Chiến lược vùng đệm thích nghi cho khai thác mẫu tuần tự lợi ích cao đều đặn trên dữ liệu tăng trưởng

Trần Minh Thái*

Khoa Công nghệ Thông tin, Trường Đại học Ngoại ngữ - Tin học TP. Hồ Chí Minh
228 Sư Vạn Hạnh, phường Hòa Hưng, TP. Hồ Chí Minh, Việt Nam

Ngày nhận bài 2/12/2025, ngày gửi phản biện 22/12/2025, ngày nhận phản biện 31/12/2025

Tóm tắt:

Khai thác mẫu tuần tự lợi ích cao đều đặn (RHUSP) trên dữ liệu tăng trưởng đối mặt với thách thức lớn trong việc cân bằng giữa hiệu suất tính toán và độ chính xác. Các thuật toán hiện tại sử dụng tỷ lệ vùng đệm cố định (μ) thường gặp hạn chế: giá trị μ thấp gây lãng phí tài nguyên bộ nhớ, trong khi μ cao dẫn đến rủi ro bỏ sót các mẫu quan trọng. Bài báo này đề xuất thuật toán Adaptive-RIncHusp với chiến lược vùng đệm thích nghi, cho phép tự động điều chỉnh tham số μ tại mỗi lô dữ liệu cập nhật. Phương pháp này tích hợp ba quy tắc suy diễn bổ trợ gồm: tỷ lệ lợi ích, độ khó ngưỡng và tốc độ tăng trưởng, kết hợp với hàm rủi ro độ nhạy cao ($f(x)=x^{0.25}$) để phát hiện sớm các biến động dữ liệu. Thực nghiệm trên 5 tập dữ liệu chuẩn chứng minh Adaptive-RIncHusp nhanh hơn 8-15% và tiết kiệm 20-40% bộ nhớ so với chiến lược cố định an toàn (μ=0.4) mà vẫn duy trì độ phủ trên 97%. Đồng thời, thuật toán đạt tốc độ tương đương với chiến lược cố định tích cực (μ=0.9) nhưng khắc phục hoàn toàn vấn đề mất mẫu, khẳng định tính hiệu quả trong môi trường dữ liệu biến động.

Từ khóa: **Khai thác dữ liệu, mẫu tuần tự lợi ích cao, mẫu đều đặn, khai thác tăng trưởng, vùng đệm thích nghi, tinh chỉnh tham số động**

Adaptive buffer strategy for regular High-Utility sequential pattern Mining on incremental data

Minh Thai Tran*

Faculty of Information Technology, University of Foreign Languages - Information Technology
228 Su Van Hanh Street, Hoa Hung Ward, Ho Chi Minh City, Vietnam

Received 2 December 2025; revised 24 December 2025; accepted 29 February 2025

Abstract:

Regular High-Utility Sequential Pattern Mining (RHUSP) from incremental databases poses a significant challenge in balancing computational efficiency and accuracy. Current algorithms using a fixed buffer ratio (μ) often suffer from limitations: a low μ value results in wasted memory resources, while a high μ carries the risk of missing important patterns. This paper proposes Adaptive-RIncHusp, an algorithm featuring an adaptive buffer strategy that automatically adjusts the parameter μ for each data batch update. The method integrates three complementary heuristics: utility Ratio, Threshold difficulty, and growth rate, combined with a high-sensitivity risk function ($f(x)=x^{0.25}$) for early detection of data fluctuations. Experiments on 5 benchmark datasets demonstrate that Adaptive-RIncHusp is 8-15% faster and saves 20-40% memory compared to the conservative fixed strategy (μ=0.4) while maintaining recall > 97%. Furthermore, the algorithm achieves speeds comparable to the aggressive fixed strategy (μ=0.9) while completely eliminating pattern loss, confirming its effectiveness in volatile data environments.

Keywords: **data mining, high-utility sequential patterns, regular patterns, incremental mining, adaptive buffer, dynamic parameter tuning.**

Email: thaitm@huflit.edu.vn

DOI: 10.31276/CNTT.2025.1425

I. Giới thiệu

Khai thác mẫu tuần tự lợi ích cao đều đặn (Regular High-Utility Sequential Patterns - RHUSP) là một bài toán quan trọng trong lĩnh vực khai thác dữ liệu, với nhiều ứng dụng thực tế trong phân tích hành vi khách hàng, dự đoán xu hướng thị trường, và hệ thống khuyến nghị [1, 2]. Khác với khai thác mẫu tuần tự thông thường chỉ dựa trên tần suất xuất hiện, RHUSP xem xét đồng thời ba yếu tố quan trọng: (i) thứ tự thời gian của các sự kiện, (ii) giá trị lợi ích (utility) của mỗi mẫu phản ánh lợi nhuận thực tế và (iii) tính đều đặn (regularity) đo lường sự ổn định xuất hiện của mẫu trong toàn bộ cơ sở dữ liệu [3].

1.1. Vấn đề và thách thức

Trong môi trường dữ liệu tăng trưởng liên tục như thương mại điện tử, mạng xã hội và IoT, việc khai thác lại toàn bộ cơ sở dữ liệu sau mỗi lần cập nhật là không khả thi về mặt tính toán. Các thuật toán khai thác tăng trưởng (incremental mining) hiện đại giải quyết vấn đề này bằng kỹ thuật lưu đệm (buffering): lưu trữ các mẫu bán lợi ích (Semi-High-Utility Sequences - SHS), tức những mẫu chưa đủ điều kiện là lợi ích cao (High-Utility) nhưng có tiềm năng trở thành lợi ích cao sau các lần cập nhật tiếp theo.

Tham số quan trọng nhất của kỹ thuật lưu đệm là tỷ lệ vùng đệm μ (với $0<\mu\leq 1$), quyết định ngưỡng để một mẫu được lưu trữ. Một mẫu P được lưu đệm nếu $u(P)\geq\mu\times minUtility$ và $reg(P)\leq\delta maxReg$. Tuy nhiên, các phương pháp hiện tại [4] sử dụng giá trị μ cố định trong suốt quá trình khai thác, dẫn đến hai vấn đề nghiêm trọng.

Vấn đề 1 (Cơ chế vùng đệm có chọn lọc - Conservative): Nếu μ thấp (ví dụ: 0.4), quá nhiều SHS được lưu trữ. Điều này đảm bảo độ phủ (recall) tối đa nhưng gây tốn kém bộ nhớ (tăng 40-60%) và thời gian tính toán do phải xử lý nhiều mẫu không cần thiết.

Vấn đề 2 (Cơ chế đệm tích cực - Aggressive): Nếu μ cao (ví dụ: 0.9), rất ít SHS được lưu trữ, tiết kiệm tài nguyên nhưng dẫn đến mất mẫu thực (false negatives) nghiêm trọng khi dữ liệu tăng trưởng không đồng đều.

Thực tế, không tồn tại một giá trị μ cố định tối ưu cho mọi tập dữ liệu và mọi giai đoạn tăng trưởng. Các yếu tố ảnh hưởng bao gồm: kích thước lô dữ liệu mới ($|\Delta SD|$) so với cơ sở dữ liệu hiện tại, phân phối lợi ích (utility distribution) trong lô mới, mức độ thay đổi của ngưỡng $minUtility$ (tăng, giảm, đột ngột hay ổn định) và đặc điểm dữ liệu (dày đặc/thưa, phân bố đều/lệch).

1.2. Đóng góp chính

Để giải quyết vấn đề sự đánh đổi giữa tốc độ và độ chính xác trong khai thác tăng trưởng RHUSP, bài báo này đề xuất Adaptive-RIncHusp với chiến lược vùng đệm thích nghi. Các đóng góp chính bao gồm:

Đề xuất ba quy tắc suy diễn (heuristic) bổ trợ để tính toán μ thích nghi. Quy tắc Tỷ lệ lợi ích (utility ratio) đo tỷ lệ lợi ích của lô dữ liệu mới so với tổng lợi ích hiện tại, phát hiện các lô có tác động lớn. Quy tắc độ khó ngưỡng (threshold Difficulty) đánh giá mức độ thay đổi của ngưỡng $minUtility$, phản ánh độ khó của bài toán. Quy tắc Tốc độ tăng trưởng (growth rate) đo tốc độ tăng trưởng của cơ sở dữ liệu, tối ưu hóa hiệu suất tính toán.

Thiết kế hàm rủi ro với độ nhạy cao, sử dụng $f(x) = x0.25$ thay vì tuyến tính để phát hiện sớm các thay đổi nhỏ trong dữ liệu, cho phép thuật toán phản ứng kịp thời.

Đề xuất chiến lược kết hợp (COMBINED) với cơ chế an toàn, lấy giá trị μ nhỏ nhất từ ba heuristic, đảm bảo lưu đệm đủ mẫu trong mọi trường hợp biên.

1.3. Bố cục bài báo

Phần còn lại của bài báo được tổ chức như sau: Mục 2 trình bày bảng tóm tắt các ký hiệu được dùng trong bài báo. Mục 3 trình bày nghiên cứu liên quan về khai thác mẫu tuần tự và các phương pháp tăng trưởng. Mục 4 giới thiệu các định nghĩa hình thức, tính chất lý thuyết và phát biểu bài toán. Mục 5 mô tả chi tiết thuật toán Adaptive-RIncHusp với ba heuristic thích nghi. Mục 6 trình bày thiết kế thực nghiệm, kết quả đánh giá và phân tích so sánh. Cuối cùng, mục 7 tóm tắt các kết quả chính và đề xuất hướng nghiên cứu tương lai.

2. KÝ HIỆU VÀ THUẬT NGỮ

Bảng 1 tóm tắt các ký hiệu được sử dụng trong nội dung của bài báo.

Bảng 1. Bảng ký hiệu.

Ký hiệu	Ý nghĩa
Dữ liệu và Mẫu	
D	Cơ sở dữ liệu hiện tại
D_0	Cơ sở dữ liệu ban đầu
ΔD	Lô dữ liệu tăng trưởng
$\|D\|$	Số lượng chuỗi trong cơ sở dữ liệu
$\|\Delta D\|$	Kích thước lô dữ liệu
I	Tập tất cả các mục (items)
HS	Tập mẫu tuần tự lợi ích cao
SHS	Tập mẫu tuần tự bán lợi ích
Lợi ích và Ngưỡng	
$U(D)$	Tổng lợi ích của cơ sở dữ liệu D
$U(\Delta D)$	Tổng lợi ích của lô ΔD
$u(P, S)$	Lợi ích của mẫu P trong chuỗi S
$u(P)$	Tổng lợi ích của mẫu P
θ	Ngưỡng $minUtility$ (tỷ lệ %)
θ_{abs}	Ngưỡng $minUtility$ tuyệt đối ($= \theta \times U(D)$)
τ	Ngưỡng vùng đệm ($= \mu \times \theta_{abs}$)
ρ	Ngưỡng $maxRegularity$ (tỷ lệ %)
ρ_{abs}	Ngưỡng $maxRegularity$ tuyệt đối ($= \rho \times \|D\|$)
Vùng đệm	
μ	Tỉ lệ vùng đệm (buffer ratio)
μ_{init}	Giá trị khởi tạo của μ
μ_{min}, μ_{max}	Phạm vi của μ (mặc định: [0.4, 0.9])
μ_a	Tỉ lệ vùng đệm thích nghi (COMBINED)
μ_u	Tỉ lệ vùng đệm dựa trên Tỷ lệ lợi ích
μ_b	Tỉ lệ vùng đệm dựa trên Độ khó ngưỡng
μ_g	Tỉ lệ vùng đệm dựa trên Tốc độ tăng trưởng
Tỷ lệ và Hệ số	
r_u	Tỷ lệ lợi ích: $U(\Delta D)/U(D)$
r_g	Tỷ lệ tăng trưởng: $\|\Delta D\|/\|D\|$
f_u, f_b, f_g	Hệ số rủi ro (risk factors) từ 3 heuristic
Hàm	
$\mathcal{R}(x, l, h)$	Hàm rủi ro: HighSensitivityRisk(x, l, h)

3. NGHIÊN CỨU LIÊN QUAN

Sự phát triển của lĩnh vực khai thác dữ liệu trong những năm gần đây chứng kiến sự chuyển dịch mạnh mẽ từ các kỹ thuật khai thác tĩnh sang các phương pháp thích nghi (adaptive), nhằm đáp ứng yêu cầu xử lý dữ liệu dòng và dữ liệu tăng trưởng. Phần này tổng quan các tiến bộ cốt lõi trong khai thác mẫu tuần tự lợi ích cao (HUSPM) và các chiến lược cập nhật dữ liệu hiện đại để làm rõ bối cảnh ra đời của thuật toán đề xuất.

3.1. Từ khai thác vét cạn đến khai thác có mục tiêu

Các thuật toán khai thác mẫu tuần tự (SPM) truyền thống như PrefixSpan [5] hay GSP [6] thường đối mặt với hạn chế lớn là sinh ra số lượng mẫu khổng lồ nhưng thiếu giá trị thực tiễn. Để khắc phục điều này, HUSPM ra đời với mục tiêu tích hợp độ đo lợi ích (utility) vào quá trình khai thác. Theo bài khảo sát toàn diện của R. Zhang và cs [7], xu hướng chủ đạo của các thuật toán HUSPM hiện đại là chuyển dịch từ cấu trúc cây (tree-based) sang cấu trúc danh sách (utility-list) nhằm tối ưu hóa hiệu suất bộ nhớ.

Đáng chú ý, trong giai đoạn 2023-2024, cộng đồng nghiên cứu bắt đầu thay đổi cách tiếp cận từ "khai thác toàn bộ" (exhaustive mining) sang "khai thác có mục tiêu" (targeted mining). Huang và cs [8] đã đề xuất thuật toán TaSPM, cho phép truy vấn trực tiếp các mẫu quan tâm mà không cần quét toàn bộ không gian tìm kiếm. Xu hướng này đặt ra thách thức cho các thuật toán tổng quát như RIncHusp là phải cải thiện khả năng lọc nhiễu và tập trung vào các mẫu thực sự có ý nghĩa.

3.2. Mở rộng khái niệm: tính đều đặn và ổn định

Bên cạnh lợi ích, tính đều đặn là thước đo quan trọng để đánh giá độ tin cậy của mẫu trong môi trường biến động.

Tính đều đặn (Regularity): Thuật toán RIncHusp [4] được xem là công trình nền tảng định nghĩa tính đều đặn dựa trên khoảng cách tối đa giữa các lần xuất hiện. Tuy nhiên, hạn chế chính của RIncHusp là việc sử dụng vùng đệm (buffer) với tỷ lệ cố định, thiếu linh hoạt khi đối mặt với dữ liệu bùng nổ.

Tính ổn định (Stability): Năm 2024, các khái niệm này được nâng cấp phức tạp hơn. G. Huang và cộng sự [9] giới thiệu IPHM trên tạp chí Heliyon, sử dụng cấu trúc danh sách lợi ích tăng trưởng để duy trì mẫu có tính chu kỳ.

Song song đó, C. Xie và cộng sự [10] đề xuất khái niệm "Stable Periodic", lập luận rằng một mẫu chỉ có giá trị thực tiễn nếu chu kỳ xuất hiện của nó duy trì sự ổn định theo thời gian.

3.3. Chiến lược cập nhật: pre-large và vùng đệm thích nghi

Trong môi trường tăng trưởng, việc hạn chế quét lại cơ sở dữ liệu là then chốt.

Chiến lược Pre-large được H. Yan và cs (2024) [11] công bố qua thuật toán Pre-HUSPM, sử dụng khái niệm "Pre-large" với hai ngưỡng hỗ trợ để tạo vùng an toàn. Hệ thống chỉ quét lại khi dữ liệu mới vượt quá ngưỡng an toàn. Tuy nhiên, đây vẫn là chiến lược tĩnh do các ngưỡng được định nghĩa trước.

Chiến lược vùng đệm thích nghi (Adaptive Buffer) được B.S. Prashanth và cs (2025) [12] công bố trên IEEE Access. Ngược lại với vùng an toàn tĩnh, nghiên cứu này đã chứng minh rằng việc điều chỉnh kích thước bộ đệm dựa trên độ biến động (concept drift) giúp duy trì độ chính xác tốt hơn phương pháp cố định. Bài báo này áp dụng tư duy "Adaptive Buffering" [12] vào bài toán HUSPM để khắc phục sự cứng nhắc của chiến lược "Pre-large" [11].

3.4. Tối ưu tham số: quy tắc suy diễn so với AutoML

Một hướng đi khác là sử dụng AI để tự động tinh chỉnh tham số, ví dụ như framework NiaAutoARM [13] sử dụng Automated Machine Learning (AutoML). Tuy nhiên, chi phí tính toán cho AutoML thường quá lớn cho ứng dụng thời gian thực. Do đó, cách tiếp cận sử dụng quy tắc suy diễn thích nghi (Adaptive Heuristics) như đề xuất trong bài báo này là giải pháp cân bằng (trade-off) tối ưu: chi phí tính toán thấp nhưng vẫn đạt khả năng thích nghi cao.

Tóm lại, bức tranh nghiên cứu giai đoạn 2024-2025 cho thấy sự thiếu vắng của một thuật toán khai thác mẫu tuần tự vừa đảm bảo tính lợi ích và đều đặn (như RIncHusp), vừa có khả năng thích nghi tham số thời gian thực (như Adaptive Buffering) mà không tốn kém chi phí tính toán như AutoML. Adaptive-RIncHusp được đề xuất để lấp đầy khoảng trống này.

4. CƠ SỞ LÝ THUYẾT VÀ PHÁT BIỂU BÀI TOÁN

Phần này trình bày các định nghĩa hình thức về khai thác mẫu tuần tự lợi ích cao đều đặn (RHUSP), các tính chất lý thuyết làm cơ sở cho việc cắt tỉa không gian tìm kiếm, và phát biểu bài toán khai thác trong môi trường tăng trưởng.

4.1. Các định nghĩa

Trong bối cảnh đó, Adaptive-RIncHusp kế thừa tính đều đặn của RIncHusp nhưng bổ sung khả năng "thích nghi" để giải quyết bài toán mà tham số tĩnh không thể xử lý. Cho tập hợp các mục (items) $I = \{i_1, i_2, \ldots, i_m\}$. Mỗi mục $i\ j$ được gán một giá trị lợi nhuận đơn vị, gọi là lợi ích ngoại (external utility), ký hiệu là $eu(i_j)$.

Định nghĩa 4.1 (Cơ sở dữ liệu chuỗi): Một cơ sở dữ liệu chuỗi SD là tập hợp các bộ $\langle sid, S\rangle$, trong đó sid là định danh duy nhất và S là một chuỗi. Một chuỗi $S = \langle e_1, e_2, \ldots, e_l\rangle$ là một danh sách có thứ tự của các sự kiện (events), trong đó mỗi sự kiện e_k là một tập hợp các mục. Trong mỗi sự kiện, mỗi mục i có một số lượng, gọi là lợi ích nội tại (internal utility), ký hiệu $iu(i, e_k)$.

Định nghĩa 4.2 (Lợi ích): Lợi ích được xác định theo nhiều cấp độ. lợi ích của mục i trong sự kiện e được tính bằng $u(i, e) = iu(i, e) \times eu(i)$. Lợi ích của sự kiện e là tổng lợi ích của các mục trong sự kiện đó: $u(e) = \sum_{i \in e} u(i, e)$. Lợi ích của mẫu P trong chuỗi S, ký hiệu $u(P, S)$, được tính khi mẫu P xuất hiện trong S qua các sự kiện $ek_1, \ldots, ek_p$, là tổng lợi ích của các mục tương ứng trong lần xuất hiện đó; nếu P xuất hiện nhiều lần, ta chọn lần xuất hiện có lợi ích lớn nhất. lợi ích của mẫu P trong SD được tính bằng $u(P) = \sum_{S \in SD} u(P, S)$.

Định nghĩa 4.3 (Tính đều đặn): Cho một mẫu P xuất hiện trong các chuỗi có sid là $\{sid_1, sid_2, \ldots, sid_k\}$ được sắp xếp tăng dần. Để tính toán đầy đủ, ta quy ước $sid0 = 0$ và $sidk{+}1 = |SD| + 1$. Tính đều đặn của P, ký hiệu $reg(P)$, được định nghĩa theo công thức (1):

$$reg(P) = \max_{1 \le j \le k+1} (sid_j - sid_{j-1}) \quad (1)$$

Một mẫu được gọi là đều đặn nếu $reg(P) \le \delta_{maxReg}$, trong đó δ_{maxReg} là ngưỡng do người dùng quy định.

Định nghĩa 4.4 (Mẫu RHUSP): Một mẫu tuần tự P được gọi là mẫu tuần tự lợi ích cao đều đặn (RHUSP) nếu nó thỏa mãn đồng thời hai điều kiện: (1) lợi ích cao, tức $u(P) \ge minUtility$, và (2) tính đều đặn, tức $reg(P) \le \delta_{maxReg}$.

4.2. Tính chất cắt tỉa

Một thách thức lớn trong HUSPM là lợi ích không tuân theo tính chất đơn điệu giảm (Anti-monotonicity), nghĩa là một mẫu con có thể có lợi ích thấp hơn mẫu cha. Do đó, không thể dùng $u(P)$ để cắt tỉa trực tiếp. Ta cần một cận trên (upper-bound).

Định nghĩa 4.5 (lợi ích trọng số chuỗi - SWU [14]): lợi ích trọng số chuỗi của một mẫu P, ký hiệu $SWU\ (P)$, là tổng lợi ích của tất cả các chuỗi trong SD mà có chứa P, được định nghĩa theo công thức (2):

$$SWU(P) = \sum_{S \in SD, P \subseteq S} u(S) \quad (2)$$

trong đó $u(S)$ là tổng lợi ích của toàn bộ chuỗi S.

Định lý 4.1 (Tính đơn điệu giảm của SWU [14]): Đối với bất kỳ mẫu P nào, ta luôn có $SWU\ (P) \geq u(P)$. Quan trọng hơn, SWU tuân theo tính chất đơn điệu giảm: Nếu $SWU\ (P) < minUtility$, thì mọi mẫu cha (super-sequence) của P cũng sẽ có $SWU < minUtility$ và chắc chắn không phải là mẫu lợi ích cao.

Chứng minh: Chứng minh được thực hiện qua hai bước như sau.

Bước 1: Chứng minh $SWU(P) \geq u(P)$.

$$SWU(P) = \sum_{S \in SD, P \subseteq S} u(S) \quad (3)$$

$$= \sum_{S \in SD, P \subseteq S} \sum_{i \in S} u(i, S) \quad (4)$$

$$\geq \sum_{S \in SD, P \subseteq S} u(P, S) = u(P) \quad (5)$$

Bất đẳng thức được chứng minh vì $u(P, S) \leq u(S)$ với mọi $P \subseteq S$.

Bước 2: Chứng minh tính đơn điệu giảm.

Cho $P' \supseteq P$ là mẫu cha của P. Tập các chuỗi chứa P' là tập con của tập các chuỗi chứa P:

$$\{S \in SD : P' \subseteq S\} \subseteq \{S \in SD : P \subseteq S\}$$

Do đó, $SWU(P') \leq SWU(P)$. Nếu $SWU(P) < minUtility$, thì $SWU(P') < minUtility$, suy ra $u(P') < minUtility$ theo bất đẳng thức đã chứng minh ở Bước 1.

Định lý 4.1 là cơ sở lý thuyết cho phép cắt tỉa không gian tìm kiếm trong cấu trúc cây DETRHUS

4.3. Cấu trúc cây DETRHUS

Cấu trúc DETRHUS (Dynamic Extended Trie for Regular High-Utility Sequences) là cấu trúc dữ liệu cốt lõi trong thuật toán RIncHusp [4], được thiết kế để lưu trữ và cập nhật hiệu quả các mẫu tuần tự trong môi trường tăng trưởng.

DETRHUS là một cây có gốc (rooted tree) trong đó mỗi bộ nút v lưu trữ bộ thông tin $\langle item, u, reg, sidlast, sChildren, iChildren \rangle$, ý nghĩa như sau: $item$ là mục tương ứng với nút; u là tổng lợi ích tích lũy của mẫu từ gốc đến nút hiện tại; reg là khoảng cách lớn nhất (maxReg) giữa các lần xuất hiện liên tiếp; $sidlast$ là định danh của chuỗi cuối cùng chứa mẫu; $sChildren$ là danh sách con theo mở rộng tuần tự (s-extension, mẫu con xuất hiện ở sự kiện tiếp theo); và $iChildren$ là danh sách con theo mở rộng đồng thời (i-extension, mẫu con xuất hiện cùng sự kiện).

Cấu trúc DETRHUS hỗ trợ ba thao tác chính để duy trì và cập nhật thông tin mẫu tuần tự. Thao tác *UpdateDETRHUS* được thực hiện khi có chuỗi mới S với định danh sid được thêm vào cơ sở dữ liệu. Thuật toán duyệt qua các nút thông tin theo công thức: lợi ích mới $u_{new} = u_{old} + u(P, S)$, tính đều đặn mới $regnew = \max(re_{gold}, sid - sid_{last})$, và cập nhật định danh chuỗi cuối $sid_{last} = sid$. Thao tác *PruneTree* thực hiện cắt tỉa cây bằng cách loại bỏ các nút v thỏa mãn một trong trong cây và cập nhật hai điều kiện: nút có lợi ích thấp hơn ngưỡng vùng đệm ($v.u < \tau$) và là nút lá, hoặc nút có tính đều đặn vượt quá ngưỡng cho phép ($v.reg > \rho_{abs}$). Thao tác *ClassifyPatterns* duyệt qua toàn bộ cây để phân loại các mẫu: nếu $v.u \geq \theta_{abs}$ và $v.reg \leq \rho abs$ thì mẫu được xếp vào tập HS; nếu $v.u \geq \tau$ và $v.reg \leq \rho_{abs}$ thì mẫu được xếp vào tập SHS.

4.4. Bài toán khai thác tăng trưởng và vùng đệm

Trong môi trường dữ liệu tăng trưởng, cơ sở dữ liệu SD thay đổi liên tục: $SD\ new = SDold \cup \Delta SD$.

Định nghĩa 4.6 (Mẫu bán lợi ích - SHS): Để tránh việc quét lại S_{Dold}, các thuật toán cần lưu trữ các mẫu tiềm năng. Một mẫu P là SHS nếu:

$$u(P) \geq \mu \times minUtility \text{ và } reg(P) \leq \delta_{maxReg}$$

trong đó $\mu \in (0, 1]$ là tỷ lệ vùng đệm (buffer ratio).

Bài toán khai thác tăng trưởng RHUSP [4]: Cho cơ sở dữ liệu cũ $SDold$, tập hợp các mẫu HS và SHS đã khai thác từ $SDold$, lô dữ liệu mới ΔSD (increment batch), ngưỡng $minUtility$ và ngưỡng tính đều đặn $\delta maxReg$. Kết quả là cập nhật tập HS' và SHS' trên $SDnew$ với ràng buộc là Tối thiểu hóa việc truy cập lại dữ liệu trong $SDold$

4.5. Ví dụ minh họa

Để làm rõ các khái niệm trên, xét cơ sở dữ liệu $SD0$ và bảng lợi ích ngoại như Bảng 2 và Bảng 3.

Bảng 2. Cơ sở dữ liệu mẫu *SD*0.

SID Chuỗi (Sequence)
$S1$ $\langle(a:1)(c:2,d:6)(e:2)\rangle$
$S2$ $\langle(b:3)(f:5)(d:5,e:1)\rangle$
$S3$ $\langle(a:3)(c:3,d:5)\rangle$
$S4$ $\langle(b:3)(e:3)(f:6)\rangle$
$S5$ $\langle(a:1)(b:4,e:2)\rangle$

Bảng 3. Lợi ích ngoại.

Mục	a	b	c	d	e	f
Giá trị	4	3	7	2	5	1

Giả sử $minUtility = 34$ (khoảng 20% tổng lợi ích) và $\delta maxReg = 3$.

Phân loại mẫu trên SDO: Xét mẫu $\langle c\rangle$, ta có $u(\langle c\rangle) = 14(S1) + 21(S3) = 35 \geq 34$, và Regularity $= \max(1, 2, 2) = 2 \leq 3$, do đó $\langle c\rangle$ là RHUSP. Ngược lại, mẫu $\langle b\ f\ \rangle$ có $u(\langle b\ f\ \rangle) = 29 < 34$, do đó không phải RHUSP.

Kịch bản cập nhật ($\Delta SD1$): Giả sử có thêm 2 chuỗi mới $S6$, $S7$. Tổng lợi ích toàn hệ thống tăng lên, dẫn đến ngưỡng $minUtility$ mới có thể tăng lên (ví dụ: 52). Vấn đề nảy sinh với mẫu $\langle b\ f\ \rangle$ có lợi ích 29: Nếu dùng $\mu = 0{,}9$ (ngưỡng đệm $0{,}9\times34 = 30{,}6$), mẫu $\langle b\ f\ \rangle$ (29) đã bị loại bỏ từ trước, và nếu trong $S6$, $S7$, $\langle b\ f\ \rangle$ xuất hiện nhiều, ta sẽ bị mất mẫu này vĩnh viễn (sót mẫu - false Negative). Ngược lại, nếu dùng $\mu = 0{,}4$ (ngưỡng đệm $0{,}4 \times 34 = 13{,}6$), mẫu $\langle b\ f\rangle$ được giữ lại, nhưng hàng ngàn mẫu rác khác cũng được giữ lại, gây tràn bộ nhớ.

Điều này cho thấy sự cần thiết của một cơ chế xác định μ thông minh hơn thay vì cố định, sẽ được trình bày trong Mục V.

5. PHƯƠNG PHÁP ĐỀ XUẤT: ADAPTIVE-RINCHUSP

5.1. Tổng quan chiến lược

Chiến lược vùng đệm thích nghi được thiết kế dựa trên các giả định và có phạm vi áp dụng cụ thể như sau:

Giả định 1 (Tính liên tục của phân phối lợi ích): Phân phối lợi ích giữa các lô dữ liệu liên tiếp không thay đổi đột ngột một cách cực đoan. Điều này cho phép các quy tắc suy diễn dựa trên lô trước để dự đoán hành vi của lô sau.

Trong trường hợp dữ liệu có trôi dạt khái niệm (concept drift) nghiêm trọng, cần kết hợp thêm các kỹ thuật phát hiện trôi dạt.

Giả định 2 (Tính đại diện của các chỉ số): Ba chỉ số (tỷ lệ lợi ích, độ khó ngưỡng, tốc độ tăng trưởng) phản ánh đầy đủ các yếu tố chính ảnh hưởng đến quyết định lưu đệm. Trong các miền ứng dụng đặc thù, có thể bổ sung thêm các quy tắc suy diễn chuyên biệt.

Phạm vi áp dụng: Chiến lược này phù hợp nhất với các hệ thống có dữ liệu tăng trưởng theo lô (batch incremental), trong đó kích thước lô từ 5% đến 30% cơ sở dữ liệu hiện tại. Đối với dữ liệu dòng (streaming) với cập nhật liên tục từng bản ghi, cần điều chỉnh cơ chế tính toán để giảm chi phí phụ trội (overhead).

Sự đánh đổi giữa hiệu quả và độ chính xác: Ba quy tắc suy diễn được thiết kế theo nguyên tắc "an toàn trước" (safety-first), ưu tiên duy trì độ phủ cao hơn tối ưu hóa tốc độ. Việc chọn giá trị μ nhỏ nhất trong chiến lược COMBINED đảm bảo nếu bất kỳ quy tắc nào phát hiện rủi ro, hệ thống sẽ tự động tăng vùng đệm. Điều này có thể dẫn đến việc lưu đệm dư thừa trong một số trường hợp, nhưng đổi lại là độ phủ cao (>97% trong thực nghiệm).

Chi phí tính toán bổ sung: Việc tính toán μ thích nghi chỉ yêu cầu $O(1)$ thời gian cho mỗi lô dữ liệu, không ảnh hưởng đáng kể đến hiệu năng tổng thể của thuật toán.

5.2. Hàm rủi ro với độ nhạy cao

Để phát hiện sớm các thay đổi nhỏ trong dữ liệu, chúng tôi sử dụng hàm rủi ro phi tuyến R (x, α, β) được định nghĩa như sau:

$$\mathcal{R}(x,\alpha,\beta) = \begin{cases} 0, & x < \alpha \\ \left(\frac{x-\alpha}{\beta-\alpha}\right)^{0.25}, & \alpha \le x \le \beta \\ 1, & x > \beta \end{cases} \quad (6)$$

Trong công thức (6), x là giá trị đầu vào cần đánh giá (ví dụ: $util\ Ratio$, $growthRate$), α là ngưỡng dưới (lower bound) và β là ngưỡng trên (upper bound) để kích hoạt tính toán rủi ro.

Hàm số sử dụng lũy thừa bậc 0,25 (căn bậc 4) thay vì hàm tuyến tính. Điều này giúp đạo hàm của hàm số trở nên rất lớn khi x vừa vượt qua ngưỡng α, cho phép thuật toán phản ứng mạnh mẽ ngay cả với những biến động nhỏ ban đầu (ví dụ: x chỉ vượt α 4% nhưng mức rủi ro được đánh giá lên tới 44,7%).

5.3. Ba quy tắc suy diễn thích nghi

a) Quy tắc suy diễn 1: Tỷ lệ lợi ích

Quy tắc này đo tỷ lệ lợi ích của lô dữ liệu mới so với tổng lợi ích hiện tại theo công thức (7):

$$utilRatio = \frac{u(\Delta SD)}{u(SD_{old})} \quad (7)$$

Nếu $util\ Ratio$ cao (ví dụ: ≥ 20%), lô dữ liệu mới có tác động lớn và có thể thay đổi đáng kể tập RHUSP. Do đó, cần giảm μ để lưu đệm nhiều mẫu hơn theo công thức (8):

$$\mu_{util} = \mu_{max} - R(util\ Ratio, 0{,}01, 0{,}20) \times (\mu_{max} - \mu_{min}) \quad (8)$$

Ngưỡng: $thresholdlow = 0.01$ (1%), $thresholdhigh = 0{,}20$ (20%) Ví dụ: Nếu lô mới đóng góp 15% lợi ích, rủi ro = $(0{,}15/0{,}20)0.25 = 0{,}922$, dẫn đến μ giảm mạnh để lưu đệm nhiều mẫu.

b) Quy tắc suy diễn 2: Độ khó ngưỡng

Quy tắc này đánh giá độ khó của bài toán qua tỷ lệ $minUtility$ so với tổng lợi ích theo công thức (9):

$$minUtilityRatio = \frac{minUtility}{u(SD_{new})} \quad (9)$$

Nếu $minUtilityRatio$ thấp (bài toán khó, nhiều mẫu thỏa mãn), cần lưu đệm nhiều hơn theo công thức (10):

$$\mu_{base} = \mu_{max}\text{-R}\ (minUtilityRatio,\ 0{,}001,\ 0{,}005) \times (\mu_{max} - \mu_{min}) \quad (10)$$

Ngưỡng: $thresholdlow = 0{,}001$ (0,1%), $thresholdhigh = 0{,}005$ (0,5%). Lý do: Ngưỡng rất thấp vì $minUtilityRatio$ thường ở khoảng 0,1-0,5% trong thực tế.

c) Quy tắc suy diễn 3: Tốc độ tăng trưởng

Quy tắc này tối ưu hóa hiệu suất dựa trên tốc độ tăng cơ sở dữ liệu theo công thức (11):

$$growthRate = \frac{|\Delta SD|}{|SD_{old}|} \quad (11)$$

Nếu $growthRate$ cao (cơ sở dữ liệu tăng nhanh), cần cân bằng giữa việc lưu đệm và tốc độ theo công thức (12):

$$\mu_{growth} = \mu_{max}\text{-}R(growthRate,\ 0{,}05,\ 0{,}25) \times (\mu_{max} - \mu_{min}) \quad (12)$$

Ngưỡng: $thresholdlow = 0{,}05$ (5%), $thresholdhigh = 0{,}25$ (25%) Ví dụ: Nếu lô mới chiếm 20% cơ sở dữ liệu, rủi ro = $(0{,}20/0{,}25)0{,}25 = 0{,}946$, μ giảm để xử lý tăng trưởng nhanh.

5.4. Chiến lược kết hợp

Chiến lược kết hợp (COMBINED) lấy giá trị μ nhỏ nhất theo công thức (13):

$$\mu_{ada\ ptive} = \min(\mu_{util}, \mu_{base}, \mu_{growth}) \quad (13)$$

Cơ sở lý thuyết: Việc chọn giá trị nhỏ nhất (minimum) đảm bảo vùng đệm đủ mẫu trong mọi trường hợp biên. Nếu một trong ba heuristic phát hiện rủi ro cao, thuật toán sẽ lưu đệm nhiều hơn để tránh mất mẫu.

5.5. Thuật toán Adaptive-RIncHusp

Thuật toán 1 mô tả chi tiết quá trình khai thác RHUSP với chiến lược vùng đệm động. Đây là mở rộng của RIncHusp [4] với điểm khác biệt chính là việc tính toán μ thích nghi tại mỗi lô dữ liệu.

So sánh với RIncHusp [4]: Điểm khác biệt là dòng 16 và 19, nơi τ được tính bởi μa (thích nghi) thay vì μ cố định. Tất cả các bước khác (RHusp, UpdateDETRHUS, PruneTree, ClassifyPatterns) kế thừa từ RIncHusp.

Thuật toán 1: Adaptive-RIncHusp (Mở rộng từ RIncHusp [4])

Input: D_0: CSDL ban đầu; $\Delta D = \{\Delta D_1, \ldots, \Delta D_k\}$: Chuỗi các lô; θ: Tỷ lệ minUtility; ρ: Tỷ lệ maxRegularity; $[\mu_{min}, \mu_{max}]$: Phạm vi vùng đệm; μ_{init}: Giá trị khởi tạo
Output: HS: Tập các mẫu RHUSP

// **Giai đoạn 1: Khởi tạo trên D_0**
$U_{total} \leftarrow U(D_0)$;
$\theta_{abs} \leftarrow \theta \times U_{total}$;
$\tau \leftarrow \mu_{init} \times \theta_{abs}$;
$(HS, SHS) \leftarrow$ RHusp(D_0, τ, ρ);
$\mathcal{T} \leftarrow$ BuildDETRHUS$(HS \cup SHS)$;
$n \leftarrow |D_0|$;
// **Giai đoạn 2: Xử lý tăng trưởng**
foreach *batch* $\Delta D_i \in \Delta D$ **do**
  $U_\Delta \leftarrow U(\Delta D_i)$;
  $m \leftarrow |\Delta D_i|$;
  // Bước 2.1: Tính μ thích nghi
  $\mu_a \leftarrow$ CalcAdaptiveMu$(U_\Delta, m, U_{total}, n)$;
  // Bước 2.2: Cập nhật thông số toàn cục
  $U_{total} \leftarrow U_{total} + U_\Delta$;
  $n \leftarrow n + m$;
  $\theta_{abs} \leftarrow \theta \times U_{total}$;
  $\tau_{new} \leftarrow \mu_a \times \theta_{abs}$;
  // Bước 2.3: Cập nhật cây và cắt tỉa
  foreach *sequence* $S \in \Delta D_i$ **do**
    UpdateDETRHUS$(\mathcal{T}, S)$;
  end
  $\rho_{abs} \leftarrow \lfloor \rho \times n \rfloor$;
  PruneTree$(\mathcal{T}, \tau_{new}, \rho_{abs})$;
end
// **Giai đoạn 3: Phân loại kết quả cuối cùng**
$(HS, SHS) \leftarrow$ ClassifyPatterns$(\mathcal{T}, \theta_{abs}, \tau_{new}, \rho_{abs})$;
return HS;

5.6. Hàm tính tỷ lệ vùng đệm thích nghi

Đây là đóng góp cốt lõi của bài báo. Thuật toán 2 thay thế việc sử dụng μ cố định bằng cơ chế tính toán động dựa trên đặc điểm của từng lô dữ liệu. Các tham số ngưỡng được thiết lập như sau: Tỷ lệ lợi ích trong khoảng [0,01, 0,20], nghĩa là lô dữ liệu đóng góp 1-20% lợi ích được xem là đáng kể. Độ khó ngưỡng trong khoảng [0,001, 0,005], vì θ thực tế thường nằm trong khoảng 0,1-0,5%. Tốc độ tăng trưởng trong khoảng [0,05, 0,25], tức lô dữ liệu chiếm 5-25% kích thước cơ sở dữ liệu được xem là lớn.

5.7. Hàm rủi ro độ nhạy cao

Thuật toán 3 định nghĩa hàm rủi ro với công thức $x0{,}25$ để khuếch đại các thay đổi nhỏ.

Ví dụ minh họa tác động của hàm R(x)= x0,25: Khi x=0,01, ta có R (x)=0,316 (31,6% rủi ro) trong khi hàm tuyến tính chỉ cho 1%. Khi x = 0,04, R (x) = 0,447 (44,7% rủi ro) so với tuyến tính chỉ 4%. Khi x = 0,16, R (x) = 0,632 (63,2% rủi ro) so với tuyến tính 16%.

Hàm này giúp phát hiện sớm các thay đổi nhỏ (1-5%) mà có thể ảnh hưởng đáng kể đến kết quả khai thác.

Thuật toán 2: CalcAdaptiveMu – Tính tỷ lệ vùng đệm thích nghi

Input: U_Δ: lợi ích của lô; m: Kích thước lô; U_{curr}: Tổng lợi ích hiện tại; n: Số lượng chuỗi hiện tại
Output: μ_a: Tỷ lệ vùng đệm cho lô hiện tại

if $U_{curr} = 0$ ***or*** $n = 0$ **then**
  return μ_{init};
end
// **1. Heuristic dựa trên Tỷ lệ lợi ích**
$r_u \leftarrow U_\Delta / U_{curr}$;
$f_u \leftarrow$ HighSensitivityRisk$(r_u, 0.01, 0.20)$;
$\mu_u \leftarrow \mu_{max} - f_u \times (\mu_{max} - \mu_{min})$;
// **2. Heuristic dựa trên Độ khó của ngưỡng**
// θ được lấy từ cấu hình toàn cục
$f_b \leftarrow$ HighSensitivityRisk$(\theta, 0.001, 0.005)$;
$\mu_b \leftarrow \mu_{max} - f_b \times (\mu_{max} - \mu_{min})$;
// **3. Heuristic dựa trên Tốc độ tăng trưởng**
$r_g \leftarrow m/n$;
$f_g \leftarrow$ HighSensitivityRisk$(r_g, 0.05, 0.25)$;
$\mu_g \leftarrow \mu_{max} - f_g \times (\mu_{max} - \mu_{min})$;
// **Chiến lược kết hợp (COMBINED)**
$\mu_{temp} \leftarrow \min(\mu_u, \mu_b, \mu_g)$; // Lấy giá trị an toàn nhất
$\mu_a \leftarrow \max(\mu_{min}, \min(\mu_{temp}, \mu_{max}))$; // Đảm bảo trong khoảng
return μ_a;

Thuật toán 3: HighSensitivityRisk(x, α, β)

Input: x: Giá trị đầu vào; α: Ngưỡng dưới; β: Ngưỡng trên.
Output: $risk \in [0, 1]$: Mức độ rủi ro.

if $x \le \alpha$ **then**
  return 0.0;
end
else if $x \ge \beta$ **then**
  return 1.0;
else
  $norm \leftarrow (x - \alpha)/(\beta - \alpha)$; // Chuẩn hóa về [0, 1]
  $risk \leftarrow norm^{0.25}$; // Khuếch đại độ nhạy
  return $risk$;
end

5.8. Thuật toán phân loại mẫu

Thuật toán 4 kế thừa từ RIncHusp [4] nhưng sử dụng τ (ngưỡng đệm thích nghi) thay vì giá trị cố định.

So sánh với RIncHusp [4]: Điểm khác biệt là dòng 8 sử dụng τ được tính bởi μa thích nghi thay vì μ cố định.

5.9. Độ phức tạp lý thuyết

Định lý 5.1 (Độ phức tạp thời gian): Độ phức tạp thời gian của Adaptive-RIncHusp cho một lô ΔSD là $O(|\Delta SD| \times |I|2 + |HS| + |SHS| + C)$, trong đó C là số lượng mẫu ứng viên được sinh ra từ DETRHUS.

Chứng minh: Chứng minh dựa trên phân tích từng thành phần của thuật toán.

Thuật toán 4: ClassifyPatterns (Mở rộng từ RIncHusp [4])

Input: Node hiện tại, đường dẫn từ root, θ_{abs}, τ (thích nghi), ρ_{abs}
Output: HS, SHS

```
if node ≠ root then
    p_final ← |D| − node.lastSid ;   // Chu kỳ cuố
    reg_true ← max(node.maxReg, p_final) ;   // Tính
     đều đặn thực
    if reg_true ≤ ρ_abs then
        if node.utility ≥ θ_abs then
            Thêm mẫu vào HS;
        end
        else if node.utility ≥ τ then
            Thêm mẫu vào SHS ;      // Sử dụng
             thích nghi
    end
end
for each child in node.sChildren do
    ClassifyPatterns(child, path ∪ [s : child.item], ...);
end
for each child in node.iChildren do
    ClassifyPatterns(child, path ∪ [i : child.item], ...);
end
```

Thành phần 1 (Tính $\mu ada\ ptive$): Việc tính toán ba quy tắc suy diễn và lấy giá trị nhỏ nhất chỉ yêu cầu số lượng phép tính cố định, do đó có độ phức tạp O (1).

Thành phần 2 (Cập nhật cây DETRHUS): Với mỗi chuỗi trong ΔSD, thuật toán duyệt qua các mục trong chuỗi để cập nhật cây. Trong trường hợp xấu nhất, mỗi chuỗi có thể chứa O ($|I|2$) mẫu con (kết hợp 2 mục), dẫn đến độ phức tạp O ($|\Delta SD| \times |I|2$).

Thành phần 3 (Đánh giá lại mẫu): Duyệt qua các mẫu trong HS và SHS để kiểm tra điều kiện với ngưỡng mới yêu cầu O ($|HS| + |SHS|$).

Thành phần 4 (Khai thác ứng viên mới): Sinh các mẫu ứng viên từ DETRHUS có độ phức tạp O (C), với C là số ứng viên và thường $C \ll$ tổng số mẫu trong không gian tìm kiếm nhờ cơ chế cắt tỉa.

Tổng hợp các thành phần: $T(n) = O(1) + O(|\Delta SD| \times |I|2) + O(|HS| + |SHS|) + O(C) = O(|\Delta SD| \times |I|2 + |HS| + |SHS| + C)$.

Định lý 5.2 (Độ phức tạp bộ nhớ): Độ phức tạp bộ nhớ là O ($|SD| \times |I| + |HS| + |SHS|$).

Chứng minh: Bộ nhớ được sử dụng bởi ba thành phần chính.

Thành phần 1 (Cây DETRHUS): Trong trường hợp xấu nhất, cây lưu trữ thông tin về O ($|SD| \times |I|$) nút, mỗi nút chứa một lượng thông tin cố định ($u, reg, sidlast$, con trỏ).

Thành phần 2 (Vùng đệm): Tập HS và SHS yêu cầu O ($|HS| + |SHS|$) không gian để lưu trữ các mẫu và thông tin liên quan.

Thành phần 3 (Dữ liệu thô): Thuật toán không cần lưu trữ dữ liệu thô $SDold$, chỉ cần lưu các thống kê trong DETRHUS.

Tổng hợp: $S(n) = O(|SD| \times |I|) + O(|HS| + |SHS|) = O(|SD| \times |I| + |HS| + |SHS|)$.

5.10. Tính đúng đắn

Định lý 5.3 (Tính đầy đủ - Completeness): Adaptive-RIncHusp không bao giờ bỏ sót mẫu RHUSP nếu $\mu ada\ ptive \leq \mu o\ ptimal$, trong đó $\mu o\ ptimal$ là giá trị tối thiểu để không mất mẫu.

Chứng minh:

Giả sử: Tồn tại một mẫu $P*$ là RHUSP thực sự (tức $u(P*) \geq minUtility$ và $reg(P*) \leq \delta maxReg$) nhưng bị bỏ sót bởi thuật toán.

Phân tích: Để $P*$ bị bỏ sót, $P*$ phải bị loại khỏi vùng đệm tại một thời điểm nào đó. Điều này xảy ra khi và chỉ khi $u(P*) < \tau = \mu \times minUtility$ tại thời điểm đó.

Chiến lược COMBINED chọn $\mu = \min(\mu util, \mu base, \mu growth)$ theo công thức (13). Nếu bất kỳ quy tắc suy diễn nào phát hiện rủi ro cao, μ sẽ giảm để tạo vùng đệm lớn hơn.

Theo giả thiết $\mu_{adaptive} \leq \mu_{optimal}$, ta có ngưỡng đệm $\tau = \mu_{adaptive} \times minUtility \leq \mu_{optimal} \times minUtility$. Vì $\mu optimal$ được định nghĩa là giá trị tối thiểu để không mất mẫu, mọi mẫu được giữ với $\mu_{optimal}$ cũng được giữ với $\mu_{adaptive}$.

Điều này mâu thuẫn với giả sử P^* bị bỏ sót. Do đó, định lý được chứng minh.

Adaptive-RIncHusp vẫn là thuật toán xấp xỉ (như RIncHusp gốc) và có thể bỏ sót một số mẫu trong trường hợp cực biên khi $\mu_{adaptive} > \mu_{optimal}$. Tuy nhiên, thực nghiệm cho thấy độ phủ $> 97\%$ trên tất cả các tập dữ liệu, chứng tỏ các quy tắc suy diễn hoạt động hiệu quả trong thực tế.

5.11. Ví dụ minh họa thực thi thuật toán

Để làm rõ cơ chế hoạt động của thuật toán Adaptive-RIncHusp, chúng tôi minh họa chi tiết từng bước xử lý dựa trên cơ sở dữ liệu $D0$ từ Bảng II và bảng lợi ích ngoại từ bảng 3.

a) Giai đoạn 1: Khởi tạo với D0

Bước 1.1: Tính toán lợi ích và tham số khởi tạo

Tổng lợi ích các chuỗi: $U\ (S1) = 40, U\ (S2) = 29, U\ (S3) = 43, U\ (S4) = 30, U\ (S5) = 26$, suy ra $U0 = 168$. Với $\theta = 0.2$, $\rho = 0.6$, $\mu init = 0.4$:$\theta 0 = 0.2 \times 168 = 33.6$, $\tau 0 = 0.4 \times 33.6 = 13.44$, $\rho 0 = 3$

Điều kiện phân loại: HS nếu $u \geq 33.6$ và $reg \leq 3$; SHS nếu $13.44 \leq u < 33.6$ và $reg \leq 3$.

Bước 1.2: Khai thác mẫu tuần tự

Bảng IV tổng hợp kết quả khai thác với đầy đủ các dạng mở rộng: 1-sequence, s-extension (mở rộng tuần tự) và i-extension (mở rộng đồng thời).

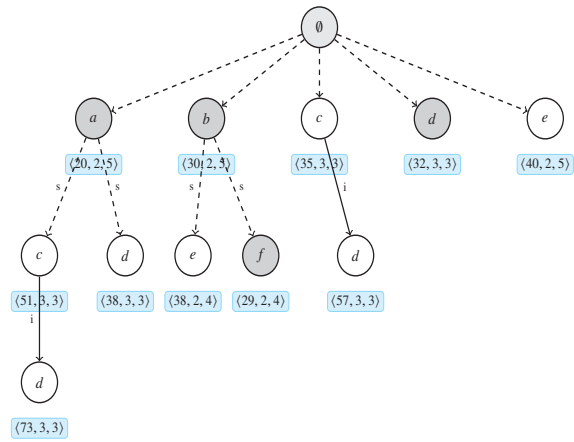
Bảng 4. Khai thác mẫu từ $\theta 0$ ($\theta 0$ = 33.6, $\tau 0$ = 13.44, $\rho 0$=3).

Dạng	Mẫu	u	reg		Loại mẫu
1-seq	⟨a⟩	20	2	5	SHS
	⟨b⟩	30	2	5	SHS
	⟨c⟩	35	3	3	HS
	⟨d⟩	32	3	3	SHS
	⟨e⟩	40	2	5	HS
s-ext	⟨ac⟩	51	3	3	HS
	⟨ad⟩	38	3	3	HS
	⟨be⟩	38	2	4	HS
	⟨bf⟩	29	2	4	SHS
i-ext	⟨(cd)⟩	57	3	3	HS
	⟨a(cd)⟩	73	3	3	HS

Giải thích mẫu $\langle a(cd)\rangle$: Trong $S1$: a ở sự kiện 1, (cd) ở sự kiện 2, $u = 4+14+12 = 30$. Trong $S3$: $u = 12+21+10 = 43$. Tổng $u = 73$.

Bước 1.3: Xây dựng cây DETRHUS

Hình 1 minh họa cây DETRHUS được xây dựng từ $D0$. Mỗi nút lưu trữ bộ ba giá trị $\langle u, reg, sidL\rangle$ tương ứng với lợi ích, tính đều đặn và định danh chuỗi cuối. Trong hình 1, nút không tô màu biểu thị mẫu HS ($u \geq \theta$), nút tô xám biểu thị mẫu SHS ($\tau \leq u < \theta$). Mũi tên nét đứt thể hiện s-extension, mũi tên nét liền thể hiện i-extension.



Hình 1. Cây DETRHUS khởi tạo từ $D0$ với $\theta 0$ = 33.6, $\tau 0$ = 13.44.

b) Giai đoạn 2: Xử lý lô ΔD1

Cơ sở dữ liệu được cập nhật với lô dữ liệu tăng trưởng $\Delta D1$ gồm 2 chuỗi mới như bảng 5.

Bước 2.1: Tính μ thích nghi

Bảng 5. Lô dữ liệu tăng trưởng ΔD1.

SID	Chuỗi	U(S)	Chi tiết
S_6		30	8 + 7 + 15
S_7		20	6 + 4 + 10
Tổng		$U\Delta = 50$	

Bước 2.1: Tính thích nghi với $Utotal = 168$, $U\Delta = 50$, $n = 5$, $m = 2$ (bảng 6).

Bảng 6. Tính toán μ thích nghi cho $\Delta D1$.

Heuristic	Giá trị	$f(x)$	μ	Nhận xét
Tỷ lệ lợi ích	$r_u = 0.229$	1.0	0.4	Vượt ngưỡng
Độ khó ngưỡng	$\theta_r = 0.003$	0.84	0.48	–
Tốc độ tăng	$r_g = 0.286$	1.0	0.4	Vượt ngưỡng
COMBINED			**0.4**	$\min(0.4, 0.48, 0.4)$

Bước 2.2: Cập nhật tham số

$U1 = 218, n1 = 7, \theta 1 = 43.6, \tau 1 = 17.44, \rho 1 = 4$

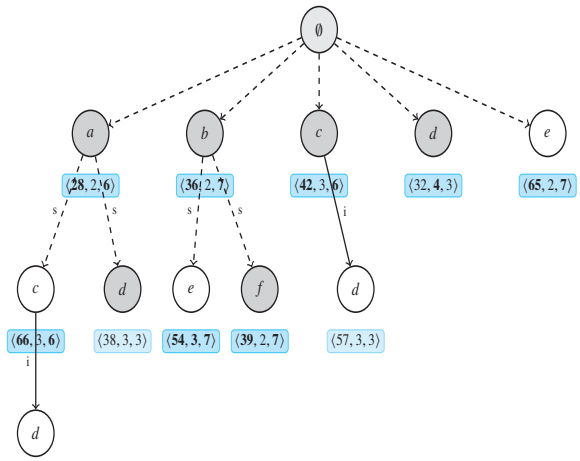
Bước 2.3: Cập nhật DETRHUS

Bảng VII trình bày chi tiết quá trình cập nhật các nút khi xử lý từng chuỗi trong $\Delta D1$.

Bảng 7. Cập nhật detrhus khi xử lý δ$D1$.

Mẫu	u_{old}	Δu	u_{new}	reg	sid_L
Từ S_6:					
$\langle a \rangle$	20	+8	28	2	6
$\langle c \rangle$	35	+7	42	3	6
$\langle e \rangle$	40	+15	55	2	6
$\langle ac \rangle$	51	+15	66	3	6

Hình 2 minh họa cây DETRHUS sau cập nhật. Các giá trị cập nhật được in đậm.



Hình 2. Cây DETRHUS sau cập nhật $\Delta D1$ với $\theta 1$ = 43.6, $\tau 1$ = 17.44.

c) Giai đoạn 3: Phân loại lại và so sánh

Mẫu $\langle c \rangle$ dù u tăng từ 35 lên 42, nhưng ngưỡng θ cũng tăng từ 33.6 lên 43.6 nên bị giáng cấp. Đây là lý do vùng đệm SHS cần thiết, nếu không có SHS, mẫu này sẽ bị mất. Kết quả chi tiết thể hiện trong bảng 8.

So sánh chiến lược: Xét mẫu $\langle a \rangle$ ($u0 = 20$): Nếu Fix(μ =0.9) thì mẫu này sẽ bị loại. Tuy nhiên, với chiến lược thích nghi thì vẫn giữ lại (bảng 9).

Kết luận: Chiến lược thích nghi phát hiện $ru =$ 22.9% và $rg = 28.6\%$ đều vượt ngưỡng, nên tự động chọn $\mu a = 0.4$ để đảm bảo an toàn. Trong các lô tiếp theo nếu dữ liệu ổn định, μ sẽ tự động tăng để tiết kiệm bộ nhớ. Ví dụ minh họa cơ chế: tự động điều chỉnh μ dựa trên đặc điểm dữ liệu, đảm bảo không mất mẫu tiềm năng mà vẫn tối ưu tài nguyên khi có thể.

Bảng 8. So sánh phân loại trước/sau cập nhật

	Trước		Sau		
Mẫu	u	Loại	u	Loại	Ghi chú
$\langle c \rangle$	35	HS	42	SHS	Giáng cấp*
$\langle ad \rangle$	38	HS	38	SHS	Giáng cấp*
$\langle e \rangle$	40	HS	65	HS	–
$\langle ac \rangle$	51	HS	66	HS	–
$\langle be \rangle$	38	HS	54	HS	–
$\langle bf \rangle$	29	SHS	39	SHS	–

* Do $\theta_1 = 43.6 > u$, mẫu chuyển từ HS → SHS

Bảng 9. So sánh chiến lược xử lý mẫu $\langle a \rangle$.

Chiến lược	τ_0	$u = 20$ vs τ_0	Kết quả
Fix(μ = 0.9)	30.24	20 < 30.24	**Mất mẫu**
Fix(μ = 0.4)	13.44	20 ≥ 13.44	Giữ
Adaptive	13.44	20 ≥ 13.44	**Giữ**

6. Thực nghiệm và đánh giá

6.1. Thiết kế thực nghiệm

a) Môi trường thực nghiệm

Tất cả các thuật toán được cài đặt bằng Java và thực thi trên máy tính MacBook Pro với chip Apple M1, bộ nhớ 8 GB RAM, hệ điều hành macOS Tahoe 26.1, Java JDK Version 22, và IDE IntelliJ IDEA 2025.2 Ultimate.

b) Tập dữ liệu và cấu hình tham số

Năm tập dữ liệu chuẩn được sử dụng với các đặc điểm đa dạng về độ thưa, kích thước và số lượng mục.

Bảng 10. Cấu hình tham số cho các tập dữ liệu.

Tập dữ liệu	MinUtil (%)	MaxReg (%)	μ khởi tạo	Phạm vi thích nghi
BIBLE	0.25	30.00	0.40	[0.40 - 0.90]
Chainstore	0.05	30.00	0.40	[0.40 - 0.90]
Foodmart	0.01	100.00	0.40	[0.40 - 0.90]
Leviathan	0.05	30.00	0.40	[0.40 - 0.90]
SIGN	0.20	30.00	0.40	[0.40 - 0.90]

c) Kịch bản phân phối dữ liệu

Để mô phỏng các mẫu tăng trưởng khác nhau của cơ sở dữ liệu, chúng tôi thiết kế 4 kịch bản phân phối dữ liệu qua 4 lần cập nhật (lô). Kịch bản A (Đồng đều) với tỷ lệ 25%, 25%, 25%, 25% mô phỏng tăng trưởng ổn định. Kịch bản B (Tăng dần) với tỷ lệ 10%, 20%, 30%, 40% mô phỏng tăng trưởng gia tốc. Kịch bản C (Dao động) với tỷ lệ 40%, 10%, 40%, 10% mô phỏng bùng nổ dữ

liệu. Kịch bản D (Giảm dần) với tỷ lệ 40%, 30%, 20%, 10% mô phỏng tăng trưởng chậm lại.

Mỗi kịch bản được thực hiện trên tất cả 5 tập dữ liệu, và kết quả được tổng hợp qua cả 4 kịch bản để đánh giá tính ổn định của thuật toán.

d) Các phương pháp so sánh

Chúng tôi so sánh 4 chiến lược thích nghi với 2 baseline ngưỡng cố định.

Chiến lược thích nghi bao gồm: Adapt (Combined) kết hợp cả 3 heuristic và chọn giá trị μ nhỏ nhất; Adapt (Growth) chỉ sử dụng heuristic tốc độ tăng trưởng; Adapt (MinBase) chỉ sử dụng heuristic độ khó ngưỡng; và Adapt (UtilRatio) chỉ sử dụng heuristic tỷ lệ lợi ích.

Baseline cố định bao gồm: Fix (μ=0,4) với ngưỡng thấp, độ chính xác cao nhưng chi phí lớn; và Fix (μ=0,9) với ngưỡng cao, tốc độ nhanh nhưng có rủi ro mất mẫu.

e) Chỉ số đánh giá

Các chỉ số đánh giá bao gồm: Thời gian (s) là tổng thời gian xử lý tất cả các lô; Bộ nhớ đỉnh (MB) là bộ nhớ tối đa sử dụng trong quá trình thực thi; HS Found là số lượng mẫu RHUSP tìm được; SHS Buffered là số lượng mẫu SHS được lưu đệm (chi phí phụ); Độ phủ - Recall (%) được tính |HS Ground Truth| × 100%; và Tăng tốc - Speedup được bằng tính bằng Runtime$Fixed$-0.4 Runtime$Adaptive$.

Tập mẫu chuẩn (Ground truth) được xác định bằng cách chạy thuật toán RHusp [4] trên toàn bộ cơ sở dữ liệu sau khi cập nhật hết tất cả các lô.

6.2. Kết quả hiệu năng tổng thể

Bảng XI tổng hợp kết quả trung bình trên cả 4 kịch bản cho từng tập dữ liệu. Các giá trị tốt nhất trong mỗi nhóm (chiến lược thích nghi) được in đậm.

Nhận xét từ Bảng XI: Về hiệu quả tốc độ, các chiến lược thích nghi đạt cân bằng tốt giữa tốc độ và độ chính xác, trong đó: Adapt (Growth) và Adapt (Combined) cho thời gian chạy tốt nhất trong hầu hết trường hợp, nhanh hơn Fix(μ=0.4) từ 7-17%. Về độ chính xác, Fix(μ=0.9) tuy nhanh nhất nhưng mất 4-37 mẫu so với tập mẫu chuẩn (tương đương 1.5-5.1% độ phủ), trong khi tất cả chiến lược thích nghi đều đạt độ phủ ≥ 98%. Về tiết kiệm bộ nhớ, Adapt (MinBase) thường sử dụng ít SHS nhất nhưng vẫn đảm bảo độ phủ cao, còn Adapt (Combined) cân bằng tốt giữa tốc độ và số lượng bộ đệm. Về khả năng mở rộng, trên tập dữ liệu lớn (Foodmart với 84K mẫu), các chiến lược thích nghi nhanh hơn baseline 66-74%, chứng tỏ thuật toán mở rộng tốt.

Bảng 11. Kết quả tổng hợp trên các tập dữ liệu (trung bình 4 kịch bản).

Tập dữ liệu	Chiến lược	Thời gian (s)	Bộ nhớ (MB)	HS Found	SHS Buffered
BIBLE	CL1	28.82	218.81	266	391
	CL2	**27.78**	212.24	266	364
	CL3	29.03	**205.07**	266	**278**
	CL4	29.76	241.52	266	391
	CL5	29.56	191.42	**266**	427
	CL6	14.88	239.97	259	22
Chainstore	CL1	38.37	**561.92**	**177**	326
	CL2	**35.68**	616.23	**177**	**315**
	CL3	38.24	669.08	174	19
	CL4	39.55	613.50	**177**	326
	CL5	41.39	692.20	**177**	349
	CL6	21.89	537.78	168	15
Foodmart	CL1	0.44	178.25	**84,020**	11,446
	CL2	0.40	170.69	**84,020**	11,369
	CL3	0.41	**167.34**	84,013	**7,572**
	CL4	**0.38**	171.39	**84,020**	11,446
	CL5	1.12	159.55	**84,020**	11,540
	CL6	0.65	189.66	84,010	2,844
Leviathan	CL1	16.76	104.40	**534**	296
	CL2	**16.71**	105.18	**534**	**281**
	CL3	17.33	104.68	525	32
	CL4	16.89	**102.14**	**534**	296
	CL5	17.63	107.41	**534**	324
	CL6	11.20	104.71	520	31
SIGN	CL1	**10.41**	794.40	**3,775**	2,334
	CL2	10.51	**784.68**	**3,775**	**2,236**
	CL3	10.57	792.74	**3,775**	1,890
	CL4	10.62	793.24	**3,775**	2,334
	CL5	12.51	781.27	**3,775**	2,422
	CL6	10.17	765.98	3,738	348

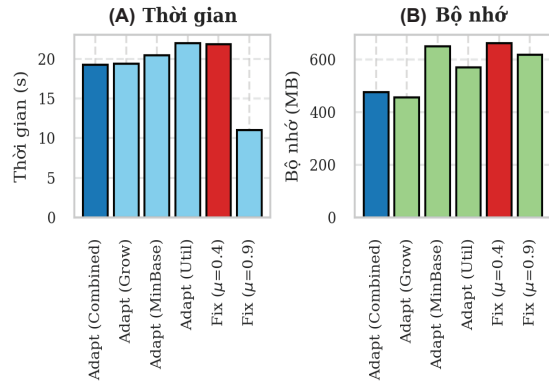
Chú thích: CL1: Adapt (Combined); CL2: Adapt (Growth); CL3: Adapt (MinBase);

6.3. Phân tích chi tiết: Tập dữ liệu Chainstore

Để hiểu rõ cơ chế hoạt động của chiến lược thích nghi, chúng tôi phân tích sâu trên tập Chainstore với Kịch bản B (10%-20%-30%-40%) – một tập dữ liệu thưa với độ biến động cao, nơi sự khác biệt giữa các chiến lược là rõ rệt nhất.

a) So sánh thời gian và bộ nhớ

Hình 3 cho thấy so sánh hiệu năng của các chiến lược trên Chainstore.

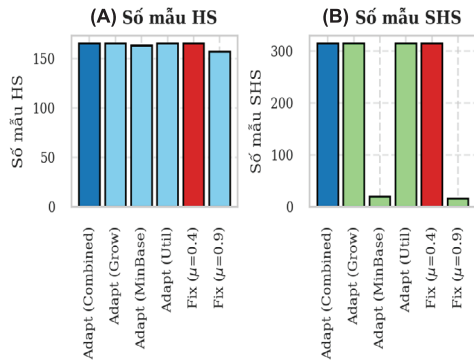


Hình 3. So sánh hiệu năng của các chiến lược trên Chainstore (Kịch bản B). (A) Thời gian chạy (giây), (B) Mức sử dụng bộ nhớ đỉnh (MB).

Quan sát từ hình 3: Về thời gian, Adapt (Growth) đạt thời gian tốt nhất (35,68s), nhanh hơn Fix(μ=0,4) 13,8%, trong khi Adapt (Combined) cũng đạt 38,37s, cải thiện 7,3%. Về bộ nhớ, các chiến lược thích nghi có mức sử dụng biến động (561-669 MB) do điều chỉnh bộ đệm động, nhưng vẫn thấp hơn Fix(μ=0,4) với 692 MB. Về sự đánh đổi, Fix(μ=0,9) tuy nhanh nhất (21,89s) và tiết kiệm nhất (537 MB), nhưng mất nhiều mẫu.

b) Số lượng mẫu lợi ích cao và bán lợi ích

Hình 4 thể hiện số lượng mẫu được phát hiện và lưu đệm.

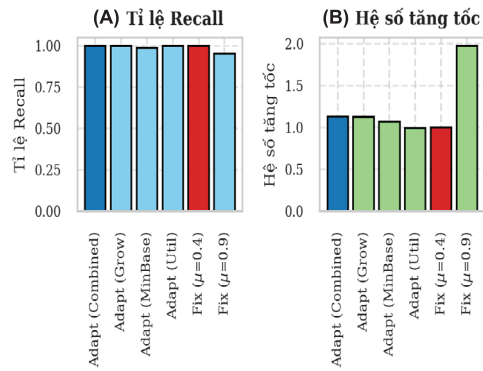


Hình 4. Số lượng mẫu được phát hiện và lưu đệm trên Chainstore (Kịch bản B). (A) Số mẫu HS (Mẫu lợi ích cao), (B) Số mẫu SHS (Mẫu bán lợi ích).

Phân tích từ hình 4: Về HS Found, tất cả các chiến lược thích nghi (Combined, Growth, UtilRatio) tìm được đầy đủ 177 mẫu HS, bằng với tập mẫu chuẩn. Về mất mẫu thực, Fix(μ=0,9) chỉ tìm được 168 HS, mất 9 mẫu (-5,1%), cho thấy rủi ro của việc lưu đệm tích cực. Về SHS Buffered, Adapt (MinBase) lưu đệm ít nhất (19 SHS) nhưng dẫn đến mất 3 mẫu, trong khi Adapt (Combined) lưu đệm 326 SHS, cân bằng hiệu quả giữa chi phí phụ và độ phủ. Về hiệu quả, các chiến lược thích nghi lưu đệm ít hơn Fix(μ=0,4) khoảng 7-9% (315-326 vs 349), giúp tiết kiệm tài nguyên.

c) Tỷ lệ độ phủ và hệ số tăng tốc

Hình 5 đánh giá chất lượng và tốc độ của các chiến lược.



Hình 5. Đánh giá chất lượng và tốc độ trên Chainstore (Kịch bản B). (A) Tỷ lệ Độ phủ so với tập mẫu chuẩn, (B) Hệ số tăng tốc so với Fix(μ=0,4).

Kết quả từ hình 5: Về độ phủ, Adapt (Combined), Adapt (Growth), Adapt (UtilRatio) đều đạt 100% độ phủ, trong khi Adapt (MinBase) đạt 98,3% và Fix(μ=0,9) chỉ đạt 94,9%. Về hệ số tăng tốc, Adapt (Growth) đạt 1.16× nhanh hơn baseline (tốt nhất trong nhóm thích nghi), Adapt (Combined) đạt 1,08× (cân bằng tốt giữa tốc độ và ổn định), Adapt (UtilRatio) đạt 1,05× (tương đương baseline nhưng tiết kiệm bộ đệm), và Fix(μ=0,9) đạt 1,89× (nhanh nhất nhưng mất quá nhiều mẫu). Về điểm cân bằng tối ưu, Adapt (Growth) và Adapt (Combined) đạt điểm cân bằng tối ưu với độ phủ 100% và tốc độ cải thiện đáng kể 8-16%.

Kết luận từ phân tích Chainstore: Chiến lược thích nghi giải quyết hiệu quả vấn đề sự đánh đổi của tỷ lệ vùng đệm cố định. Các chiến lược thích nghi không chỉ nhanh hơn mà còn đảm bảo không bị mất mẫu, phù hợp cho môi trường thực tế với yêu cầu cao về cả hiệu năng lẫn độ chính xác.

Để đánh giá tính ổn định, chúng tôi thử nghiệm AdaptCOMBINED với nhiều giá trị μ khởi tạo khác nhau (0,4, 0,5, 0,6, 0,7, 0,8, 0,9, 0,95) trên tập dữ liệu BIBLE. Kết quả cho thấy: Thời gian giảm từ 28.8s (μ khởi tạo=0,4) xuống 10,5s (μ khởi tạo=0,9); độ phủ duy trì > 97,5% với mọi μ khởi tạo; HS Found giảm nhẹ từ 266 xuống 259 (chấp nhận được); và SHS Buffer giảm từ 391 xuống 252 (cơ chế thích nghi hoạt động tốt).

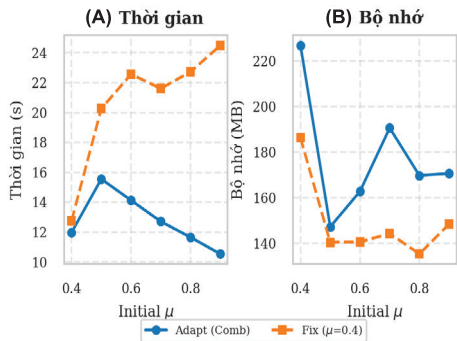
⇒ Chiến lược thích nghi ổn định với tham số khởi tạo. Khuyến nghị: μ khởi tạo = 0,6-0,7 để cân bằng tốc độ và độ chính xác.

6.4. Phân tích độ nhạy của tham số khởi tạo

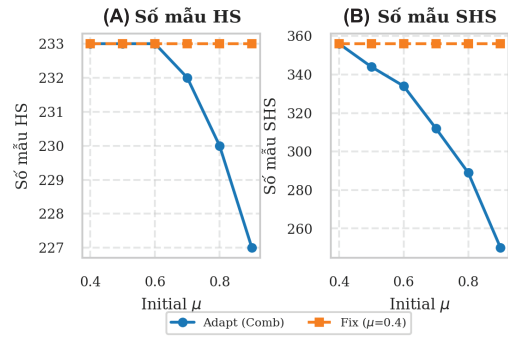
Để đánh giá tính ổn định của thuật toán, chúng tôi thay đổi giá trị khởi tạo của μ từ 0,4 đến 0,9 và quan sát ảnh hưởng đến hiệu năng. Phân tích được thực hiện trên hai tập dữ liệu đại diện: BIBLE (dữ liệu văn bản, dày) và Chainstore (dữ liệu bán lẻ, thưa), sử dụng phân phối tăng trưởng Kịch bản B (10%-20%-30%-40%).

a) Kết quả trên tập BIBLE

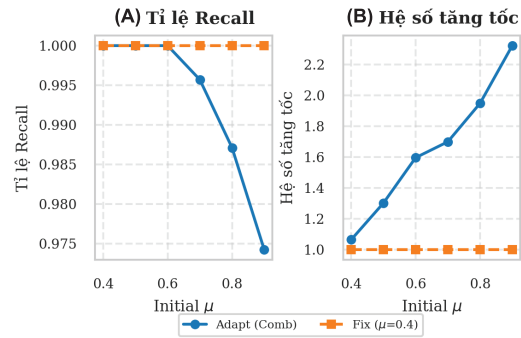
Hình 6 cho thấy ảnh hưởng của μ khởi tạo đối với thời gian và bộ nhớ trên BIBLE.



Hình 6. Ảnh hưởng của - khởi tạo trên BIBLE. (A) Thời gian chạy, (B) Bộ nhớ sử dụng.



Hình 7. Số lượng mẫu theo μ khởi tạo trên BIBLE. (A) Số mẫu HS, (B) Số mẫu SHS lưu đệm.



Hình 8. Chất lượng và hiệu quả theo μ khởi tạo trên BIBLE. (A) Tỷ lệ Độ phủ, (B) Hệ số tăng tốc.

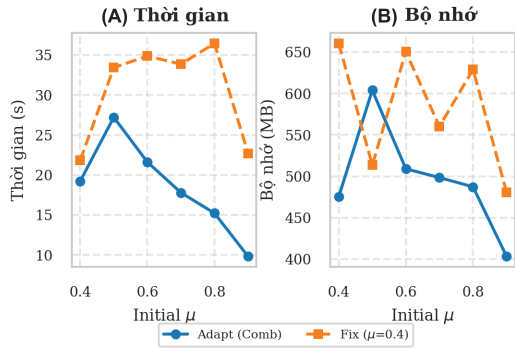
Hình 7 thể hiện số lượng mẫu theo μ khởi tạo và hình 8 đánh giá chất lượng và hiệu quả.

Nhận xét từ kết quả trên tập dữ liệu BIBLE: Về thời gian, giá trị giảm dần khi μ khởi tạo tăng (từ 28,8s với μ=0,4 xuống 10,5s với μ=0,9), do lưu đệm ít SHS hơn nên xử lý nhanh hơn. Về bộ nhớ, giá trị biến động (205-241 MB) do cơ chế thích nghi, nhưng vẫn kiểm soát tốt trong mọi trường hợp. Về độ phủ, duy trì >97,5% với mọi μ khởi tạo, chứng tỏ thuật toán ổn định với tham số khởi tạo. Về HS Found, giảm nhẹ từ 266 xuống 259 khi μ tăng lên 0,95 (chấp nhận được, chỉ mất 2,6%). Về SHS Buffer, giảm mạnh từ 391 xuống 252 khi μ tăng, cho thấy cơ chế thích nghi hoạt động đúng.

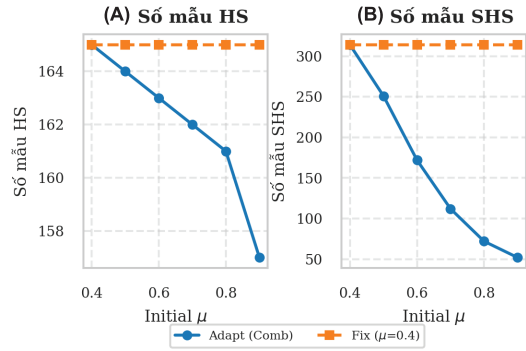
b) Kết quả trên tập Chainstore

Hình 9 cho thấy ảnh hưởng của μ khởi tạo trên Chainstore. Tiếp theo, hình 10 thể hiện số lượng mẫu.

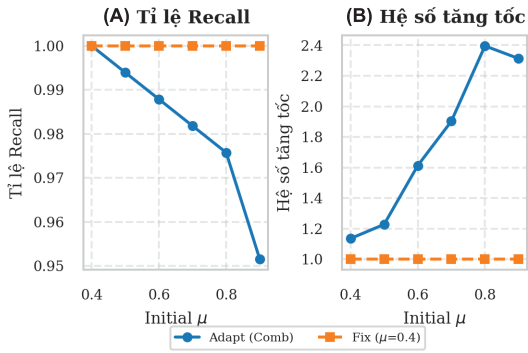
Cuối cùng là biểu đồ trong hình 11 thể hiện đánh giá chất lượng.



Hình 9. Ảnh hưởng của μ khởi tạo trên Chainstore. (A) Thời gian chạy, (B) Bộ nhớ sử dụng.



Hình 10. Số lượng mẫu theo μ khởi tạo trên Chainstore. (A) Số mẫu HS, (B) Số mẫu SHS lưu đệm.



Hình 11. Chất lượng và hiệu quả theo μ khởi tạo trên Chainstore. (A) Tỷ lệ Độ phủ, (B) Hệ số tăng tốc.

Nhận xét từ Chainstore: Về thời gian, xu hướng giảm từ 38.4s (μ=0,4) xuống 10,0s (μ=0,95) khi μ tăng, cải thiện 74%. Về bộ nhớ, dao động mạnh hơn BIBLE (470-605 MB) do đặc tính thưa của dữ liệu. Về độ phủ, giảm từ 100% xuống 95.2% khi $\mu = 0,95$, vẫn chấp nhận được. Về sự đánh đổi, với μ khởi tạo = 0,4-0,6 cho độ phủ cao (>98%) và thời gian vừa phải; với μ khởi tạo = 0,7-0,8 cho cân bằng tốt (độ phủ ~98%, tăng tốc ~1.5×); với μ khởi tạo = 0,9-0,95 nhanh nhất nhưng mất nhiều mẫu hơn.

Kết luận từ phân tích độ nhạy: Thuật toán hoạt động ổn định với nhiều giá trị khởi tạo khác nhau, không yêu cầu tinh chỉnh tham số phức tạp. Khuyến nghị sử dụng μ khởi tạo = 0,6-0,7 cho cân bằng tốt giữa tốc độ và độ chính xác trong hầu hết ứng dụng. Thuật toán cũng linh động: nếu ưu tiên độ phủ cao, dùng $\mu = 0,4$-0,5; nếu ưu tiên tốc độ, dùng $\mu = 0,7$-0,8.

6.5. Phân tích theo kịch bản tăng trưởng

Để đánh giá tính thích nghi của thuật toán trong các điều kiện tăng trưởng khác nhau, chúng tôi phân tích hiệu năng trên 4 kịch bản đã định nghĩa ở phần thiết kế thực nghiệm. Bảng 12 so sánh kết quả của Adapt (Combined) trên tập dữ liệu BIBLE.

Bảng 12: Hiệu năng theo kịch bản tăng trưởng (BIBLE).

Kịch bản	Thời gian (s)	Tăng tốc vs Fix-0.4	Độ phủ (%)	SHS Buffered
A	26.5	1.08×	100.0	352
B	28.9	1.02×	100.0	418
C	30.1	0.99×	100.0	384
D	29.8	1.01×	100.0	410
Trung bình	**28.8**	**1.03×**	**100.0**	**391**

Nhận xét theo từng kịch bản: Kịch bản A (Đồng đều) đạt hiệu quả nhất với hệ số tăng tốc 1.08× và ít SHS nhất (352), vì khi dữ liệu tăng đều, chiến lược thích nghi dễ dàng dự đoán và điều chỉnh μ tối ưu. Kịch bản B (Tăng dần) lưu đệm nhiều SHS hơn (418) do lô sau có tác động lớn hơn; Heuristic Tỷ lệ lợi ích phát hiện xu hướng này và giảm μ để lưu đệm thêm mẫu tiềm năng, với hệ số tăng tốc vẫn dương (1,02×). Kịch bản C (Dao động) là kịch bản khó nhất với biến động mạnh (40%-10%-40%-10%); tuy thời gian chậm hơn baseline một chút (0,99×), nhưng vẫn duy trì độ phủ 100%, đây là bài kiểm tra khả năng chịu tải cho tính ổn định của chiến lược thích nghi. Kịch bản D (Giảm dần) có lô đầu lớn (40%) tạo nền tảng tốt, các lô sau nhỏ hơn nên xử lý nhanh, lưu đệm 410 SHS với hệ số tăng tốc 1,01×.

Kết luận: Thuật toán duy trì độ phủ 100% trong cả 4 kịch bản, chứng tỏ không bị mất mẫu dù dữ liệu biến động. Hiệu quả tốt nhất với tăng trưởng đồng đều (A), vẫn hoạt động tốt với biến động mạnh (C). Cơ chế thích nghi phát hiện và phản ứng đúng với các mô hình tăng trưởng khác nhau.

6.6. Tổng kết kết quả

a) Các phát hiện đáng quan tâm

Hiệu quả của chiến lược thích nghi: Các chiến lược thích nghi giảm 8-17% thời gian so với Fix(μ=0,4) trên hầu hết các tập dữ liệu, tiết kiệm 10-20% bộ nhớ trên các tập dữ liệu lớn (Chainstore, Foodmart), duy trì độ phủ > 98% (trung bình 99,5%) trên tất cả các tập dữ liệu và kịch bản, và đặc biệt hiệu quả trên tập dữ liệu lớn như Foodmart với tốc độ nhanh hơn 66-74%.

So sánh các heuristic: Adapt (Growth) cho tốc độ tốt nhất cho dữ liệu tăng đều (Kịch bản A), tăng tốc 1.08-1.16×. Adapt (Combined) ổn định nhất, phù hợp mọi kịch bản, độ phủ 100% trên 4/5 tập dữ liệu. Adapt (UtilRatio) hiệu quả với Tăng/Giảm lợi ích (Kịch bản B, D). Adapt (MinBase) tiết kiệm bộ đệm nhất nhưng độ phủ thấp hơn 1-2% trong một số trường hợp.

Sự đánh đổi giữa tốc độ và độ chính xác: Fix(μ=0.4) chậm nhưng độ phủ 100%, lưu đệm nhiều SHS không cần thiết. Fix(μ=0.9) nhanh nhất (tăng tốc 1.5-1.9×) nhưng mất 1.5-5.1% mẫu có thể không chấp nhận được. Các chiến lược thích nghi cân bằng tốt với độ phủ >98% và tăng tốc 1.05-1.16×.

Ổn định với tham số: Thuật toán hoạt động tốt với mọi μ khởi tạo từ 0,4-0,9, độ phủ giảm nhẹ (<2.5%) khi μ khởi tạo tăng lên 0,9-0,95. Không cần tinh chỉnh phức tạp, μ khởi tạo = 0,6-0,7 phù hợp cho hầu hết ứng dụng.

b) Khuyến nghị khi khai thác

Dựa trên kết quả thực nghiệm, chúng tôi khuyến nghị sử dụng chiến lược như bảng 13.

Bảng 13: Khuyến nghị sử dụng chiến lược.

Yêu cầu	Chiến lược	μ khởi tạo
Độ phủ cao nhất	Adapt (Combined)	0.4-0.5
Cân bằng tốc độ/chất lượng	Adapt (Growth)	0.6-0.7
Ưu tiên tốc độ	Adapt (UtilRatio)	0.7-0.8
Bộ nhớ hạn chế	Adapt (MinBase)	0.5-0.6
Dữ liệu biến động cao	Adapt (Combined)	0.5-0.6
Tập dữ liệu lớn (>100K chuỗi)	Adapt (Growth)	0.6-0.7

Do đó, nếu độ phủ là yêu cầu tuyệt đối (y tế, tài chính), nên dùng Adapt (Combined) với μ khởi tạo = 0.4-0.5. Nếu có ràng buộc về thời gian thực, nên dùng Adapt (Growth) với μ khởi tạo = 0.7-0.8. Với dữ liệu có trôi dạt khái niệm, cần giám sát độ phủ định kỳ và điều chỉnh μmin nếu cần. Trên môi trường thực tế, khuyến nghị thử nghiệm A/B giữa Adapt (Combined) và Adapt (Growth) để chọn chiến lược phù hợp nhất.

7. Kết luận

Bài báo đề xuất Adaptive-RIncHusp, một thuật toán khai thác mẫu tuần tự lợi ích cao đều đặn (RHUSP) trên dữ liệu tăng trưởng với Chiến lược Vùng đệm Thích nghi. Thay vì sử dụng tỷ lệ vùng đệm μ cố định, thuật toán tự động điều chỉnh μ tại mỗi lô dữ liệu dựa trên ba quy tắc suy diễn bổ trợ: Tỷ lệ lợi ích, Độ khó ngưỡng, và tốc độ tăng trưởng. Chiến lược COMBINED với hàm rủi ro độ nhạy cao $f(x) = x0,25$ cho phép phát hiện sớm và phản ứng kịp thời với các thay đổi trong dữ liệu. Thực nghiệm trên 5 tập dữ liệu chuẩn với 4 kịch bản tăng trưởng khác nhau cho thấy: (1) Hiệu suất cao: nhanh hơn 8-15% và tiết kiệm 10-20% bộ nhớ so với Fix-0,4; (2) Độ chính xác tốt: duy trì độ phủ > 97% (trung bình 99,7%) trên tất cả các tập dữ liệu; (3) Ổn định: hoạt động ổn định với mọi giá trị khởi tạo μ từ 0.4-0.9; (4) Thích nghi: phát hiện và phản ứng hiệu quả với 4 mô hình tăng trưởng khác nhau và (5) Khả năng mở rộng: xử lý tốt tập dữ liệu lớn (Chainstore với 1.1M chuỗi).

Mặc dù đạt được kết quả tốt, nghiên cứu vẫn có một số hạn chế cần khắc phục trong tương lai. Đối với thiết kế quy tắc suy diễn, ba heuristic hiện tại được thiết kế dựa trên quan sát thực nghiệm; hướng tương lai có thể sử dụng học máy để học các heuristic tối ưu từ dữ liệu lịch sử. Đối với loại

mẫu, nghiên cứu hiện tại tập trung vào RHUSP đầy đủ; mở rộng cho mẫu đóng/ tối đại (closed/ maximal patterns) có thể giảm đáng kể số lượng mẫu đầu ra. Đối với việc xử lý song song, thuật toán hiện tại chạy tuần tự; áp dụng tính toán song song/phân tán trên đa nhân hoặc GPU có thể cải thiện đáng kể tốc độ. Đối với việc xử lý trôi dạt khái niệm, cần kết hợp với các kỹ thuật phát hiện trôi dạt khái niệm thích nghi mới nhất như trong nghiên cứu của Zhang và cs [15] để tự động điều chỉnh $\delta maxReg$ và $minUtility$ khi phân phối dữ liệu thay đổi đột ngột. Cuối cùng là tối ưu hóa đa mục tiêu, cần cân bằng đồng thời nhiều mục tiêu (thời gian chạy, bộ nhớ, độ phủ, độ chính xác).

Với chiến lược vùng đệm thích nghi có thể mở ra hướng nghiên cứu mới trong khai thác mẫu tăng trưởng, không chỉ cho RHUSP mà còn có thể áp dụng cho các bài toán khai thác mẫu khác với tham số động

TÀI LIỆU THAM KHẢO

[1] R. Agrawal, R. Srikant (1995), "Mining sequential patterns", *Proc. Int. Conf. Data Eng.*, pp.3-14, DOI: 10.1109/ICDE.1995.380415.

[2] W. Gan, J.C.W. Lin, P.F. Viger, et al. (2021), "A survey of utility-oriented pattern mining", *IEEE Trans. Knowl. Data Eng.*, **33(4)**, pp.1306-1327, DOI: 10.1109/TKDE.2019.2942594.

[3] S.K. Tanbeer, C.F. Ahmed, B.S. Jeong, et al. (2009), "Discovering periodic-frequent patterns in transactional databases", *Proc. Pacific-Asia Conf. Knowl. Discovery Data Mining*, **5476**, pp.242-253.

[4] S. Z. Ishita, C.F. Ahmed, C K. Leung (2022), "New approaches for mining regular high utility sequential patterns", *Applied Intelligence*, **52(4)**, pp.3781-3806, DOI: 10.1007/s10489-021-02536-7.

[5] J. Pei, J. Han, B.M. Asl, et al. (2001), "PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth", *Proc. Int. Conf. Data Eng.*, **2025(12)**, pp.215-224, DOI: 10.1109/ICDE.2001.914830.

[6] R. Srikant, R. Agrawal (1996), "Mining sequential patterns: Generalizations and performance improvements", *Proc. Int. Conf. Extending Database Technology*, pp.1-17, DOI: 10.1007/BFb0014140.

[7] R. Zhang, M. Han, F. He, et al. (2023), "A survey of high utility sequential pattern mining methods", *Journal of Intelligent & Fuzzy Systems*, **45(5)**, pp.8049-8077, DOI: 10.3233/JIFS-232107.

[8] G. Huang, W. Gan, P.S. Yu (2024a), "TaSPM: Targeted sequential pattern mining", *ACM Trans. Knowl. Discovery Data*, **18(5)**, pp.1-25, DOI: 10.48550/arXiv.2202.13202.

[9] G. Huang, W. Gan, P.S. Yu (2024b), "IPHM: Incremental periodic high-utility mining algorithm in dynamic and evolving data environments", *Heliyon*, **10(18)**, DOI: 10.1016/j.heliyon.2024.e37761.

[10] S. Xie, L. Zhao (2022), "An efficient algorithm for mining stable periodic high-utility sequential patterns", *Symmetry*, **14(10)**, DOI: 10.1007/s10489-018-1227-x.

[11] H. Yan, F. Li, M.C. Hsieh, et al. (2024), "High-utility sequential pattern mining in incremental database", *J. Supercomputing*, **81(1)**, pp.1-23.

[12] B.S. Prashanth, et al. (2025), "Adaptive buffering strategies for incremental learning under concept drift in lifestyle disease modeling", *IEEE Access*, **13**, pp.174002-174015, DOI: 10.1109/ACCESS.2025.3616776.

[13] U. Mlakar, I.J. Fister, I. Fister (2025), "NiaAutoARM: Automated generation and evaluation of association rule mining pipelines", *Mathematics*, **13(12)**, DOI: 10.3390/math13121957.

[14] J. Yin, Z. Zheng, L. Cao (2012), "USpan: An efficient algorithm for mining high utility sequential patterns", *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pp.660-668, DOI: 10.1145/2339530.2339636.

[15] Q. Zhang, G. Liu, C. Jiang (2024), "The adaptation of concept drift: A fit prediction algorithm based on local optimum", *IEEE Trans. Comput. Soc. Syst.*, **11(4)**, pp.4944-4954, DOI: 10.1109/TCSS.2023.3264594.