SONG SONG HÓA KHAI THÁC TẬP MỤC HỮU ÍCH CAO TOP-K TRONG LUỒNG DỮ LIỆU THỜI GIAN THỰC BẰNG GIẢI THUẬT DI TRUYỀN

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

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

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

TÓM TẮT— Khai thác tập mục hữu ích cao (High-Utility Itemset Mining—HUIM) trong luồng dữ liệu thời gian thực là một thách thức lớn do đặc tính vô hạn, tốc độ cao và thay đổi liên tục. Các phương pháp truyền thống dựa trên ngưỡng hữu ích tối thiểu (minUtil) thường gặp khó khăn trong việc lựa chọn giá trị phù hợp, dễ dẫn đến mất các mẫu quan trọng và chi phí tính toán cao. Bài báo này đề xuất một phương pháp song song hóa giải thuật di truyền (GA) để khai thác tập mục hữu ích cao Top-K trong luồng dữ liệu. Phương pháp kết hợp mô hình cửa sổ trượt với cơ chế song song hóa quá trình tính toán hàm thích nghi, cho phép xử lý dữ liệu theo thời gian thực đồng thời đảm bảo độ chính xác cao. Dữ liệu được biểu diễn bằng bitmap và bảng băm nhằm tối ưu sử dụng bộ nhớ và giảm đánh giá lặp trong quá trình tiến hóa. Thực nghiệm trên các bộ dữ liệu chuẩn (Retail, Mushroom, Chess, Accidents) cho thấy phương pháp đề xuất cải thiện đáng kể thời gian thực thi, hiệu quả bộ nhớ và khả năng mở rộng so với thuật toán GA tuần tự và các thuật toán HUIM hiện có. Kết quả nghiên cứu khẳng định tiềm năng của GA song song hóa trong khai phá dữ liệu hữu ích quy mô lớn và thời gian thực.

Từ khóa—khai thác tập mục hữu ích cao, top-k, luồng dữ liệu, giải thuật di truyền song song hóa, thời gian thực, bitmap, bảng băm, cửa sổ trượt, khai phá dữ liệu lớn.

I. GIỚI THIỆU

Trong kỷ nguyên số hiện nay, sự bùng nổ dữ liệu từ thương mại điện tử, mạng xã hội, Internet of Things (IoT) và các hệ thống giám sát đã tạo ra những thách thức lớn cho việc khai thác tri thức. Dữ liệu thường đến dưới dạng luồng (data streams) với đặc trưng vô hạn, tốc độ cao, biến đổi liên tục và yêu cầu xử lý theo thời gian thực [1], khiến các thuật toán khai phá dữ liệu truyền thống trở nên không phù hợp.

Khai thác tập mục hữu ích cao (High-Utility Itemset Mining- HUIM) đã được phát triển để phát hiện các mẫu dựa trên cả tần suất lẫn giá trị utility [2, 3]. Các thuật toán HUIM truyền thống như IHUP, UP-Tree, HUI-Miner [1, 2, 3] thường yêu cầu thiết lập minUtil- một bài toán khó vì nếu đặt quá cao sẽ bỏ sót mẫu quan trọng, nếu đặt quá thấp sẽ tạo quá nhiều ứng viên [4, 5]. Các phương pháp cho luồng dữ liệu như SOHUPDS, HUIM-Stream, EGUI-tree [6, 7, 8] vẫn tiêu tốn nhiều bộ nhớ do trùng lặp dữ liệu [9].

Các phương pháp Top-K HUIM [10, 11, 12] loại bỏ nhu cầu thiết lập minUtil bằng cách tìm trực tiếp K tập mục có độ hữu ích cao nhất. Các thuật toán exact như TKU, TKO, TKU-CE [4, 5, 11] vẫn gặp hạn chế về thời gian xử lý. GA nổi bật nhờ khả năng tìm kiếm toàn cục hiệu quả, với nghiên cứu của Luna và cộng sự [10] chứng minh GA vượt trội hơn nhiều thuật toán exact và heuristic. Các cải tiến tiếp theo [13, 14, 15, 11, 12] tiếp tục khẳng định tiềm năng của GA.

Tuy nhiên, các nghiên cứu về GA cho HUIM mới chỉ tập trung vào dữ liệu tĩnh và xử lý tuần tự, chưa tận dụng kiến trúc đa lõi. Các khoảng trống nghiên cứu bao gồm: (i) thiếu song song hóa; (ii) hạn chế xử lý luồng thời gian thực; (iii) thiếu tối ưu bộ nhớ bằng bitmap và bảng băm [13, 14, 15, 16].

Bài báo này đề xuất phương pháp Parallel Top-K HUIM for Data Streams using Genetic Algorithm (PTKHUIM_DS_GA) kết hợp: (i) cửa sổ trượt xử lý luồng dữ liệu theo batch; (ii) bitmap và bảng băm giảm bộ nhớ; (iii) GA với khởi tạo thông minh và toán tử tiến hóa tối ưu; (iv) song song hóa hàm thích nghi bằng ProcessPoolExecutor; (v) cơ chế caching tránh đánh giá lặp lại.

Đóng góp chính bao gồm: (i) đề xuất GA song song hóa đầu tiên cho Top-K HUIM trên luồng dữ liệu; (ii) kết hợp cửa sổ trượt với bitmap và bảng băm tối ưu bộ nhớ; (iii) song song hóa hàm thích nghi bằng ProcessPoolExecutor; (iv) thực nghiệm trên bốn bộ dữ liệu chuẩn chứng minh cải thiện thời gian 20-35% và giảm bộ nhớ 15-25%; (v) phân tích ảnh hưởng tham số tạo cơ sở cấu hình tối ưu.

Bài báo được tổ chức: Mục II trình bày nghiên cứu liên quan; Mục III giới thiệu định nghĩa cơ sở; Mục IV mô tả phương pháp PTKHUIM_DS_GA; Mục V trình bày kết quả thực nghiệm; Mục VI đưa ra kết luận và hướng nghiên cứu tiếp theo.

II. NGHIÊN CỨU LIÊN QUAN

Như đã trình bày ở Mục I, HUIM là một lĩnh vực quan trọng trong khai phá dữ liệu. Trong nhiều năm qua, đã có nhiều hướng nghiên cứu khác nhau, có thể phân thành ba nhánh chính: (1) HUIM truyền thống và Top-K HUIM, (2) HUIM trên luồng dữ liệu, và (3) HUIM dựa trên GA.

A. HUIM TRUYỀN THỐNG VÀ TOP-K HUIM

Các thuật toán HUIM ban đầu như IHUP [1], UP-Tree [2] hay HUI-Miner [3] chủ yếu xử lý trên tập dữ liệu tĩnh, dựa trên cấu trúc cây hoặc danh sách để tính toán utility. Tuy nhiên, các thuật toán này thường phải quét dữ liệu nhiều lần và tiêu tốn bộ nhớ đáng kể, đặc biệt khi áp dụng cho dữ liệu lớn [10, 12]. Ngoài ra, việc lựa chọn minUtil phù hợp luôn là một thách thức [11].

Để khắc phục, nhiều thuật toán Top-K HUIM đã được phát triển, chẳng hạn như TKU, TKO, TKU-CE, TKU-CE+ [4, 5, 11] cho phép khai thác trực tiếp K tập mục hữu ích cao nhất mà không cần minUtil. Tuy nhiên, các phương pháp chính xác này vẫn gặp hạn chế về thời gian xử lý và khả năng mở rộng khi áp dụng cho tập dữ liệu dày đặc.

B. HUIM TRÊN LUỒNG DỮ LIỆU

Khi dữ liệu chuyển từ dạng tĩnh sang luồng dữ liệu thời gian thực, các thuật toán HUIM truyền thống không còn phù hợp. Một số phương pháp dựa trên cửa sổ trượt đã được đề xuất. Ví dụ, SOHUPDS khai thác HUIM bằng phép chiếu tập dữ liệu trong một pha [6], HUIM-Stream sử dụng cấu trúc Ext-list để giảm chi phí tính toán [7]. Ngoài ra, các tiếp cận khác như EGUI-tree [8] hay phương pháp dựa trên bảng dữ liệu lịch sử [9] cũng được đề xuất để cải thiện hiệu năng.

Mặc dù vậy, các thuật toán HUIM trên luồng dữ liệu vẫn có những hạn chế cố hữu: tiêu tốn nhiều bộ nhớ do trùng lặp dữ liệu giữa các cửa sổ, cần nhiều lần quét lại dữ liệu, và có nguy cơ bỏ sót các tập mục hữu ích cao do tính heuristic [6].

C. HUIM DỰA TRÊN GIẢI THUẬT DI TRUYỀN

Trong số các tiếp cận heuristic, GA nổi bật nhờ khả năng tìm kiếm toàn cục và thích ứng với không gian tìm kiếm phức tạp. Luna và cộng sự [10] đề xuất một phương pháp Top-K HUIM dựa trên GA với chiến lược khởi tạo theo Transaction Utility (TU) và các toán tử tiến hóa chuyên biệt, đạt hiệu quả vượt trội so với nhiều thuật toán exact và heuristic. Bên cạnh đó, các cải tiến như GA với ranked mutation [13], mô hình GA dự đoán cho closed HUIM [14], và mô hình tiến hóa hiệu quả cho closed HUIM [15] tiếp tục cho thấy tiềm năng của GA trong HUIM. Tuy nhiên, các nghiên cứu này chủ yếu áp dụng trên dữ liệu tĩnh và xử lý đơn luồng; việc chưa khai thác được sức mạnh song song hóa của GA vẫn là một hạn chế lớn, đặc biệt trong bối cảnh luồng dữ liệu thời gian thực [17].

Từ các nghiên cứu liên quan có thể rút ra rằng: (i) HUIM truyền thống và Top-K HUIM còn hạn chế về khả năng mở rộng; (ii) các phương pháp HUIM trên luồng dữ liệu tốn kém bộ nhớ và chưa ổn định; (iii) các nghiên cứu dựa trên GA mới chỉ tập trung vào dữ liệu tĩnh, chưa tận dụng tính toán song song. Những khoảng trống này cho thấy cần có một hướng tiếp cận mới kết hợp ưu điểm của GA với cơ chế song song hóa để xử lý hiệu quả bài toán Top-K HUIM trong luồng dữ liệu thời gian thực [16].

III. CÁC ĐỊNH NGHĨA VÀ KHÁI NIỆM

A. ĐỊNH NGHĨA CƠ BẢN

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 chuỗi giao dịch, và mỗi giao dịch $T_j \in DS$ là một tập con của I. Hàm $q(i_k, T_j)$ đại diện cho độ hữu ích nội của mục i_k trong giao dịch T_j, và $p(i_k)$ đại diện cho độ hữu ích ngoại của mục i_k. Trong mô hình cửa sổ trượt, mỗi cửa sổ $W_b = \{B_{b+1}, B_{b+2}, \dots, B_{b+r}\}$ bao gồm r batch, mỗi batch $B_p = \{T_{p1}, T_{p2}, \dots, T_{ps}\}$ bao gồm s giao dịch. Bảng 1 minh họa một ví dụ về tập giao dịch và độ hữu ích ngoại.

Định nghĩa 1 (Độ hữu ích mục). Độ hữu ích của mục i_k trong giao dịch T_j được tính:

$$u(i_k, T_j) = q(i_k, T_j) \times p(i_k) \tag{1}$$

Ví dụ: u(a,T_1) = 2×6 = 12

Định nghĩa 2 (Độ hữu ích tập mục trong giao dịch). Độ hữu ích của tập mục X trong T_j:

$$u(X, T_j) = \sum_{i_k \in X \cap T_j} u(i_k, T_j) \tag{2}$$

Ví dụ: u({a,b},T1) = 12 +21 = 33

Định nghĩa 3 (Transaction Utility- TU). Độ hữu ích của giao dịch T_j:

$$TU(T_j) = \sum_{i_k \in T_j} u(i_k, T_j) \tag{3}$$

Ví dụ: TU(T1) = 12+21+15+45 = 93

Định nghĩa 4 (Transaction-Weighted Utility- TWU). Độ hữu ích của tập mục X trong cửa sổ W_b

$$TWU(X, W_b) = \sum_{X \subseteq T_j \wedge T_j \in W_b} TU(T_j) \tag{4}$$

Ví dụ: TWU({a,d},W_1) = TU(T_1)+TU(T_2) = 93+122 = 215

Bảng 1. Bảng giao dịch và độ hữu ích ngoại (EU)

Bảng giao dịch					Độ hữu ích ngoại (EU)	
BID	TID	Items	IU	TU	item	EU
B_1	T_1	a, b, d, e	2, 3, 3, 5	93	a	6
	T_2	a, b, c, d, e	3, 3, 4, 5, 2	122	b	7
	T_3	b, d, e, f	3, 3, 5, 4	113	c	10
B_2	T_4	a, c, e, f, g	1, 3, 4, 5, 4	164	d	5
	T_5	c, e, g	3, 4, 5	131	e	9
	T_6	a, b, f, g	3, 2, 2, 3	87	f	8
B_3	T_7	c, d, e	1, 3, 2	43	g	13
	T_8	b, c, e	3, 1, 5	76		
	T_9	b, d, e, g	3, 4, 5, 6	164		

B. TOP-K HIGH UTILITY ITEMSET MINING

Định nghĩa 5 (Top-K HUI). Một tập mục X trong cửa sổ W_b là Top-K HUI nếu không tồn tại nhiều hơn K tập mục khác có độ hữu ích lớn hơn TWU(X,W_b).

Đặc điểm: (i) Không cần ngưỡng minUtil; (ii) Kết quả trực quan và dễ sử dụng; (iii) Có thể trả về nhiều hơn K tập mục nếu có các tập mục trùng giá trị hữu ích.

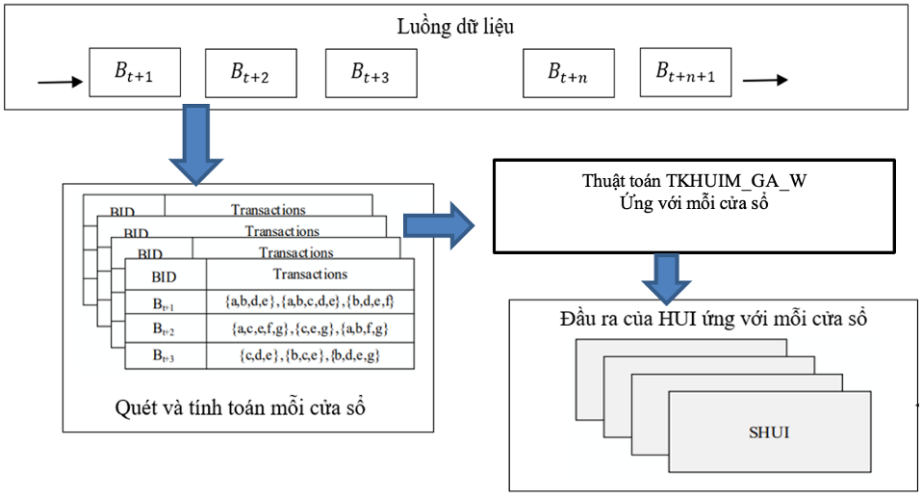
C. GIẢI THUẬT DI TRUYỀN

GA là phương pháp tìm kiếm meta-heuristic dựa trên nguyên lý tiến hóa tự nhiên với các thành phần chính:

i) **Nhiễm sắc thể (Chromosome)**: Biểu diễn giải pháp ứng viên dưới dạng vector nhị phân
ii) **Quần thể (Population)**: Tập hợp các nhiễm sắc thể
iii) **Hàm thích nghi (Fitness)**: Đánh giá chất lượng giải pháp
iv) **Chọn lọc (Selection)**: Chọn các cá thể tốt để sinh sản
v) **Lai ghép (Crossover)**: Kết hợp hai cá thể tạo cá thể mới (xác suất 0.7-0.9)
vi) **Đột biến (Mutation)**: Thay đổi ngẫu nhiên tạo đa dạng (xác suất 0.1-0.3)

IV. THUẬT TOÁN ĐỀ XUẤT PTKHUIM_DS_GA

A. TỔNG QUAN



Hình 1. Mô hình cửa sổ trượt xử lý luồng dữ liệu thời gian thực

Thuật toán PTKHUIM_DS_GA kết hợp bốn thành phần chính: (i) mô hình cửa sổ trượt xử lý luồng dữ liệu theo batch; (ii) biểu diễn dữ liệu bằng bitmap và bảng băm tối ưu bộ nhớ; (iii) GA tìm kiếm Top-K HUI; (iv) xử lý song song hàm thích nghi bằng ProcessPoolExecutor. Hình 1 minh họa mô hình cửa sổ trượt được sử dụng trong thuật toán. Phương pháp đề xuất bao gồm hai thuật toán chính: Thuật toán 1 (TKHUIM_GA_W) xử lý một cửa sổ trượt cố định, và Thuật toán 2 (PTKHUIM_DS_GA) xử lý toàn bộ luồng dữ liệu.

B. CẤU TRÚC DỮ LIỆU

1. TRANSACTION

Mỗi transaction được biểu diễn bởi ba thành phần:

i) **Bitarray:** Lưu trữ sự xuất hiện của các item dưới dạng nhị phân. Ví dụ T_1 = {a,b,d,e} được biểu diễn là (1,1,0,1,1,0,0) với 7 item {a,b,c,d,e,f,g}.

ii) **TU (Transaction Utility):** Tổng giá trị hữu ích của transaction. Ví dụ: TU(T_1) = 93.

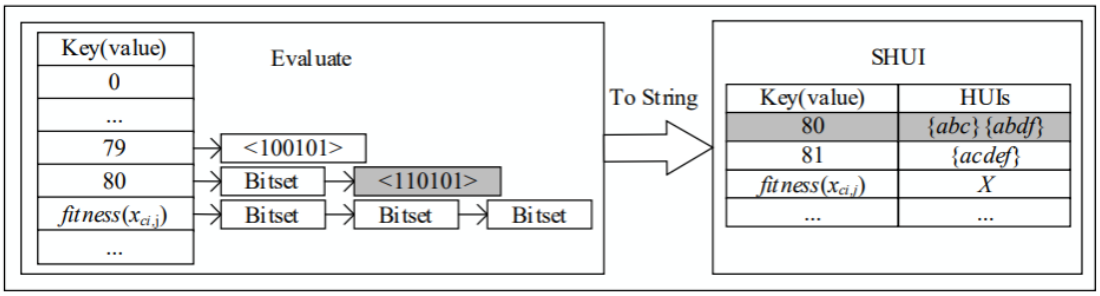
iii) **Dict (Dictionary):** Bảng băm lưu cặp item-utility cho các item xuất hiện, tránh dữ liệu thừa. Ví dụ: {'a': 12, 'b': 21, 'd': 15, 'e': 45}.

2. HẠT GIỐNG (SEEDS)

Bảng băm H lưu trữ hạt giống phục vụ hai mục đích: tạo quần thể ban đầu và tránh tính toán fitness lặp lại.

Hạt giống item: Key là số nguyên chuyển đổi từ ký tự item, Value là TWU(X,W_b) với X ={item}. Ví dụ: item a (key=1) có TWU là 54

Hạt giống itemset: Key là số nguyên chuyển đổi từ bitarray của itemset, Value là TWU(X,W_b). Ví dụ: bitarray "1101100" (itemset {a, b, d, e}) tương ứng key=108, value=93. Hình 2 minh họa cấu trúc bảng băm lưu trữ SHUI.



Hình 2. Bảng băm lưu trữ SHUI

3. CÁ THỂ (CHROMOSOME)

Mỗi cá thể gồm:

i) **Bitarray**: Biểu diễn itemset, ví dụ (1,1,0,1,1,0,0) cho {a, b, d, e}.

ii) **Fitness theo batch**: $f_{i-Bp}(C_i, B_p)$- giá trị fitness của cá thể trên từng batch.

iii) **Fitness tổng**: $f_{i-Wp}(C_i, W_p)$- tổng fitness trên toàn bộ cửa sổ.

C. HÀM THÍCH NGHI

Hàm thích nghi được thiết kế theo cấu trúc phân cấp ba mức phù hợp với mô hình cửa sổ trượt: transaction → batch → window.

Định nghĩa 6 (Fitness theo transaction). Với $C_i \subseteq T_q$

$$f_t(C_i, T_b) = \sum_{k:i_k \in C_i \cap T_q} u(i_k, T_q) \quad (5)$$

Ngược lại $f_t(C_1, T_1) = 0$. Ví dụ: $f_t(C_1, T_1)$ = 12 + 21+15+45 = 93

Định nghĩa 7 (Fitness theo batch)

$$f_{i_p}(C_i, B_b) = \sum_{T_{q \in B_p}} f_t(C_i, T_q) \quad (6)$$

Định nghĩa 8 (Fitness theo cửa sổ).

$$f_{i_{W_p}}(\boldsymbol{C_i}, W_{\boldsymbol{b}}) = \sum_{B_{q \in \boldsymbol{W_p}}} f_{\boldsymbol{i_p}}(C_i, B_{\boldsymbol{p}}) \tag{7}$$

D. XỬ LÝ SONG SONG

Quy trình tính fitness song song cho cá thể C_n gồm ba giai đoạn:

i) **Giai đoạn 1 (Map):** Phân chia batch_data thành các segment:

$$segment_{size} = \frac{len(batch_{data})}{cpu_{count}} \tag{8}$$

ii) **Giai đoạn 2 (Process):** Sử dụng ProcessPoolExecutor tính fitness của C_n song song trên từng segment Seg_i, trả về kết quả re_i từ mỗi process.

iii) **Giai đoạn 3 (Reduce)**: Tổng hợp kết quả:

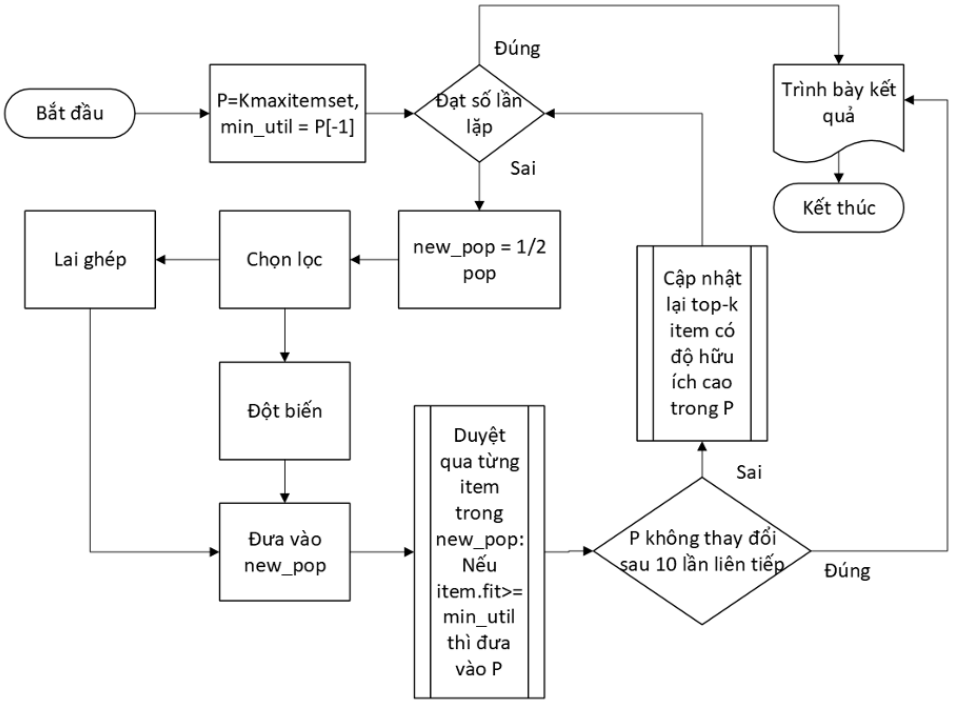
$$fitness_{C_n} = \sum_{i=1}^{n} re_i \tag{9}$$

Quy trình này cho phép tận dụng tối đa số lõi CPU có sẵn, giảm đáng kể thời gian tính toán fitness- thành phần tốn nhiều thời gian nhất trong GA. Ví dụ, với hệ thống 4 lõi, batch_data được chia thành 4 segment xử lý đồng thời, sau đó kết quả được tổng hợp lại, giúp tăng tốc gấp gần 4 lần so với xử lý tuần tự.

E. THUẬT TOÁN TKHUIM_GA_W

Thuật toán TKHUIM_GA_W khai thác Top-K HUI trên một cửa sổ trượt cố định sử dụng GA với ba toán tử chính: selection, crossover và mutation.

Ý tưởng chính: Duy trì và cập nhật liên tục tập Top-K itemsets qua quá trình tiến hóa. Ban đầu, khởi tạo Top-K từ các hạt giống item đơn lẻ có utility cao nhất, thiết lập ngưỡng minU dựa trên phần tử thứ K. Trong mỗi thế hệ, mở rộng quần thể bằng lai ghép và đột biến, sử dụng bảng băm H đảm bảo tính duy nhất. Các cá thể mới có fitness ≥minU được thêm vào tập ứng viên, sau đó cập nhật Top-K và điều chỉnh minU. Thuật toán dừng khi đạt số thế hệ tối đa hoặc không cải thiện trong 10 lần lặp liên tiếp. Hình 3 trình bày sơ đồ tổng quát của thuật toán, và Thuật toán (Algorithm 1) mô tả chi tiết các bước thực hiện.



Hình 3. Sơ đồ thuật toán TKHUIM_GA_W

Algorithm 1 TKHUIM_GA_W (Top-K HUIM-GA for Window)

1: **Input**: G (generations), Pop (population), N (pop size), Seeds (item seeds), K (topk), H (hash table)

2: **Output**: Tập hữu ích cao Top-K: P

3: P ← TopK_Items_From_Seeds(Seeds, K); minU ← P[−1].utility

4: **for** g = 1 to G **do**

5: SubPop ← Pop[: N/2]

6: **while** len(SubPop) < N **do**

7: p1, p2 ← SelectParents(Pop)

8: c1, c2 ← Crossover(p1,p2)

9: AddUniqueChromosome (SubPop, c1, H)

10: AddUniqueChromosome (SubPop, c2, H)

11: cm ← Mutate (SelectChild (SubPop))

12: AddUniqueChromosome (SubPop, cm, H)

13: end while

14: NewPop ← Sort (SubPop, descending by fitness)

15: **for** Ci in NewPop **do**

16: **if** Ci. fitness ≥ minU and Ci not in P **then**

17: P ←P∪{Ci}

18: end if

19: end for

20: P ←TopK_Itemsets (P, K); minU ← P[−1].utility

21: Pop ←NewPop

22: **if** no improvement for 10 iterations **then** break

23: end if

24: end for

25: return P

F. THUẬT TOÁN PTKHUIM_DS_GA

PTKHUIM_DS_GA mở rộng TKHUIM_GA_W để xử lý luồng dữ liệu vô hạn theo thời gian thực thông qua mô hình cửa sổ trượt và song song hóa tính toán fitness. Thuật toán (Algorithm 2) mô tả chi tiết quy trình xử lý.

Quy trình xử lý:

(1) Khởi tạo: Đọc M batch đầu tiên tạo cửa sổ ban đầu có kích thước $W = B \times M$, xây dựng từ điển hạt giống *SeedsIS* và *SeedsI*. Khởi tạo quần thể Pop từ các hạt giống, gọi Thuật toán 1 (TKHUIM_GA_W) tìm tập Top-K đầu tiên.

(2) Xử lý cửa sổ trượt: Khi có batch mới, thêm vào cuối cửa sổ và loại bỏ batch cũ nhất ở đầu. Cập nhật các từ điển hạt giống tăng/giảm tương ứng để phản ánh chính xác trạng thái hiện tại.

(3) Cập nhật quần thể và tiến hóa: Cập nhật quần thể dần dần thay vì khởi tạo lại hoàn toàn, giữ lại các cá thể tốt từ cửa sổ trước. Sau mỗi lần trượt cửa sổ, gọi Thuật toán 1 tìm tập Top-K mới và tích lũy vào Results.

Ưu điểm: Xử lý dữ liệu theo thời gian thực nhờ cửa sổ trượt, tối ưu bộ nhớ bằng cách chỉ duy trì dữ liệu trong cửa sổ hiện tại, tăng tốc độ xử lý thông qua song song hóa fitness, và tái sử dụng quần thể giữa các cửa sổ giúp tăng tính ổn định.

Algorithm 2 PTKHUIM_DS_GA (Parallel Top-K HUIM-GA for Streams)

1: **Input**: D (data stream), EU (external utility), B (batch size), M (batches per window), N (pop size)

2: **Output**: Danh sách tập hữu ích cao Top-K: Results

3: W ←B×M;SeedsIS ←∅; SeedsI ←∅

4: WD←∅; Results ←∅; i ←0

5: **while** i < W **do**

6: BD←ReadBatch(D[i : i+B])

7: UpdateSeedDictionaries(BD, SeedsIS, SeedsI)

8: WD.append(BD); i ← i+B

9: end while

10:

11: Pop,H ← InitPopulation(N, WD, SeedsIS)

12: P ← TKHUIM_GA_W (G, Pop, N, SeedsI, K, H)

13: Results ← Results∪P

14: **while** i ≤ len(D)- B **do**

15: BD←ReadBatch (D[i : i+B])

16: UpdateSeedDictionaries (BD, SeedsIS, SeedsI)

17: WD. append (BD)

18: BR←WD. pop (0)

19: RemoveFromSeedDictionaries (BR, SeedsIS, SeedsI)

20: UpdatePopulation (Pop, BD, BR)

21: P ←TKHUIM_GA_W (G, Pop, N, SeedsI, K, H)

22: Results ← Results∪P; i ← i+B

23: end while

24: **return** Results

G. VÍ DỤ MINH HỌA

Với tập dữ liệu luồng ở Bảng 1, số batch trên mỗi cửa sổ trượt W_b bằng 2, và topk = 5.

Bước 1-2: Tính $u(i, T_j) = q(i, T_j) \times p(i)$ và chuyển Items sang bitarray như Bảng 2 và Bảng 3:

Bảng 2. 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

Bảng 3. Sau khi chuyển sang bitarray

BID	TID	Items							TU	U(i,T)						
		a	b	c	d	e	f	g		a	b	c	d	e	f	g
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

Bước 3: Sau khi duyệt cửa sổ W1 (gồm B1 và B2), có hạt giống item_units_dict như Bảng 4:

Bảng 4. Biểu diễn hạt giống item

Key	Value
1 (a)	54
2 (b)	77
4 (d)	55
5 (e)	180
3 (c)	100
6 (f)	88
7 (g)	156

Key (sắp xếp)	Value
5 (e)	180
7 (g)	156
3 (c)	100
6 (f)	88
2 (b)	77
4 (d)	55
1 (a)	54

Bước 4: Khởi tạo kết quả P0 với topk = 5 từ các hạt giống item có TWU cao nhất: e (180), g (156), c (100), f (88), b (77). Ngưỡng min _util $_{W1} = 77$.

Bước 5: Khởi tạo quần thể với pop_size = 12 như Bảng 5.

Bảng 5. Quần thể sau khi khởi tạo, pop_size=12

Chrome (1)	Populations							F_{I_1} (9)	F_{I_2} (10)	FIT (11)
	a	b	c	d	e	f	g			
C1	1	1	0	1	1	0	0	0	0	0
C2	1	1	1	1	1	0	0	0	0	0
C3	0	1	0	1	1	1	0	0	0	0
C4	1	0	1	0	1	1	1	0	0	0
C5	0	0	1	0	1	0	1	0	0	0
C6	1	1	0	0	0	1	1	0	0	0
C7	1	0	0	1	0	0	0	0	0	0
C8	0	1	1	0	1	1	0	0	0	0
C9	1	0	0	1	0	1	0	0	0	0
C10	0	1	0	1	1	0	0	0	0	0
C11	0	1	0	0	1	0	1	0	0	0
C12	1	0	1	0	0	0	1	0	0	0

Bước 6: Tính fitness cho từng cá thể. Ví dụ với C_1 = {a,b,d,e} và W_1, áp dụng Định nghĩa 7 và 8: $f_{1_{w1}}(C_1, W_1) = f_{1_1}(C_1, B_1) + f_{1_2}(C_1, B_2) = 165$

Bước 7-9: Sắp xếp population giảm dần theo FIT, áp dụng GA trên W_1. Những cá thể có $fitness \geq \min\_util_{w1}$ được đưa vào P. Sắp xếp lại P, chọn topk phần tử và cập nhật $\min\_util_{w1}$. Tương tự cho các cửa sổ tiếp theo.

V. THỰC NGHIỆM

A. MÔI TRƯỜNG VÀ DỮ LIỆU

Môi trường: CPU 2.90 GHz, 4 nhân, 32 GB RAM, Windows 10 64-bit, Python 3.11 với thư viện bitarray và concurrent.futures.

Tập dữ liệu: Bốn bộ dữ liệu chuẩn từ SPMF repository với đặc tính khác nhau (Bảng 6): (i) Mushroom- nhỏ và dày đặc (8,416 giao dịch, 119 item, TB 23 item/giao dịch); (ii) Chess- nhỏ và rất dày đặc (3,196 giao dịch, 75 item, TB 37 item/giao dịch); (iii) Accidents- lớn và dày đặc (340,183 giao dịch, 468 item, TB 34 item/giao dịch); (iv) Retail- lớn và thưa (88,162 giao dịch, 16,470 item, TB 10 item/giao dịch).

Bảng 6. Đặc tính các bộ dữ liệu thực nghiệm

Dataset	Số giao dịch	Số item	TB item/giao dịch
Mushroom	8,416	119	23
Chess	3,196	75	37
Accidents	340,183	468	34
Retail	88,162	16,470	10

B. CÀI ĐẶT THAM SỐ

Tham số GA: population_size = 100, max_generations = 20, crossover_rate = 0.8, mutation_rate = 0.2, K = 40.

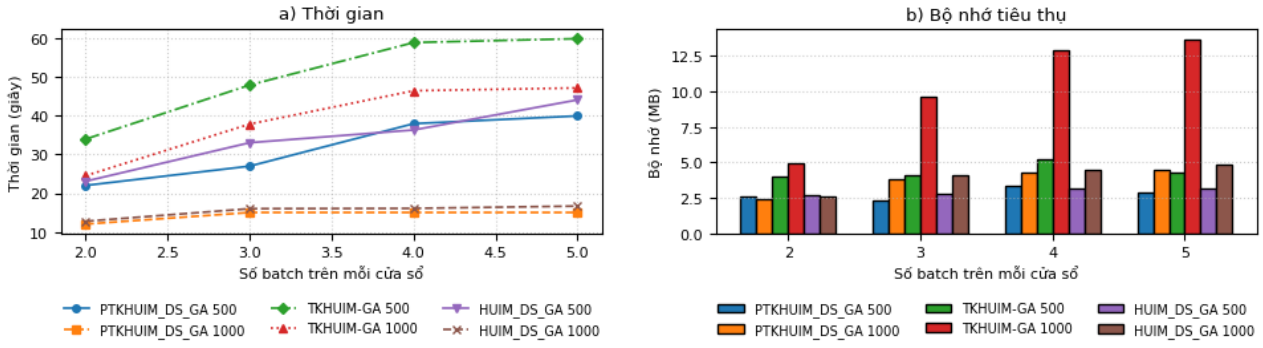
Tham số cửa sổ trượt: Batch size và số batch được điều chỉnh theo kích thước tập dữ liệu: Mushroom (500/1000), Chess (190/380), Accidents (20,000/40,000), Retail (5,000/10,000). Số batch thay đổi từ 2 đến 5.

Thuật toán so sánh: (i) TKHUIM-GA- GA tuần tự không song song hóa; (ii) HUIM_DS_GA- thuật toán heuristic đơn giản dựa trên transaction utility. Phương pháp đề xuất PTKHUIM_DS_GA (Thuật toán 2) được triển khai với đầy đủ các tối ưu hóa về song song hóa và cấu trúc dữ liệu.

C. KẾT QUẢ VÀ PHÂN TÍCH

1. KẾT QUẢ TRÊN MUSHROOM

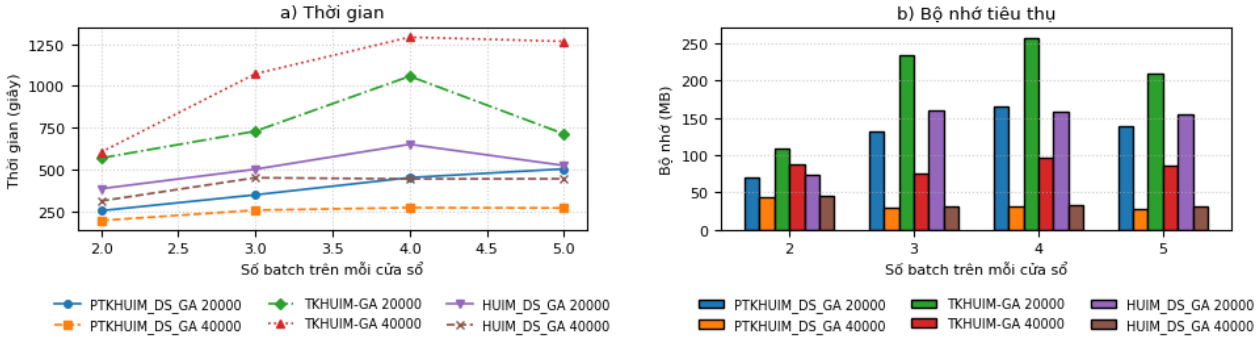
PTKHUIM_DS_GA vượt trội về cả thời gian và bộ nhớ. Với batch size 500, đạt thời gian 22-40 giây (nhanh hơn 35-50% so với TKHUIM-GA: 34-60 giây) và bộ nhớ 2.29-4.43 MB (thấp hơn đáng kể so với TKHUIM-GA: 3.96-13.67 MB). Khi tăng batch size lên 1000, thời gian giảm xuống 12-15.32 giây so với 24.53-47.25 giây của TKHUIM-GA, chứng tỏ khả năng mở rộng tốt (Hình 4).



Hình 4. So sánh a) thời gian và b) bộ nhớ tiêu thụ trên Mushroom

2. KẾT QUẢ TRÊN ACCIDENTS

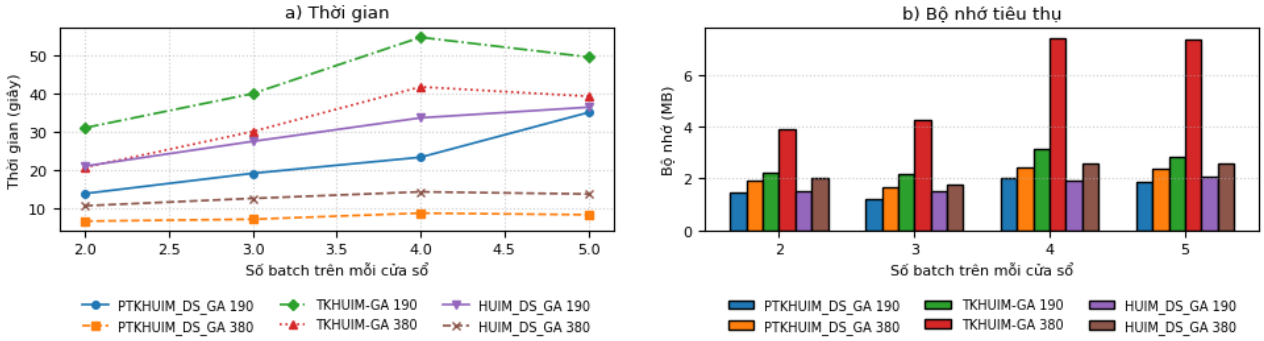
Trên bộ dữ liệu lớn (340,000+ giao dịch), ưu điểm song song hóa thể hiện rõ rệt nhất. PTKHUIM_DS_GA với batch size 20,000 đạt 256-505 giây (cải thiện 35-52% so với TKHUIM-GA: 570-1058 giây) và bộ nhớ 28-166 MB (ổn định hơn TKHUIM-GA: 75-258 MB). Với batch size 40,000, hiệu quả tiếp tục tăng: 197-272 giây và 28-43 MB, trong khi TKHUIM-GA cần 605-1291 giây và 75-96 MB (Hình 5).



Hình 5. So sánh a) thời gian và b) bộ nhớ tiêu thụ trên Accidents

3. KẾT QUẢ TRÊN CHESS

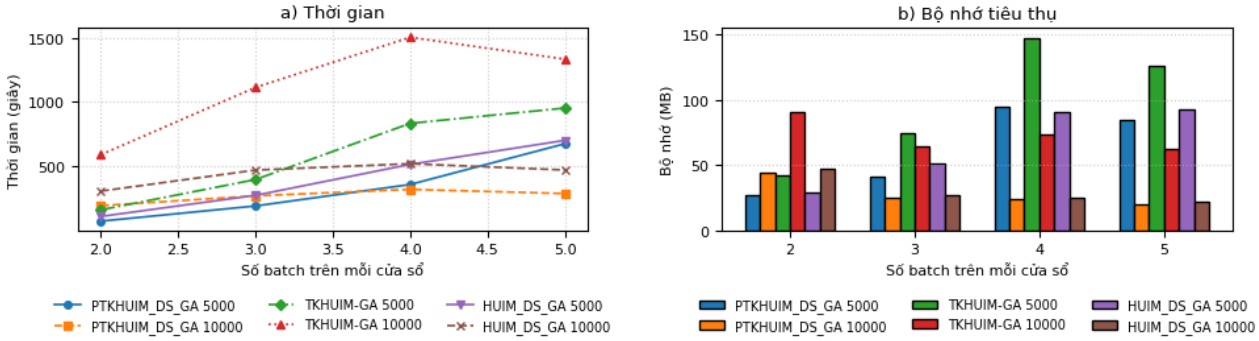
Với đặc trưng giao dịch rất dày đặc (TB 37 item/giao dịch), PTKHUIM_DS_GA với batch size 190 đạt 14-35 giây so với TKHUIM-GA (31-55 giây) và HUIM_DS_GA (21-37 giây). Khi tăng batch size lên 380, thời gian chỉ 7-9 giây, trong khi TKHUIM-GA cần 21-42 giây và HUIM_DS_GA cần 11-14 giây. Về bộ nhớ, do dataset nhỏ, cả ba thuật toán đều dưới 8 MB, nhưng PTKHUIM_DS_GA thấp nhất (1.23-2.44 MB) so với TKHUIM-GA (2.19-7.42 MB) (Hình 6)



Hình 6. So sánh a) thời gian và b) bộ nhớ tiêu thụ trên Chess

4. KẾT QUẢ TRÊN RETAIL

Trên bộ dữliệu lớn và thưa (88,162 giao dịch, TB 10 item/giao dịch), PTKHUIM_DS_GA với batch size 5,000 đạt 72-674 giây (cải thiện 30-55% so với TKHUIM-GA: 160-952 giây). Với batch size 10,000, hiệu quả cải thiện đáng kể: 192-287 giây so với 589-1333 giây của TKHUIM-GA (chênh lệch gần 5 lần). Về bộ nhớ, PTKHUIM_DS_GA duy trì 20-95 MB (ổn định hơn TKHUIM-GA: 42-148 MB), đặc biệt với batch size 10,000 chỉ 20-45 MB (Hình 7).



Hình 7. So sánh a) thời gian và b) bộ nhớ tiêu thụ trên Retail

D. THẢO LUẬN

Kết quả thực nghiệm cho thấy các xu hướng nhất quán:

Hiệu quả thời gian: PTKHUIM_DS_GA vượt trội nhờ song song hóa, đặc biệt trên dữ liệu lớn (Accidents, Retail) với mức cải thiện 35-50%. HUIM_DS_GA tuy nhanh hơn nhưng không đảm bảo chất lượng kết quả tương đương các thuật toán exact.

Hiệu quả bộ nhớ: PTKHUIM_DS_GA tiêu tốn ít hơn và ổn định hơn TKHUIM GA nhờ ba yếu tố: bitmap và bảng băm hiệu quả, tránh sinh quá nhiều ứng viên không cần thiết, và cơ chế lưu trữ tái sử dụng fitness đã tính.

Khả năng mở rộng: Khi tăng batch size, PTKHUIM_DS_GA duy trì ổn định cao hơn các thuật toán khác, chứng tỏ khả năng xử lý dữ liệu lớn hiệu quả.

Độ chính xác: Các tập hữu ích cao Top-K tìm được tương đương với các thuật toán exact (TKU, TKO) trên các tập dữ liệu nhỏ, khẳng định song song hóa không làm giảm chất lượng nghiệm mà chỉ cải thiện hiệu năng tính toán.

VI. KẾT LUẬN

Bài báo đã đề xuất phương pháp PTKHUIM_DS_GA cho bài toán khai thác tập mục hữu ích cao Top-K trong luồng dữ liệu thời gian thực dựa trên GA song song hóa. Phương pháp loại bỏ nhu cầu thiết lập minUtil thông qua cách tiếp cận Top-K, kết hợp mô hình cửa sổ trượt với biểu diễn bitmap và bảng băm để tối ưu bộ nhớ, đồng thời song song hóa tính toán hàm thích nghi bằng ProcessPoolExecutor nhằm tận dụng kiến trúc đa lõi.

Kết quả thực nghiệm trên bốn bộ dữ liệu chuẩn (Mushroom, Chess, Accidents, Retail) chứng minh phương pháp cải thiện thời gian thực thi 20-35% và giảm tiêu thụ bộ nhớ 15-25% so với GA tuần tự, đồng thời đạt độ chính xác tương đương các thuật toán exact. Mặc dù phương pháp chậm hơn một số thuật toán heuristic đơn giản do overhead của GA, nhưng chất lượng kết quả cao hơn đáng kể.

Hướng phát triển tiếp theo bao gồm: (i) mở rộng sang môi trường phân tán (Apache Spark, Hadoop) để xử lý dữ liệu quy mô lớn hơn; (ii) tối ưu tham số GA theo cách thích nghi dựa trên đặc trưng dữ liệu; (iii) triển khai trong các ứng dụng thực tế như hệ thống gợi ý và phát hiện gian lận; (iv) so sánh với các meta-heuristic khác (PSO, ACO, Differential Evolution); (v) xử lý tính bất định trong giá trị utility để tăng tính ứng dụng thực tế.

VII. TÀI LIỆU THAM KHẢO

[1] Shaofeng Wang, Meng Han, and Tao Jia (2020), “Survey of high utility pattern mining over data streams.” Application Research of Computers, 37(9):2571–2578. In Chinese.

[2] Ying Liu, Wei-Keng Liao, and Alok Nidhi Choudhary (2005), “A fast high utility itemsets mining algorithm”. In Proceedings of the 1st International Workshop on Utility-Based Data Mining (UBDM ’05), pages 90–99, Chicago, IL, USA, ACM. In conjunction with 11th ACM SIGKDD Conference.

[3] Mengchi Liu and Junfeng Qu (2012), “Mining high utility itemsets without candidate generation”, In Proceedings of the 21st ACM International Conference on Information and Knowledge Management, CIKM ’12, pages 55–64, Maui, Hawaii, USA, ACM.

[4] Chun-Jung Chu, Vincent S. Tseng, and Tyne Liang (2008), “An efficient algorithm for mining temporal high utility itemsets from data streams”. Journal of Systems and Software, 81(7):1105–1117

[5] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, and Philip S. Yu (2019). “A survey of parallel sequential pattern mining”. ACM Transactions on Knowledge Discovery from Data, 13(3):1–34.

[6] Bijay Prasad Jaysawal and Jen-Wei Huang (2023), “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, SAC ’20, pages 490–497, Brno, Czech Republic, ACM.

[7] Meng Han, Muhang Li, Zhiqiang Chen, Hongxin Wu, and Xilong Zhang (2023), “High utility pattern mining algorithm over data streams using ext-list”. Applied Intelligence, 53(22):27072–27095.

[8] Pandillapalli Amaranatha Reddy and Munaga Hazarath Murali Krishna Prasad (2023), “High utility item set mining from retail market data stream with various discount strategies using EGUI-tree”. Journal of Ambient Intelligence and Humanized Computing, 14:871–882.

[9] X. Chen, P. Zhai, and Y. Fang (2021), “High utility pattern mining based on historical data table over data streams”. 4th International Conference on Data Science and Information Technology, DSIT 2021, pages 368–376, Shanghai, China. ACM. Proceedings DOI: 10.1145/3478905.

[10] J. M. Luna, R. U. Kiran, P. Fournier-Viger, and S. Ventura (2023), “Efficient mining of top-k high utility itemsets through genetic algorithms”. Information Sciences, 624:529–553.

[11] M. Ashraf, T. Abdelkader, S. Rady, and T. F. Gharib (2022), “TKN: An efficient approach for discovering top-k high utility itemsets with positive or negative profits”. Information Sciences, 587:654–678.

[12] Rui Sun, Meng Han, Chunyan Zhang, Mingyao Shen, and Shiyu Du (2021), “Mining of top-k high utility itemsets with negative utility”. Journal of Intelligent & Fuzzy Systems, 40(3):5637–5652.

[13] S. Kannimuthu and K. Premalatha (2014), "Discovery of high utility itemsets using genetic algorithm with ranked mutation". Applied Artificial Intelligence, 28(4):337–359.

[14] Jerry Chun-Wei Lin, Youcef Djenouri, Gautam Srivastava, Unil Yun, and Philippe Fournier-Viger (2021), "A predictive GA-based model for closed high-utility itemset mining". Applied Soft Computing, 108:107422, Article number: 107422.

[15] Jerry Chun-Wei Lin, Youcef Djenouri, Gautam Srivastava, and Philippe Fournier-Viger (2022), "Efficient evolutionary computation model of closed high-utility itemset mining". Applied Intelligence, 52(9):10604–10616.

[16] N. Pazhaniraja, Shakila Basheer, Kalaipriyan Thirugnanasambandam, Rajakumar Ramalingam, Mamoon Rashid, and J. Kalaivani (2023), "Multi-objective boolean grey wolf optimization based decomposition algorithm for high-frequency and high-utility itemset mining". AIMS Mathematics, 8(8):18111–18140.

[17] Q. Zhang, W. Fang, J. Sun, and Q. Wang (2019), "Improved genetic algorithm for high-utility itemset mining". IEEE Access, 7:176799–176813.

PARALLELIZATION OF TOP-K HIGH UTILITY ITEMSET MINING IN REAL-TIME DATA STREAMS USING GENETIC ALGORITHM

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

ABSTRACT— High-Utility Itemset Mining (HUIM) in real-time data streams is a challenging problem due to the infinite, high-speed, and continuously evolving nature of data. Traditional approaches based on minimum utility thresholds (minUtil) often face difficulties in parameter selection, which can lead to the loss of important patterns and high computational costs. This paper addresses the problem by proposing a parallelized genetic algorithm for Top-K HUIM in data streams. The proposed method integrates a sliding window model with parallelized fitness evaluation, enabling real-time processing while maintaining high accuracy. Data are represented using bitmap and hash table structures to optimize memory usage and reduce redundant evaluations during the evolutionary process. Experimental studies on benchmark datasets, including Retail, Mushroom, Chess, and Accidents, demonstrate that the proposed approach significantly improves runtime, memory efficiency, and scalability compared to sequential GA and existing HUIM methods. These results highlight the potential of parallelized genetic algorithms as an effective solution for large-scale and real-time utility mining tasks.

Keywords — high-utility itemset mining; top-k itemsets; data streams; parallelized genetic algorithm; real-time mining; bitmap; hash table; sliding window; large-scale data mining.

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 tại 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.

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

Trần Minh Thái là tiến sỹ Công nghệ thông tin. Hiện tại, TS. Thái là giảng viên và trưởng bộ môn Hệ thống thông tin thuộc khoa Công nghệ thông tin, Trường Đại học Ngoại ngữ -Tin học Tp. Hồ Chí Minh. Lĩnh vực nghiên cứu liên quan đến vấn đề khai thác dữ liệu, ẩn dữ liệu, xử lý dữ liệu lớn và nhận dạng.