổ biến 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. Độ hỗ trợ (Support) được trình bày để tính toán số lần xuất hiện tối thiểu để xác định tập phổ biến trong dữ liệu luồng sử dụng cửa sổ trượt.

Bài báo đã thực hiện hai cải tiến xử lý song song để thời gian thực thi của thuật toán nhanh hơn. Thứ nhất, thuật toán di truyền có thể được đa luồng để cho phép dòng dữ liệu tiếp tục trong khi các tập phổ biến đang được tạo ra, bằng cách xử lý từng lô (batch) vào từng tiểu trình. Thứ hai, có thể song song hóa hàm thích nghi khi tính support_count của từng ứng viên với các transactions trong cửa sổ trượt. Ngoài ra còn một cải tiến nữa là dùng ***BitWise*** để thuật toán chạy nhanh hơn.

Trong tương lai, cải tiến thuật toán để áp dụng vào tìm tập hữu ích cao, tập hữu ích cao phổ biến (skyline_frequent_utility - SFU) với các ràng buộc khác nhau.

VII. TÀI LIỆU THAM KHẢO

[1] M. Cox and D. Ellsworth (1997), "Application-controlled demand paging for out-of-core visualization.," in *In: Proceedings of the 8th Conference on Visualization '97. VIS '97, pp. 235–244. IEEE Computer Society Press, Washington, DC, USA, https://doi.org/10.1109/VISUAL*.

[2] Almeida, Brás and Sargento (2023), "Time series big data: a survey on data stream frameworks, analysis and algorithms," *in Journal of Big Data. 10:83 https://doi.org/10.1186/s40537-023-00760-1*.

[3] Phạm Đức Thành and Lê Thị Minh Nguyện (2024), "Khai thác tập phổ biến từ dữ liệu luồng bằng cách sử dụng thuật toán di truyền.," *Tạp chí Huflit,* vol. Tập 8 số 1.

[4] Conor McArdle, Xiaojun Wang, J. Deng, Z. G. Qu and X.X. Niu (2013), "Frequent Itemset Mining Over Stream Data: Overview," *Conference: IET International Conference on Information and Communications Technologies. DOI:10.1049/cp.2013.0032*.

[5] Syed Khairuzzaman Tanbeer, Chowdhury Farhan Ahmed, Byeong-Soo Jeong and Young-Koo Lee (2009), "Sliding window-based frequent pattern mining over data streams," *Information sciences,* vol. Volume 179, no. 22, pp. 3843-3865.

[6] L. Bhuvanagiri, Sumit Ganguly, Deepanjan Kesh and Chandan Saha (2006), "Simpler algorithm for estimating frequency moments of data streams," in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, DOI:10.1145/1109557.1109634*.

[7] Khan, Bahga and Madisetti (2016), "A Survey of Genetic Algorithms for Finding Frequent Itemsets in Data Streams," *IEEE Transactions on Knowledge and Data Engineering*.

[8] S. Ramírez-Gallego, Bartosz Krawczyk, Michał Woźniak and Francisco Herrera (2017), "A survey on data preprocessing for data stream mining: Current status and future directions," *Neurocomputing,* vol. 239, pp. 39-57.

[9] C. Han, Lijuan Sun, Jian Guo and Xiaodong Chen (2013), "Mining Frequent Itemsets in Data Streams using a Genetic Algorithm," *2013 15th IEEE International Conference on Communication Technology, Guilin. doi: 10.1109/ICCT.2013.6820474.,* pp. 748-753.

[10] A. Asllani and Alireza Lari (2007), "Using genetic algorithm for dynamic and multiple criteria web-site optimizations," *European Journal of Operational Research,* vol. 176, no. 3, pp. 1767-1777.

[11] X. Yuan (2017), "An improved Apriori algorithm for mining association rules," in *AIP Conference Proceedings 1820, https://doi.org/10.1063/1.4977361*, Shanghai.

[12] V. 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,* vol. 25, no. 8, pp. 1772-1786.

[13] T. A. e. Aguiar, Salman Khan and Shankar B. Naik (2024), "Efficient High Utility Itemset Mining Using Genetic Algorithms and Bit-Vector Optimization," *Data Science and Security Lecture Notes in Networks and Systems. https://doi.org/10.1007/978-981-97-0975-5_34,* pp. 369-376.

[14] Y. Liu, W.-k. Liao and A. Choudhary (2005), "A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets," *Advances in Knowledge Discovery and Data Mining, https://doi.org/10.1007/11430919_79,* p. 689–695.

[15] jerry chun-wei lin, lu yang, Philippe Fournier-Viger, Jimmy Ming-Thai Wu, Tzung-Pei Hong, Leon Shyue-Liang Wang and Justin Zhan (2016), "mining high-utility itemsets based on particle swarm optimization," *Engineering applications of artificial intelligence,* vol. 55, pp. 320-330.

[16] Kannimuthu, S. and Premalatha, K, (2014)"Discovery of High Utility Itemsets Using Genetic Algorithm with Ranked Mutation," *An International Journal, Applied Artificial Intelligence, https://doi.org/10.1080/08839514.2014.891839,* vol. 28, no. 4, p. 337–359.

[17] S. B. Naik, Jyoti D. Pawar and Authors Info & Claims (2015), "A quick algorithm for incremental mining closed frequent itemsets over data streams," *Proceedings of the 2nd ACM IKDD Conference on Data Sciences, https://doi.org/10.1145/2732587.2732611,* pp. 126 - 127.

MINING FREQUENT PATTERNS FROM DATA STREAMS USING A GENETIC ALGORITHM WITH BIT REPRESENTATION AND PARALLEL PROCESSING

Pham Duc Thanh, Le Thi Minh Nguyen

In the age of big data, the ability to extract and get useful insights from streaming data is very important for applications like real-time analytics, anomaly detection, and decision-making processes. This paper proposes a novel approach to mine frequent patterns in a stream by using genetic algorithm, parallel processing, and bitwise operations. The core of this method is to mainly employ Python's ThreadPoolExecutor as a means of parallel processing to speed up the calculations and efficiently manage large streams of data. The proposed algorithm employs a method using a sliding window technique to dynamically retain and update frequent patterns when new data arrives. This approach method maintains relevant analysis to recent data, and resolves the challenges posed by the transiency of data streams. By the usage of bitwise operations **in** the genetic algorithm, this method optimizes **the** representation and manipulation of frequent patterns, thus reducing computational costs and improving performance. ThreadPoolExecutor is utilized to obtain parallel processing, this technique allows concurrently process multiple segments of the data stream. This improvement not only speeds up the algorithm but also assures scalability and compatibility with high throughput data environments. Experimental results demonstrate that in terms of both speed and accuracy, the proposed method performs an outstanding result compared to traditional frequent pattern mining techniques, especially in scenarios involving large and continuous data streams. The paper also provides a detailed discussion about the implementation of design genetic algorithm, bitwise operations among others and a parallel processing framework. The paper also provides extensive performance analysis, showcasing how efficient the solution is in real-world data stream scenarios. The suggested methodology offers a new solution for real-time data stream mining, providing powerful solutions with high applicability from continuous data streams.

Keyword— Frequent itemset, Stream data, Slide window, Mining association rules, Genetic algorithm (GA), Data mining, Batch.

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.

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 Tp. Hồ Chí Minh. Lĩnh vực nghiên cứu đang quan tâm là khai thác dữ liệu.