IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version December 23, 2024.

Digital Object Identifier 10.1109/ACCESS.2024.Doi Number

Thiết kế và phân tích các thuật toán tìm kiếm Top K tập mục định kỳ có độ hữu ích cao từ cơ sở dữ liệu không chắc chắn

Phạm Nguyễn Khôi Nguyên
Khoa Công nghệ Thông tin, Trường Đại học Tôn Đức Thắng, TP. Hồ Chí Minh, Việt Nam

Tác giả liên hệ: Phạm Nguyễn Khôi Nguyên (e-mail: 52000695@student.tdtu.edu.vn).

Nghiên cứu này được hướng dẫn bởi TS. Nguyễn Chí Thiện, Khoa Công Nghệ Thông Tin, Trường Đại học Tôn Đức Thắng.

ABSTRACT Mining high-utility itemsets (HUIs) have been a pivotal area in data mining, especially for applications requiring the discovery of patterns with high profitability. Traditional approaches have primarily focused on static datasets with deterministic values, often neglecting periodic patterns, data uncertainty, and scenarios involving negative utilities. This research proposes a novel framework for mining Top-k periodic high-utility itemsets from uncertain database to addressing these limitations. Our approach integrates concepts from periodic pattern mining, utility itemset mining, and uncertain data modeling to effectively discover meaningful patterns under real-world conditions. Additionally, the framework is designed to handle both positive and negative utilities, ensuring its applicability in datasets with mixed utility dynamics often observed in real-world scenarios. We improve upon existing algorithms, including PHMN, PHMN+, and TKN, by introducing advanced pruning strategies and dynamic threshold adjustment mechanisms tailored for periodic and uncertain data. Experimental evaluations on large-scale datasets demonstrate that the proposed method outperforms state-of-the-art algorithms in terms of execution time, memory efficiency, and scalability. Moreover, our approach demonstrates strong adaptability and efficiency across various dataset characteristics. The experimental results validate its effectiveness and scalability, providing a solid foundation for further advancements in high-utility pattern mining under periodic and uncertain conditions.

INDEX TERMS Data mining, Negative utilities, Periodic pattern, Threshold raising strategies, Top-k high-utility itemset, Uncertain database

I. INTRODUCTION

Khai phá tập mục có độ tiện ích cao (HUIM) đã trở thành một trong lĩnh vực cốt lõi trong khai phá dữ liệu nhờ vào những ứng dụng rộng rãi trong đa dạng các lĩnh vực thực tế. Các thuật toán HUIM truyền thống như HUI-Miner [1] và FHM [2] đã đạt được những thành tựu đáng kể thông qua việc giới thiệu các chiến lược cắt tỉa hiệu quả và cấu trúc danh sách tiện ích (PNUList). Tuy nhiên, các phương pháp này vẫn chủ yếu tập trung vào dữ liệu tĩnh với các giá trị xác định, bỏ qua hai khía cạnh quan trọng vốn phổ biến trong dữ liệu thực tế: tính định kỳ (periodicity) và tính không chắc chắn (uncertainty).

Tính định kỳ phản ánh các mẫu xuất hiện theo chu kỳ đều đặn theo thời gian, một đặc tính thiết yếu trong các bài toán như phân tích xu hướng bán hàng, giám sát lưu lượng mạng và chẩn đoán y khoa,..Trong khi đó, dữ liệu không chắc chắn phát sinh từ các tình huống mà thông tin được biểu diễn kèm với các giá trị xác suất, chẳng hạn như các dữ liệu dự báo, phân tích,… từ đó giúp đưa ra những quyết định đúng đắn hơn. Sự kết hợp giữa tính định kỳ và tính không chắc chắn trong khai phá tập mục có độ tiện ích cao đã đặt ra những thách thức tính toán đáng kể, khiến các thuật toán HUIM truyền thống không thể đáp ứng hiệu quả.

Để giải quyết những hạn chế này, nghiên cứu này đề xuất một khung thuật toán mới nhằm khai phá Top-k tập mục định kỳ có độ tiện ích cao từ cơ sở dữ liệu không chắc chắn. Phương pháp này tích hợp các khái niệm cốt lõi từ khai phá mẫu định kỳ, khai phá tập mục có độ tiện ích cao, và mô hình dữ liệu không chắc chắn để hiệu quả phát hiện các mẫu không chỉ có giá trị cao mà còn mang tính định kỳ trong môi trường dữ liệu không chắc chắn. Các thuật toán được đề xuất giới thiệu các chiến lược cắt tỉa tiên tiến, kết hợp nhiều ràng buộc chặc chẽ giúp sàn lọc ra những ứng viên tiềm năng một cách nhanh chóng và tiết kiệm chi phí,

đồng thời sử dụng thêm những chiến lược mới dựa trên Dynamic Upper Bound Pruning, đề xuất thêm về một cấu trúc dữ liệu được kế thừa từ danh sách tiện ích PNUList trước đó,..Những cải tiến này giúp cải thiện đáng kể về thời gian thực thi, hiệu suất bộ nhớ và khả năng mở rộng trên nhiều bộ dữ liệu khác nhau.

Và cuối cùng các thuật toán được thử nghiệm trên các bộ dữ liệu lớn và đa dạng phản ánh từng bài toán trong thực tế, đồng thời chứng minh rằng phương pháp của chúng tôi mang lại hiệu quả tốt hơn hơn so với các giải thuật truyền thống và là giải pháp cho thách thức mà tính định kỳ và không chắc chắn đặt ra trong khai phá dữ liệu tiện ích cao.

II. RELATED WORK

A. HUI-Miner

HUI-Miner, được đề xuất bởi Liu và Qu [2], đánh dấu một cột mốc quan trọng trong lĩnh vực khai phá tập mục có độ tiện ích cao (HUIM). Thuật toán này sử dụng cấu trúc danh sách tiện ích (Utility-List) để lưu trữ thông tin tiện ích của các tập mục, giúp giảm đáng kể số lần quét cơ sở dữ liệu và tối ưu hóa hiệu suất xử lý. Thay vì tạo ra và kiểm tra các tập hợp con tiềm năng không cần thiết, HUI-Miner trực tiếp tính toán giá trị tiện ích dựa trên danh sách tiện ích, từ đó cải thiện đáng kể hiệu quả tính toán. Các biến thể như FHM [3] và HUP-Miner mở rộng phương pháp này bằng cách giới thiệu các chiến lược cắt tỉa tiên tiến như Estimated Utility Co-occurrence Pruning (EUCP) và RU-Prune. Những chiến lược này giúp giảm thiểu số lượng tổ hợp ứng viên phải xử lý trong quá trình khai phá, từ đó cải thiện tốc độ tính toán và tiết kiệm bộ nhớ.

Dù mang lại nhiều cải tiến, các phương pháp này vẫn chủ yếu hoạt động trên dữ liệu xác định, thiếu khả năng xử lý dữ liệu không chắc chắn và tiện ích âm—những thách thức phổ biến trong dữ liệu thực tế. Tuy nhiên, những đóng góp từ HUI-Miner và các biến thể của nó đã tạo nền tảng quan trọng cho các thuật toán HUIM sau này, mở đường cho việc nghiên cứu và phát triển các phương pháp khai phá dữ liệu phức tạp hơn.

B. EFIM (Efficient High-Utility Itemset Mining)

EFIM (Efficient High-Utility Itemset Mining), được giới thiệu bởi Souleymane Zida và cộng sự [1], đã mang lại một bước tiến lớn trong việc cải thiện hiệu suất khai phá tập mục có độ tiện ích cao. Thuật toán này áp dụng các giới hạn trên chặt chẽ hơn, bao gồm Sub-tree Utility và Local Utility, nhằm tối ưu hóa việc cắt tỉa không gian tìm kiếm. EFIM sử dụng hai kỹ thuật quan trọng: High-Utility Database Projection (HDP) và High-Utility Transaction Merging. Hai kỹ thuật này giúp giảm đáng kể số lần quét cơ sở dữ liệu và xử lý hiệu quả cả dữ liệu dày đặc (dense data) lẫn dữ liệu thưa (sparse data). Thay vì chỉ dựa vào danh sách tiện ích như các thuật toán trước đó, EFIM sử dụng cấu trúc utility-bin, cho phép tính toán tiện ích với độ phức tạp tuyến tính trên từng giao dịch.

Những cải tiến này giúp EFIM vượt qua các hạn chế của HUI-Miner và FHM, đặc biệt là về hiệu suất bộ nhớ sử dụng, tu từ đó trở thành nền tảng cho các thuật toán khai phá tiện ích cao hiện đại. Tuy nhiên, nhược điểm của EFIM là về thời gian xử lý tính toán khi không ứng dụng thêm bất kì cấu trúc dữ liệu nào để lưu trữ thông tin, đồng thời còn thiếu khả năng xử lý tính định kỳ (periodicity) và dữ liệu không chắc chắn (uncertainty) — hai yếu tố đóng vai trò quan trọng trong nhiều ứng dụng thực tế.

III. PRELIMINARIES AND PROBLEM DEFINITION

Trong phần này chúng tôi trình bày các khái niệm cơ bản, chiến lược cắt tỉa, và cấu trúc được sử dụng trong các thuật toán đề xuất.

Cho một tập mục chứa các item đơn lẻ: $I = \{i_1, i_2, \dots, i_k\}$ ứng với mỗi item sẽ có giá trị tiện ích (utility) tương ứng $u(i_j)$ $(1 \le j \le k)$ và giá trị $u(i_j)$ hoàn toàn có thể lớn hơn 0 (positive utility) hoặc nhỏ hơn 0 (negative utility) – ***Bảng 1*** Một giao dịch định lượng không chắc chắn là một tập hợp các item cùng với giá trị xác suất (p) và số lượng (q) tương ứng với nó. $T_j = \{(i_1, p_{i_1}, q_{i_1}), (i_2, p_{i_2}, q_{i_2}), \dots\}$

Một tập hợp các giao dịch định lượng không chắc chắc $\{T_1, T_2, \dots, T_{|D|}\}$ được gọi là một cơ sở dữ liệu định lượng không chắc chắn (với |D| là kích thước của cơ sở dữ liệu), và nếu như có ít nhất một item $i \in I$ có giá trị tiện ích âm, thì ta gọi là cơ sở dữ liệu định lượng không chắc chắn với tiện ích dương và âm, được kí hiệu là N-Database. – ***Bảng 2***

Items	a	b	c	d	e	f	g
Utility	3	7	-5	10	-3	-2	-1

Bảng 1: Danh mục các item và giá trị tiện ích

Tid	Items
1	(a, 0.7, 4), (b, 0.6, 2), (c, 0.2, 1), (d, 0.6, 2)
2	(a, 0.8, 1), (c, 0.7, 1), (d, 0.6, 1), (g, 0.6, 3)
3	(a, 0.6, 1), (c, 0.9, 1), (f, 0.5, 1)
4	(a, 0.7, 1), (f, 0.7, 4), (g, 0.6, 2)
5	(a, 0.8, 1), (g, 0.6, 2)
6	(b, 0.8, 3), (c, 0.6, 3), (d, 0.5, 3), (e, 0.7, 1)
7	(c, 0.7, 6), (e, 0.6, 4)
8	(e, 0.8, 1), (f, 0.7, 3)

Bảng 2: Cơ sở dữ liệu định lượng không chắc chắn

A. Periodic pattern

Định nghĩa 1 (Periods): Cho một tập mục X, ta có thể tìm được một danh sách các giao dịch chứa X, được kí hiệu:
$T(X) = \{tid \mid X \in T_{tid}\} = \{tid_1, tid_2, \dots tid_m\}$;
$PS(X) = \{tid_{i+1} - tid_i \mid i = 0, 1, \dots, m\}$;
với $tid_0 = 0$ và $tid_{m+1} = |D|$. Và các giá trị min, max, average periodic của itemset X lần lượt là:

- minPer(X) = min{PS(X)}
- maxPer(X) = max{PS(X)}
- avgPer(X) = |D| / |PS(X)|

Ví dụ: Xét tập mục X = {a}, ta có tập mục Tid của các giao dịch chứa X là: T(X) = {1, 2, 3, 4, 5}, PS(X) = {1, 1, 1, 1, 1, 3}, minPer(X) = min{PS(X)} = 1, maxPer(X) = max{PS(X)} = 3, avgPer(X) = |D| / |PS(X)| = 8/5 = 1.6

Định nghĩa 2 (Periodic Pattern): Cho các giá trị minPer, maxPer, minAvg, maxAvg, là các ngưỡng giá trị do người dùng định nghĩa. Với một itemset X, nếu các điều kiện sau được thoả mãn: minPer(X) ≥ minPer; maxPer(X) ≤ maxPer; minAvg ≤ avgPer(X) ≤ maxAvg. Thì tập mục X là được xem là một mẫu định kỳ (Periodic Pattern)
Ví dụ: Xét tập mục X = {a}, ta có minPer(X) = 1, maxPer(X) = 3, avgPer(X) = 1.6. Giả sử các ngưỡng periodic người dùng định nghĩa như sau: minPer = 1, maxPer = 5, minAvg = 1, maxAvg = 3. Ta có thể nói tập mục X là một mẫu định kỳ vì cả 3 ràng buộc đều thoả mãn

Chiến lược 1 (maxPer Pruning): Cho itemset X, nếu như maxPer(X) > maxPer, thì itemset X và các phần mở rộng của nó đều không phải là một mẫu định kỳ (periodic pattern).
Chiến lược 2 (avgPer Pruning): Cho itemset X, nếu như avgPer(X) > maxAvg, thì itemset X và các phần mở rộng của nó đều không phải là một mẫu định kỳ (periodic pattern).

B. High-utility pattern

Định nghĩa 3 (Utility of itemset): Cho một cơ sở dữ liệu định lượng không chắc chắn D, một itemset X và một giao dịch không chắc chắn T, nếu như X ∈ T, giá trị tiện ích của X trong giao dịch T được tính bởi công thức:
- $U(X, T) = \sum_{i \subseteq X} u(i) \times q_i$

Và giá trị tiện ích của X trong cơ sở dữ liệu sẽ được tính bởi công thức:
- $U(X) = \sum_{X \subseteq T_j \wedge T_j \subseteq D} U(X, T_j)$

Ví dụ: Xét tập mục X = {a}, giá trị tiện ích của X trong các giao dịch: $U(X, T_1) = 12$, $U(X, T_2) = 3$, $U(X, T_3) = 3$, $U(X, T_4) = 3$, $U(X, T_5) = 3$. Giá trị tiện ích của X trong cơ sở dữ liệu là: U(X) = 12 + 3 + 3 + 3 + 3 = 24.

Định nghĩa 4 (Positive Utility of itemset): Cho một cơ sở dữ liệu định lượng không chắc chắn D, và một itemset và một giao dịch không chắc chắn T, nếu như X ∈ T, giá trị tiện ích dương (positive utility) của X trong giao dịch T được tính bởi công thức:
- $PU(X, T) = \sum_{i \subseteq X \wedge u(i) \geq 0} (u(i) \times q_i)$

Và giá trị tiện ích dương của X trong cơ sở dữ liệu sẽ được tính bởi công thức:
- $PU(X) = \sum_{X \subseteq T_j \wedge T_j \subseteq D} PU(X, T_j)$

Ví dụ: Xét tập mục X = {a}, giá trị tiện ích dương của X trong các giao dịch: $PU(X, T_1) = 12$, $PU(X, T_2) = 3$, $PU(X, T_3) = 3$, $PU(X, T_4) = 3$, $PU(X, T_5) = 3$.
Giá trị tiện ích dương của X trong cơ sở dữ liệu là:
PU(X) = 12 + 3 + 3 + 3 + 3 = 24.

Định nghĩa 5 (Negative Utility of itemset): Cho một cơ sở dữ liệu định lượng không chắc chắn D, một itemset X và một giao dịch không chắc chắn T, nếu như X ∈ T, giá trị tiện ích âm (negative utility) của X trong giao dịch T được tính bởi công thức:
- $NU(X, T) = \sum_{i \subseteq X \wedge u(i) < 0} (u(i) \times q_i)$

Và giá trị tiện ích âm của X trong cơ sở dữ liệu sẽ được tính bởi công thức:
- $NU(X) = \sum_{X \subseteq T_j \wedge T_j \subseteq D} NU(X, T_j)$

Ví dụ: Xét tập mục X = {c}, giá trị tiện ích âm của X trong các giao dịch lần lượt là: $NU(X, T_1) = -5$, $NU(X, T_2) = -5$, $NU(X, T_3) = -5$, $NU(X, T_6) = -15$, $NU(X, T_7) = -30$.
Giá trị tiện ích âm của X trong cơ sở dữ liệu là:
NU(X) = (-5) + (-5) + (-5) + (-15) + (-30) = -60.

Định nghĩa 6 (Expected support): Cho một cơ sở dữ liệu định lượng không chắc chắn D, một itemset X và một giao dịch không chắc chắn T, nếu như X ∈ T, giá trị xác suất của X trong giao dịch T sẽ được tính bởi công thức:
- $P(X, T) = \prod_{i \subseteq X} p_{iT}$

Và độ hỗ trợ kì vọng (expected support) của X trong cơ sở dữ liệu sẽ được tính bởi công thức:
- $P(X) = \sum_{X \subseteq T_j \wedge T_j \subseteq D} P(X, T_j)$

Ví dụ: Xét tập mục X = {a}, giá trị xác suất của X trong các giao dịch lần lượt là: $P(X, T_1) = 0.7$, $P(X, T_2) = 0.8$, $P(X, T_3) = 0.6$, $P(X, T_4) = 0.7$, $P(X, T_5) = 0.8$.
Giá trị hỗ trợ kì vọng của X là: expSup(X) = 0.7 + 0.8 + 0.6 + 0.7 + 0.8 = 3.6.

Chiến lược 3 (Probability pruning): Cho một cơ sở dữ liệu định lượng không chắc chắn D và itemset X, ngưỡng xác suất tối thiểu minProb, nếu như P(X) < minProb × |D|, thì tập mục X và các phần mở rộng của nó đều không phải là một mẫu kì vọng cao.

Định nghĩa 7 (Expected High Utility Pattern): Cho một cơ sở dữ liệu định lượng không chắc chắn D, một itemset X, một ngưỡng xác suất tối thiểu minProb, ngưỡng tiện ích tối thiểu minUtil. Nếu như itemset X thoả mãn cả 2 điều kiện:
(1) P(X) ≥ minProb × |D|, (2) U(X) ≥ minUtil
Thì tập mục X được xem là một mẫu tiện ích – kì vọng cao.
Ví dụ: Xét tập mục X = {a}, ta có U(X) = 24 và P(X) = 3.6. Giả sử các ngưỡng minUtil = 20, và giá trị minProb = 0.1. Ta có U(X) > minUtil và P(X) > 0.8 (= 0.1 * 8). Ta nói X là một tập mục tiện ích kì vọng cao.

Định nghĩa 8 (Transaction Weighted Utility): Cho một cơ sở dữ liệu định lượng không chắc chắn D, một itemset X và một giao dịch không chắc chắn T, giá trị tiện ích dương của giao dịch T được tính bởi công thức:
- $PU(T) = \sum_{i \subseteq T \wedge u(i) > 0} (u(i) \times q_i)$

Và nếu như X ∈ T, giá trị tiện tích có trọng số của X trong giao dịch T được tính bởi công thức:
- $TWU(X) = \sum_{X \subseteq T_j \wedge T_j \subseteq D} PU(T_j)$

Ví dụ: Xét tập mục X = {a}, giá trị tiện ích dương của các giao dịch chứa X lần lượt là: PU(T_1) = 46, PU(T_2) = 13, PU(T_3) = 3, PU(T_4) = 3, PU(T_5) = 3.
Giá trị trị TWU(X) = 46 + 13 + 3 + 3 + 3 = 68.

Chiến lược 4 (Transaction Weighted Utility pruning): Xét những item đơn lẻ i (i ∈ I), nếu như TWU({i}) < minUtil thì bất kì tập mục nào chứa i, đều không phải là một mẫu tiện ích cao.
Chiến lược 5 (Estimated Utility Co-occurrence Pruning in N-database): Đối với hai item đơn lẻ i_i, i_j chúng ta có thể lưu giá trị TWU của chúng vào một ma trận tam giác được gọi là EUCP. Đối với một tập mục X, nếu nó chứa cả hai item i_i, i_j và TWU($\{i_i, i_j\}$) < minUtil thì tập mục X và các phần mở rộng của nó đều không phải là một mẫu tiện ích cao.

Định nghĩa 9 (Order of Items): Cho một tập mục chứa các item đơn lẻ: $I = \{i_1, i_2, \dots, i_k\}$, ta định nghĩa thứ tự ưu tiên giữa hai item đơn lẻ i_a và i_b ($i_a, i_b \in I$) như sau:
(1) Nếu u(i_a) > 0 và u(i_b) ≤ 0, ta kết luận $i_a < i_b$.
(2) Nếu u(i_a) × u(i_b) > 0 và TWU($\{i_a\}$) < TWU($\{i_b\}$), ta kết luận $i_a < i_b$.
(3) Nếu u(i_a) × u(i_b) > 0 và TWU($\{i_a\}$) = TWU($\{i_b\}$) và u(i_a) > u(i_b), ta kết luận $i_a < i_b$.
Ta cũng có thể mở rộng thứ tự ưu tiên giữa hai tập mục A, B
Nếu $i_a < i_b$ ($\forall i_a \in A$ và $\forall i_b \in B$), ta có thể kết luận $A < B$
Ví dụ: Xét hai item a và d, thứ tự ưu tiên của hai items có thể được sắp xếp như sau: a < d (u(a) > 0 và u(d) < 0).
Xét hai item a và b, thứ tự ưu tiên của hai items có thể được sắp xếp như sau: a < b.
(u(a) × u(b) > 0 và TWU(a) = 68 < TWU(d) = 110).
Từ đó, ta có thể sắp xếp thứ tự ưu tiên (Bảng 1) như sau:
a < b < d < f < g < e < c.

Định nghĩa 10 (Remaining Utility): Cho một cơ sở dữ liệu định lượng không chắc chắn D, một itemset X và một giao dịch không chắc chắn T, nếu như X ∈ T, giá trị tiện còn lại (remaining utility) của X trong giao dịch T được tính bởi công thức:
- RU(X, T) = $\sum_{i \in T \wedge i > x \, \forall x \in X \wedge u(i) > 0} (u(\mathrm{i}) * q_i)$

Và giá trị tiện ích còn lại của X trong cơ sở dữ liệu sẽ được tính bởi công thức:
- RU(X) = $\sum_{X \subseteq T_j \wedge T_j \subseteq D} RU(X, T_j)$

Ví dụ: Xét tập mục X = {a}, ta có giá trị tiện ích còn lại của X trong các giao dịch lần lượt là:
RU(X, T_1) = 34 , RU(X, T_2) = 10, RU(X, T_3) = 0,
RU(X, T_4) = 0, RU(X, T_5) = 0.
Giá trị tiện ích còn lại của X trong cơ sở dữ liệu là:
RU(X) = 34 + 10 = 44.

Chiến lược 6 (Remaining Utility pruning): Cho một cơ sở dữ liệu định lượng không chắc chắn D, một itemset X, ngưỡng tiện ích tối thiểu minUtil, nếu như RU(X) + U(X) < minUtil, thì tập mục X và các phần mở rộng của nó đều không phải là một mẫu tiện ích cao.

Định nghĩa 11 (PNU-List): Cấu trúc danh mục tiện ích dương và âm của một tập mục X là một tuple gồm hai phần tử (X, utilitty-list), utilitity-list đại diện cho một mảng và mỗi phần tử trong mảng là một 5-tuple (tid, pu, nu, ru, prob) với:
- tid là id của giao dịch chứa nó.
- pu là giá trị tiện ích dương.
- nu là giá trị tiện ích âm.
- ru là giá trị tiện ích còn lại.
- prob là giá trị xác xuất trong giao dịch đó.

Ví dụ: Xét tập mục X = {a}, ta có cấu trúc PNU-List(X)

Tid	pu	nu	ru	prob
2	6	0	13	0.9
3	6	0	33	0.5
5	24	0	9	1.0

Định nghĩa 12 (MList): Cấu trúc MList là một cấu trúc được cải tiến từ cấu trúc PNU-List là một 6-tuple được tạo bởi các thành phần (X, X', X'-PNUList, Prefix-X', pu, ru) với:
- X là tập mục được tạo thành.
- X' là một tập con của X và đại diện cho tập mục thực sự được lưu trữ trong Mlist.
- X'-PNUList là cấu trúc PNUList của tập mục X'.
- Prefix-X' là tập mục tiền tố của X'.
- pu là giá trị tiện ích dương.
- ru là giá trị tiện ích còn lại.

Định nghĩa 13 (Dynamic Upper Bound): Cho hai tập mục X và Y có cùng bậc trong một cây liệt kê (Enumeration tree) và X là tập mục sắp được mở rộng và $Y > X$ (có độ ưu tiên thấp hơn). Khi đó ta có thể viết lại: $X = \tilde{X} \cup x$, $Y = \tilde{X} \cup y$ với x, y là hai item có trong tập I.
Nếu tập mục Y được tạo bởi PNUList (định nghĩa 11):
- DU(Y, X) = RU(Y) + PU(Y) + PU(x),

Nếu tập mục Y được tạo bởi MList (định nghĩa 12):
- DU(Y, X) = RU(Y-Mlist) + PU(Y-Mlist) + PU(x).

Chiến lược 7 (Dynamic Upper Bound pruning): Cho hai tập mục X và Y có cùng bậc trong một cây liệt kê (Enumeration tree), X là tập mục sắp được mở rộng và $Y > X$ (có độ ưu tiên thấp hơn).
- Nếu DU(Y, X) < minUtil thì ta không cần tạo ra ứng viên $X \cup Y$.
- Nếu Y được tạo bởi cấu trúc PNUList và Y có tập mục prefix là $\tilde{Y}$ thì ta sẽ khởi tạo cấu trúc Mlist như sau: MList($X \cup Y$, Y, Y-PNUList, $\tilde{Y}$-PNUList)
- Nếu Y được tạo bởi cấu trúc MList(Y, Y', Y'-PNUList, $\tilde{Y}'$-PNUList) thì ta sẽ khởi tạo cấu trúc MList như sau: MList($X \cup Y$, Y', Y'-PNUList, $\tilde{Y}'$-PNUList)

Định nghĩa 14 (Local Utility): Cho một tập mục α và một item z bất kì có mức độ ưu tiên thấp hơn α (z > α). Giá trị tiện ích cục bộ (local utility) của α và z được tính theo công thức sau: lu(α, z) = $\sum_{\alpha \cup \{z\} \subseteq \mathrm{T} \wedge \mathrm{T} \subseteq D} U(\alpha, T) + RU(\alpha, T)$

Ví dụ: Xét tập mục $\alpha = \emptyset$ ta có thể tính giá trị tiện ích cục bộ của các item khi kết hợp với α lu(α, a) = 68
Xét tập mục $\alpha = \{a\}$, lu(α, b) = 46, lu(α, c) = 62.

Định nghĩa 15 (Subtree Utility): Cho một tập mục α và một item z bất kì có mức độ ưu tiên thấp hơn α (z > α). Giá trị tiện ích cây con (subtree utility) của α và z là su(α, z) được tính theo công thức sau:
$\sum_{\alpha \cup \{z\} \subseteq T \wedge T \subseteq D} [U(\alpha, T) + U(z, T) + RU(\alpha \cup \{z\}, T)]$
Ví dụ: Xét tập mục $\alpha = \emptyset$ ta có thể tính giá trị tiện ích cây con của các item khi kết hợp với α su(α, a) = 68.
Xét tập mục $\alpha = \{a\}$, su(α, b) = 97, su(α, c) = -42,
su(α, d) = 75, su(α, e) = -18

C. Top-k high utility itemset mining

Định nghĩa 16 (Top-k High-Utility Itemset): Cho tập mục I chứa tất cả các item, một cơ sở dữ liệu định lượng không chắc chắn D và một hằng số được người dùng định nghĩa k. Top-k High-Utility Itemset là một tập hợp chứa k tập mục có giá trị tiện ích cao nhất có thể được tìm thấy.
Ví dụ: Giả sử các ngưỡng đầu vào mà người dùng định nghĩa như sau: k = 4, minProb = 0.1, minPer = 1,
maxPer = 5, minAvg = 1, maxAvg = 3
Với k = 4, ta sẽ tìm được các tập mục có giá trị tiện ích cao nhất lần lượt là: {b, d} - 85, {d} - 63, {b} – 35, {c, d} – 35

Định nghĩa 17 (Leaf Itemset Utility Structure): Cho một tập mục I' chứa các item tiềm năng (TWU({i}) ≥ minUtil, $\forall i \in$ I') và đã được sắp xếp dựa trên Định nghĩa 9 – Order of items. LIUS là một ma trận tam giác được sử dụng để lưu trữ giá trị tiện ích (utility) của từng tập 2-item có trong I'.
Ví dụ: Giả sử giá trị tiện ích tối thiểu minUtil = 24, thứ tự của các item đã được sắp xếp a < b < d < c
Ta có cấu trúc LIUS như sau:

Item	a	b	d
b	26		
d	45	85	
c	3	15	35

Chiến lược 8 (Positive Real Item Utility strategy): Trong lần quét cơ sở dữ liệu lần đầu tiên, các giá trị tiện ích dương của từng item trong tập I sẽ được tính toán dựa trên Định nghĩa 4. Sau đó chiến lược PRIU sẽ cập nhật giá trị minUtil bằng giá trị tiện ích lớn thứ k-th trong danh sách đã được sắp xếp.

Chiến lược 9 (Positive LIU-Exact strategy): Chiến lược này sử dụng các giá trị tiện ích (utility) được lưu trữ trong cấu trúc LIUS được nhắc đến trong Định nghĩa 17. Sau đó PLIU-Exact sẽ cập nhật giá trị minUtil bằng giá trị tiện ích lớn thứ k-th trong danh sách đã được sắp xếp.

Định nghĩa 18 (Tid-set of an Itemset): Tập hợp Tid-set của một tập mục X được kí hiệu là T(X) và được định nghĩa là tập hợp các id của những giao dịch chứa X.

- T(X) = $\{id \mid X \subseteq T_{id} \wedge T_{id} \subseteq D\}$

Ví dụ: Xét tập mục X = {a}, ta có tập mục Tid của các giao dịch chứa X là: T(X) = {1, 2, 3, 4, 5}

Định nghĩa 19 (Coverage): Cho i và j là hai item đơn lẻ ($i, j \in I'$ = { i | TWU({i}) ≥ minUtil, $i \in$ I}) và j được xem là một item nằm trong vùng (coverage) của i nếu: G({j}) ⊆ G({i}).
Ví dụ: Ta có T({a}) = {1, 2, 3, 4, 5}, T({g}) = {2, 4, 5}
Vì T({a}) ⊆ T({g}), ta có thể nói item g nằm trong vùng (coverage) của item a.

Định nghĩa 20 (The Coverage Utilility): Cho một item p, và tập hợp C(p) chứa những item nằm trong vùng (coverage) của nó dựa trên Định nghĩa 19, giá trị tiện ích của vùng item i sẽ được tính bởi công thức

- U(C(p)) = $\sum_{1 \le i \le r} U(\{p, p_i\}) - (1 - r) \times U(p)$

Ví dụ: Item g nằm trong vùng của Item a, ta có công thức tính vùng của item như sau:
U(C(a)) = U({a, g}) – (1 – r) × U(g) = 46

Chiến lược 10 (COVL Strategy): Cho tập mục I chứa tất cả những item đơn lẻ, ta có thể tính các giá trị vùng tiện ích (coverage utility) của từng item bên trong tập mục. Sau đó COVL sẽ cập nhật giá trị minUtil bằng giá trị tiện ích lớn thứ k-th trong danh sách đã được sắp xếp.

IV. PROPOSED ALGORITHMS

Xét bài toán cơ sở: Dựa vào tập mục các item và giá trị tiện ích (Bảng 1), cơ sở dữ liệu định lượng không chắc chắn (Bảng 2), các giá trị ngưỡng đầu vào được người dùng định nghĩa như sau: k = 4, minProb = 0.1, minPer = 1, maxPer = 5, minAvg = 1, maxAvg = 3. Hãy tìm k tập mục định kì có giá trị tiện ích cao nhất (PHUIs).

A. Top-k PHMN and PHMN+ Algorithms

Cả hai thuật toán Top-k PHMN và PHMN+ đều sẽ bao gồm hai giai đoạn chính, giai đoạn đầu tiên của thuật toán là chuẩn bị (Preparation Procedure). Ý tưởng của thuật toán này sử dụng các chiến lượt để khởi tạo giá trị ngưỡng tiện ích tối thiểu, tiếp theo đó là giảm không gian tìm kiếm bằng cách tiền xử lý cơ sở dữ liệu bằng các chiến lược cắt tỉa những item không tìm năng dựa vào Periodic (Chiến lược 1 và 2), TWU (Chiến lược 3), ExpSup (Chiến lược 4). Điều này sẽ giúp giảm thiểu các tổ hợp ứng viên không tiềm năng được sinh ra trong giai đoạn khai thác. Và cuối cùng sắp xếp lại cơ sở dữ liệu dữ liệu (Định nghĩa 9) để mang lại hiệu quả tối ưu trong việc tính toán.

Giai đoạn thứ hai trong thuật toán là giai đoạn tìm kiếm và mở rộng (Searching Procedure). Trong giai đoạn này, thuật toán sẽ duyệt qua các tập mục tiềm năng, mở rộng chúng thành các tập lớn hơn và kiểm tra xem có đáp ứng các ràng buộc hay không. Bằng cách sử dụng các chiến lược cắt tỉa (RU, ExpSup, maxPer, avgPer), chỉ mở rộng với các tập mục có tiềm năng giúp tối ưu hóa tốc độ và hiệu quả khai thác. Đặc biệt hơn, thuật toán PHMN+ được cải tiến với việc bổ sung

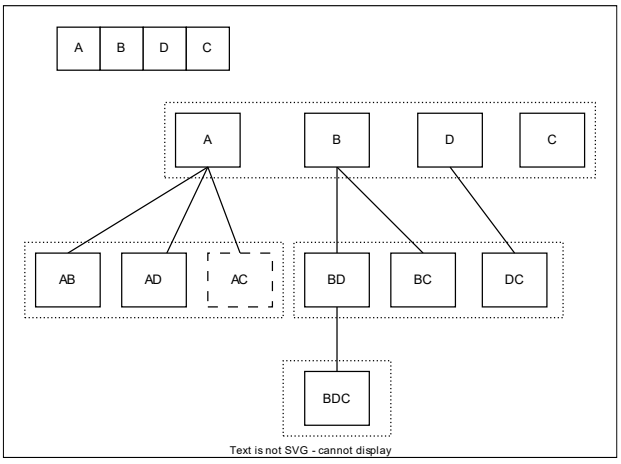
chiến lược DU Pruning và sử dụng cấu trúc MList (được kế thừa từ PNUList), giúp tối ưu quá trình khai thác thông qua việc hạn chế việc tạo các tập mục không tiềm năng.

Algorithm 1: Preparation Procedure

Input:
item_list: List of items
database: List of transactions
k: Maximum number of PHUIs
minPer, maxPer, minAvg, maxAvg: periodic thresholds
min_prob: Minimum probability threshold
Output:
Top-k highest-utility item-sets

1. Initialize a priority queue topk_queue to store top-k results
2. Apply PLIU strategy to initialize current_min_util
3. Remove items from item_list and database by Periodic, TWU, Expected conditions.
4. Sort the remaining items in item_list and database by TWU and utility
5. Build necessary data structures: Transaction dictionary, PNU-Lists, EUCST & CUDM dictionaries
6. Apply PLIUE strategy to increase current_min_util
7. Apply COVL strategy to increase current_min_util
8. PHUI_Searching_Plus(item_list, database, ...) if is_plus then PHUI_Searching(item_list, database, ...)
9. Return topk_queue containing the k-highest utility item-sets

Đầu tiên giá trị minUtil sẽ được khởi tạo bằng chiến lượt PLIU (minUtil = 24), tiếp theo đó các giá trị periodic, expSup, và TWU sẽ lần lượt được tính toán cho từng item. Và nếu một trong các điều kiện không được thoả mãn thì có thể loại bỏ item không tiềm năng khỏi item_list ban đầu và cơ sở dữ liệu. Sau đó, các item còn lại sẽ được sắp xếp theo thứ tự ưu tiên (dựa trên Định nghĩa 9) để sẵn sàng cho việc khởi tạo các cấu trúc dữ liệu như EUSCT, CUDM,... Lúc này các item không đạt yêu cầu sẽ bị loại bỏ lần lượt là: {e, f, g} và những item còn lại {a, b, c, d} sẽ được sắp xếp theo thứ tự: a < b < d < c. (Định nghĩa 9). Cuối cùng áp dụng hai chiến lược PLIUE và COVL để tiếp tục nâng ngưỡng giá trị cho minUtil (minUtil = 35) và gọi hàm Searcing Procedure để mở rộng không gian tìm kiếm (quan sát thuật toán 1).



Hình 3: Các tập mục được khởi tạo từ item_list

Trước khi trình bày thuật toán Searching Procedure giúp khám phá không gian tìm kiếm cũng như các tập mục tiềm năng, chúng tôi xin giới thiệu về một thuật toán giúp khởi tạo cấu trúc PNUList của một tập mục Pxy thông qua việc kết hợp từ hai tập mở rộng của nó là Px và Py.

Ý tưởng chính của thuật toán là sẽ tận dụng các giá trị tiện ích được lưu trữ trong PNUList của Px và Py, thuật toán sẽ tìm kiếm các giao dịch chung của Px và Py để kết hợp thay vì phải tính toán từ đầu bằng cách duyệt qua toàn bộ cơ sở dữ liệu, đồng thời dựa vào các ngưỡng xác suất và ngưỡng tiện ích để cắt tỉa sớm những tập mục không tiềm năng, giúp tối ưu hóa hiệu suất và giảm chi phí tính toán.

Algorithm 2: PNUList Construct

Input:
P: PNUList of prefix P.
Px: PNUList of the extension of P with an item x.
Py: PNUList of the extension of P with an item y.
min_util: minimum utility threshold.
user_prob_threshold: minimum probability threshold.
Output:
The PNUList for the combined item set Px ∪ Py

```
1. if Px or Py is empty then
2.     return null
3. end
4. Pxy.PUL ← ∅
5. set probability ← sum(Px.pro), utility ← sum(Px.pu)
+ sum(Px.ru)
6. foreach tuple ex in Px.PUL do
7.     find ey in Py.PUL such that ex.tid = ey.tid
8.     if ey exists then
9.         if P.PUL ≠ ∅ then
10.            find e in P.PUL such that e.tid = ex.tid
11.            exy ← ⟨ ex.tid, (ex.pro × ey.pro) / e.pro,
12.                   ex.pu + ey.pu - e.pu, ex.nu + ey.nu -
e.nu, ey.ru ⟩
13.            add exy to Pxy.PUL
14.        else
15.            exy ← ⟨ ex.tid, ex.pro × ey.pro,
16.                   ex.pu + ey.pu, ex.nu + ey.nu, ey.ru ⟩
```

```
17.            add exy to Pxy.PUL
18.     else
19.        probability ← probability - ex.pro
20.        utility ← utility - ex.pu - ex.ru
21.        if probability < user_prob_threshold or utility <
min_util then
22.            return null
23.        end
24. end
25. return Pxy.PUL
```

Ví dụ: Ta có cấu trúc PNUList của hai items b và d lần lượt như sau:

{b}				
Tid	pu	nu	ru	prob
1	14	0	20	0.6
6	21	0	30	0.8

{d}				
Tid	pu	nu	ru	prob
1	20	0	0	0.6
2	10	0	0	0.6
6	30	0	0	0.5

Khi áp dụng Algorithm 2, ta có thể construct PNUList của itemset {b, d}:

{b, d}				
Tid	pu	nu	ru	prob
1	34	0	20	0.36
6	51	0	30	0.4

Thuật toán Searching được sử dụng tìm kiếm các tập mục có lợi ích cao định kỳ (PHUIs) bằng cách duyệt qua danh sách ứng viên tiềm năng (PNUList) đã được sàn lọc và khởi tạo ở bước trước đó (Preparation Procedure). Thuật toán bắt đầu với vòng lặp for, duyệt qua từng ứng viên, những tập mục đạt yêu cầu sẽ được đưa vào hàng đợi ưu tiên (topk-queue) để duy trì k tập mục có tiện ích cao nhất, đồng thời ngưỡng tiện ích tối thiểu (minUtil) cũng sẽ được cập nhật mỗi khi hàng đợi đầy. Giá trị này sẽ được cập nhật bằng giá trị nhỏ nhất trong hàng đợi, mục đích để chỉ xét duyệt những tập mục có giá trị tiện ích lớn hơn những phần tử trong hàng đợi. Sau đó, thuật toán tiếp tục kiểm tra các điều kiện cắt tỉa như tiện ích còn lại (RU), điều kiện định kỳ (Periodic) và điều kiện xác suất (expSup). Các tập mục đủ điều kiện được kết hợp để tạo danh sách ứng viên mới và quá trình này được lặp lại đệ quy nhằm đảm bảo chỉ những tập mục hứa hẹn nhất được mở rộng và xử lý, tối ưu hóa hiệu quả khai thác. Ngược lại, nếu tập mục không thoả 1 trong 3 điều kiện trên sẽ bị cắt tỉa ngay tại đó.

Algorithm 3: PHMN Searching Procedure

Input:
P: a PNUList representing the prefix.
lists: a candidates PNUList lists.
current_min_util: current minimum utility threshold.
k: Maximum number of PHUIs
minPer, maxPer, minAvg, maxAvg: periodic thresholds.
min_prob: Minimum probability threshold
topkqueue: A heap contains k-highest utility itemsets.
Output: None

```
1. foreach itemset X in lists do
2.   if X satisfies utility, expSup, periodic then:
3.      push X into topk_queue
4.      update current_min_util if topk_queue is full
5.   end
6.   if X satisfies conditions of RU, maxPer, avgPer then:
7.      new_lists ← ∅
8.      foreach itemset Y in lists, and Y > X do
9.        if twu(x, y) ≥ current then:
10.           Z ← construct(P, X, Y)
11.           new_lists.add(Z)
12.         end
13.       end
14.       call Search(X, X-list, new_lists)
15.   end
16. end
```

Những item tiềm năng còn lại và được sắp xếp theo mức độ ưu tiên lần lượt là: a < b < d < c, và các cấu trúc PNUList tương ứng cũng đã được khởi tạo cho chúng, lúc này giá trị minUtil = 35. Thuật toán sẽ bắt đầu bước tiếp theo với phần tử đầu tiên là a, và kết hợp với những item phía sau nó lần lượt là (b, c, d) để tạo thành một danh sách mới: {a, b} < {a, d} < {a, c}. Thuật toán sẽ tiếp tục tìm kiếm đệ quy theo chiều sâu dựa trên các tập mục mới này. Lần đệ quy đầu tiên, thuật toán cũng sẽ xuất phát từ {a, b} và lần lượt kết hợp với {a, c}, và {a, d}.Tuy nhiên giá trị U({a, b}) = 26 < minUtil = 35 nên tập mục {a, b} không phải là một trong các tập mục tiềm năng, tiếp theo các giá trị tiện ích còn lại (RU), tổng xác suất (expSup), và periodic sẽ được tính toán để xem xét {a, b} có thể kết hợp mở rộng không gian tìm kiếm hay không, tuy nhiên giá trị maxPer({a, b}) = 7 > maxPer = 5 dẫn đến không thoả điều kiện về periodic (Chiến lược 1) nên {a, b} sẽ bị cắt nhánh ngay tại đây. Thuật toán tiếp tục với tập mục {a, d}, giá trị maxPer({a, d}) = 6 > maxPer = 5 cũng không thoả điều kiện về chiến lược 1, và chỉ còn một tập mục duy nhất cuối cùng là {a, c} nên không thể tiếp tục mở rộng, và thuật toán sẽ kết thúc việc gọi hàm đệ quy vì không còn ứng viên nào được sinh ra. Vòng lặp sẽ tiếp tục xét đến item b, và xem xét các điều kiện để kết hợp với c và d, nếu có ứng viên được sinh ra thì thuật toán sẽ tiếp tục đệ quy theo chiều sâu để tìm kiếm, và ngược lại thì thuật toán sẽ kết thúc. Tất cả những ứng viên được sinh ra được thể hiện trong ***Hình 3***
Và các tập mục định kỳ tiện ích cao (PHUIs) của bài toán được tìm thấy sẽ là: {b, d} - 85, {d} - 63, {b} – 35, {c, d} – 35

Algorithm 4: PHMN+ Searching Procedure

Input:
P: a PNUList representing the prefix.
lists: a candidates PNUList lists.
current_min_util: current minimum utility threshold.
k: Maximum number of PHUIs
minPer, maxPer, minAvg, maxAvg: periodic thresholds
min_prob: Minimum probability threshold
topkqueue: A heap contains k-highest utility itemsets.
Output: None

```
1. foreach candidate list X in lists do
2.   if X satisfies of utility, expSup, periodic then:
3.       push X into topk_queue
4.       update current_min_util if topk_queue is full
5.     end
6.   if X satisfies of RU, expSup, maxPer, avgPer then:
7.     initialize new_lists ← ∅
8.     foreach itemset Y in lists where Y > X do
9.       if twu(x, y) ≥ current then:
10.          if DU(Y, X) ≥ current_min_util then
11.            if Y is an MList then
12.              Z = PNUListConstruct(P, X, Y)
13.              add Z to new_lists
14.            else
15.              Z = PNUListConstruct(P, X, Y)
16.              add Z to new_lists
17.          else
18.            if Y is an MList then
19.              construct Z using GenMList1
20.              add Z to new_lists
21.            else
22.              construct Z using GenMList2
23.              add Z to new_lists
24.            end
25.          end
26.      end
27.      phui_searching_procedure_plus(X, new_lists, ...)
28. end
```

Thuật toán PHMN+ Searching Procedure cũng có cách hoạt động tương tự như thuật toán PHMN, tuy nhiên sự khác biệt ở đây là trước khi kết hợp 2 tập mục X và Y, giá trị DU(Y, X) sẽ được tính (dựa trên định nghĩa 13) và chiến lược 7 sẽ được áp dụng để giảm thiểu quá trình sinh ra một tập mục mới không tiềm năng. Cụ thể hơn, ở ví dụ đầu tiên (quan sát hình 3) khi chiến lược DU Pruning áp dụng thì tập mục {a, c} sẽ không được khởi tạo,vì giá trị DU(c, a) = 24 < min_util = 35, chính vì thế thuật toán sẽ bỏ qua và không khởi tạo tập mục này.

Phân tích độ phức tạp của thuật toán searching:

Bước 1: Kích thước đầu vào: |lists| = n => T(n)

Bước 2: Thao tác cơ bản (basic operation): "twu(x, y) ≥ current" ở dòng thứ 9 của thuật toán 3 và 4.

Bước 3: Trường hợp tệ nhất (worst case): Không có item nào bị loại bỏ sau thuật toán 1 và các chiến lượt cắt tỉa ở thuật toán 3 và 4 không cắt được nhánh nào.

Bước 4: Đếm thao tác cơ bản:

- i = 1 (phần tử đầu tiên trong lists), ta sẽ có (n – 1) tập mục ứng viên được sinh ra. Đây sẽ là tập mục đầu vào của lần gọi đệ quy tiếp theo (dòng thứ 14 của thuật toán 3, và dòng thứ 27 của thuật toán 4). Ta sẽ gọi hàm đệ quy có độ phức tạp T(n – 1).

  Ví dụ: Giả sử lists = {a, b, c, d, e}. Khi i = 1, có nghĩa đang xét tại phần tử đầu tiên (a), các ứng viên được sinh ra lần lược sẽ là: {ab, ac, ad, ae}. Tập mục này có kích thước n – 1 và là đầu vào (input) của lần gọi đệ quy tiếp theo.
- i = 2, tương ứng sẽ có n – 2 tập mục ứng viên, ta sẽ gọi hàm đệ quy có độ phức tạp T(n – 2).
- i = 3, tương ứng sẽ có n – 3 tập mục ứng viên, ta sẽ gọi hàm đệ quy có độ phức tạp T(n – 3).
- ….
- i = n – 1, tương ứng sẽ có 1 tập mục ứng viên, ta sẽ gọi hàm đệ quy có độ phức tạp T(1)
- i = n (phần tử cuối cùng trong list) lúc này không còn ứng viên nào được sinh ra, thuật toán sẽ kết thúc.

  => T(n) = T(n – 1)

Đếm số lần thực hiện thao tác cơ bản:

T(1) = 1 = $(2^1 - 1)$

T(2) = 3 + 1 = $(2^2 - 1)$ + T(1)

T(3) = 7 + 4 = $(2^3 - 1)$ + T(2)

….

T(n – 1) = $(2^{n-1} - 1)$ + T(n – 2)

Thực hiện cộng hai vế, ta được:

=> T(n – 1) = $(2^1 - 1)$ + $(2^2 - 1)$ + … + $(2^{n-1} - 1)$

$= 2^{n-1} + \ldots + 2^2 + 2^1 + (n-1) \times (-1)$

$= 2^{n-1} + \ldots + 2^1 + 2^0 - n$

$= 2^n - n - 1$

=> T(n) = 2^n – n – 1

Bước 5: Kết luận: T(n) ∈ $\theta(2^n)$

B. TKN Algorithm

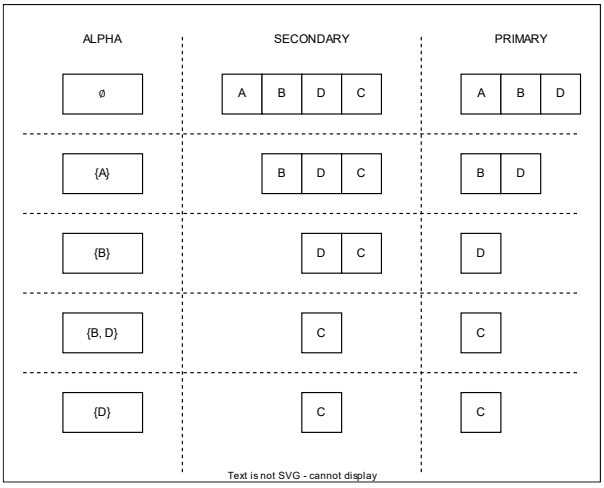
Thuật toán TKN được đề xuất bởi Mohamed Ashraf và đồng đội, ý tưởng chính của thuật toán là xây dựng một tập mục chính, kết hợp với các item tiềm năng được xác định bởi các giá trị tiện ích cục bộ và tiện ích cây con (local & subtree utlity). Đầu tiên, thuật toán sử dụng chiến lược nâng ngưỡng để cập nhật ngưỡng tiện ích minUtil. Sau đó, các mục không có tiềm năng sẽ được loại bỏ dựa trên các chiến lược như TWU, expSup, periodic. Tiếp theo thuật toán khởi tạo một tập hợp rỗng được gọi là ***alpha*** và tính toán giá trị lợi ích cây con (subtree utility) và lợi ích cục bộ (local utility) cho các item còn lại, danh sách các item được chia thành hai nhóm: ***secondary*** (mục phụ) và ***primary*** (mục chính) để phục vụ việc sinh các ứng viên tiềm năng. Cuối cùng, thuật toán gọi hàm tìm kiếm đệ quy để mở rộng các tập hợp mục, tập trung vào các mục chính và đảm bảo chỉ giữ lại các tập hợp có giá trị tiện ích cao nhất.

Algorithm 5: TKN

Input:
item_list: List of items.
database: List of transactions.
k: Maximum number of PHUIs.
minPer, maxPer, minAvg, maxAvg: periodic thresholds.
min_prob: Minimum probability threshold.
Output:
Top-k highest-utility item-sets.

1. Initialize an empty set alpha α and priority queue topk_queue size k
2. Set current_min_util using PLIU strategy and calculate user_prob_threshold
3. Create twu, probability, utility arrays for all items in item_list
4. Remove unqualified items based on expSup, periodic, TWU strategies
5. Sort remaining item_list and database by TWU and utility
6. Build necessary data structures: Transaction dictionary and EUCST values
7. Apply PLIUE, COVL strategy to increase current_min_util
8. Initialize secondary = {item | item ∈ item_list ∧ lu(α, item) ≥ minUtil}
9. Initialize primary = {item | item ∈ secondary ∧ su(α, item) ≥ minUtil}
10. Call tkn_searching(alpha, primary, secondary, current_min_util, ...)
11. return topkqueue

Tương tự với thuật toán PHMN, đầu tiên giá trị minUtil sẽ được khởi tạo bằng chiến lượt PLIU (minUtil = 24), tiếp theo đó các giá trị periodic, expSup, và TWU sẽ lần lượt được tính toán cho từng item. Những item còn lại thoả mãn cả ba điều kiện và được sắp xếp theo thứ tự ưu tiên lần lượt là: a < b < d < c. Sau đó tập mục α sẽ được khởi tạo rỗng, primary và secondary sẽ được khởi tạo từ những item còn lại. Dựa trên các kết quả về lu(α, i) của các item lần lượt: lu(α, a) = 68, lu(α, b) = 97, lu(α, d) = 110, lu(α, c) = 113 (đều lớn hơn minUtil = 24), ta có secondary = {a, b, d, c}, và các giá trị và su(α, i) lần lượt là: su(α, a) = 68, su(α, b) = 85, su(α, d) = 60, lu(α, c) = -60 (nhỏ hơn minUtil) ta có primary = {a, b, d}. Cuối cùng ngưỡng minUtil sẽ tiếp tục được nâng bằng các chiến lược PLIUE và COVL để khám phá những tập có giá trị tiềm năng cao hơn, và gọi hàm searching procedure.



Hình 4: Ví dụ khi áp dụng thuật toán TKN Searching

Algorithm 6: TKN Searching

Input:
α: prefix itemset
primary: list of primary items
secondary: list of secondary items
current_min_util: current minimum utility threshold
minPer, maxPer, minAvg, maxAvg: periodic thresholds.
user_prob_threshold: minimum probability threshold
k: size of the priority queue
topkqueue: priority queue to store top-k item-sets
Output: None

1. foreach item z in primary do:
2. β ← α ∪ {z}
3. compute utility, probability, periodics of β
4. if β satisfies of utility, expSup, periodic then:
5. push β into topk_queue
6. update current_min_util if topk_queue is full
7. end if
8. end if
9. if β satisfies of RU, expSup, maxPer, avgPer then:
10. find the index of z in secondary
11. if β **not** satisfies of utility, periodic then
12. continue
13. end if
14. create projected database for β
15. new_secondary = {item | item ∈ item_list ∧ lu(β, item) ≥ min_util}
16. new_primary = {item | item ∈ new_secondary ∧ su(β, item) ≥ min_util}
17. if new_primary and new_secondary then
18. call efim_global_search(β, new_primary, new_secondary, ...)
19. end if
20. end foreach

Thuật toán sẽ bắt đầu với vòng lặp for, duyệt qua các phần tử trong mảng primary, đầu tiên giá trị β được khởi tạo bởi α và lần lượt các item tiềm năng, tiếp theo thuật toán sẽ kiểm tra các điều kiện về expSup, utility, và periodic, nếu tất cả điều kiện được thoả mãn, thì β sẽ được

xem là một tập mục định kỳ có tiện ích cao và được thêm vào heap.

Tiếp theo đó ta sẽ tiến hành tạo ra những mảng secondary và primary mới dựa trên β và các item đơn lẻ còn lại trong secondary ban đầu (khi β = {a}, các items còn lại là: {b, c, d} quan sát hàng thứ hai ***Hình 4***). Các giá trị lu(β, b) = 77, lu(β, d) = 65, lu(β, c) = 88, lần lượt được tính toán và đều có giá trị lớn hơn ngưỡng minUtil, mảng secondary mới được tạo thành gồm các phần tử: {b, d, c}. Sau đó, những phần tử này sẽ tiếp tục kết hợp với β để tính toán các giá trị subtree utility: su(β, b) = 91, su(β, d) = 72, su(β, c) = -55,, mảng primary mới được tạo thành gồm các phần tử: {b, d} (quan sát hàng thứ hai ***Hình 4***).

Cuối cùng sau khi các mảng mới đã được tạo thành, lời gọi đệ quy TKN_Searching được thực hiện, lúc này β sẽ đóng vai trò tương tự như tập mục α lúc đầu, sẽ kết hợp với các phần tử trong mảng primary mới, sinh ra một β mới, và các mảng primary, secondary mới,…và cứ tiếp tục như vậy cho đến khi không còn mảng ứng viên nào được sinh ra thì thuật toán sẽ kết thúc.

Và với k = 4, các tập mục định kì có tiện ích cao được tìm thấy lần lượt là:

Phân tích độ phức tạp của thuật toán searching:

Bước 1: Kích thước đầu vào: |primary| = n => T(n), hoặc |secondary| = m => T(m), hoặc phụ thuộc vào cả 2 => T(n, m)

Bước 2: Thao tác cơ bản (basic operation): "lu(β, item) ≥ min_util" ở dòng thứ 15 của thuật toán 6.

Bước 3: Trường hợp tệ nhất (worst case): Không có item nào bị loại bỏ sau thuật toán 5, kích thước của primary và secondary bằng với kích thước của item list ban đầu và các chiến lược cắt tỉa ứng viên ở thuật toán 6 không cắt được nhánh nào.

Bước 4: Đếm thao tác cơ bản:

- i = 1, tập mục primary mới sẽ có kích thước là n – 1. Vì trong trường hợp tệ nhất tập primary mới sẽ có kích thước bằng với kích thước của item_set ban đầu trừ 1 (|new_primary| = |old_primary| - 1) do tất cả các item đứng sau item đang xét đều thoả điều kiện về subtree utility, cuối cùng gọi hàm đệ quy có độ phức tạp: T(n – 1)
- i = 2, tập mục primary mới sẽ có kích thước là n – 2, ta sẽ gọi hàm đệ quy có độ phức tạp T(n – 2).
- i = 3, tập mục primary mới sẽ có kích thước là n – 3, ta sẽ gọi hàm đệ quy có độ phức tạp T(n – 3).
- ….
- i = n – 1, tập mục primary mới sẽ có kích thước là 1 (là phần tử cuối cùng), ta sẽ gọi hàm đệ quy có độ phức tạp T(1)
- i = n (phần tử cuối cùng trong primary) tập mục primary = ∅, thuật toán sẽ kết thúc.

  => T(n) = T(n – 1)

Đếm số lần thực hiện thao tác cơ bản:

T(1) = 1 = $(2^1 - 1)$

T(2) = 3 + 1 = $(2^2 - 1)$ + T(1)

T(3) = 7 + 4 = $(2^3 - 1)$ + T(2)

….

T(n – 1) = $(2^{n-1} - 1)$ + T(n – 2)

=> T(n – 1) = $(2^1 - 1)$ + $(2^2 - 1)$ + … + $(2^{n-1} - 1)$

= 2^{n-1} + …. + 2^2 + 2^1 + (n – 1) × (-1)

= 2^{n-1} + …. + 2^1 + 2^0 – n

= 2^n – n – 1

=> T(n) = 2^n – n – 1

Bước 5: Kết luận: T(n) ∈ $\theta(2^n)$

V. PERFORMANCE EVALUATION

Để kiểm tra hiệu quả và tính thực tiễn của các thuật toán mà tôi đề xuất trong việc khai phá các tập mục định kỳ tiện ích cao (PHUIs) từ cơ sở dữ liệu không chắc chắn, chúng tôi sẽ tiến hành thử nghiệm trên ba bộ dữ liệu khác nhau: Mushroom, Connect, và BMS. Các bộ dữ liệu này được chọn nhằm đại diện cho các đặc điểm và cấu trúc dữ liệu đa dạng, qua đó đánh giá được khả năng mở rộng, hiệu suất xử lý, cũng như độ chính xác của từng thuật toán trong những kịch bản thực tế khác nhau.

A. EXPERIMENTAL ENVIRONMENT AND DATASETS

Dataset	T.Transaction	T.Items	Avg.Item	Density
mushrooms	8416	119	23.00	19.33%
connect	67557	129	43.00	33.33%
BMS	59602	497	2.51	0.51%

Hình 5: Thông tin về các bộ dataset thực nghiệm (Nguồn *SPMF*)

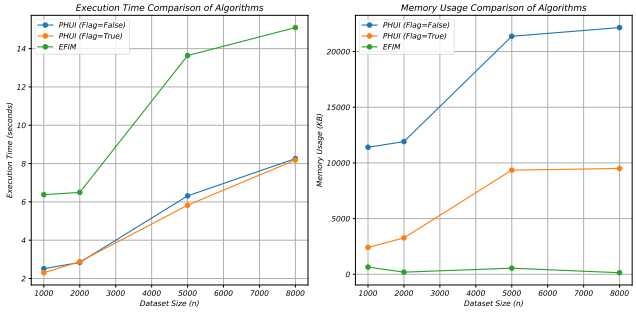
Ba bộ dữ liệu được sử dụng trong thử nghiệm đại diện cho các loại bài toán khác nhau trong thực tế. Mushroom là dữ liệu tương đối dày đặc với mật độ 19.33%, mỗi giao dịch trung bình chứa khoảng 23 items, đại diện cho các bài toán phân loại và nhận dạng mẫu. Thực tế, bộ dữ liệu này được sử dụng để phân loại loài nấm độc và nấm ăn được dựa trên các đặc điểm như màu sắc, hình dáng, và cấu trúc. Connect là bộ dữ liệu có mật độ cao nhất (33,33%), với giao dịch dài (43 mục), thường đại diện cho các bài toán phân tích chiến lược, chẳng hạn như dự đoán kết quả trò chơi dựa trên các nước đi. Bộ dữ liệu này đòi hỏi các thuật toán phải xử lý được dữ liệu lớn, phức tạp, và có nhiều mối liên hệ chặt chẽ giữa các tập mục. Cuối cùng, BMS là dữ liệu thưa nhất với mật độ chỉ 0,51%, đại diện cho các bài toán trong thương mại điện tử, nơi mỗi giao dịch thường chỉ chứa vài sản phẩm nhưng đòi hỏi các thuật toán phải tìm ra các mẫu hữu ích nhất, chẳng hạn như phát hiện các sản phẩm thường được mua chung để tối ưu hóa chiến lược bán hàng. Sự khác biệt trong cấu trúc và đặc điểm của ba bộ dữ liệu này mang lại bối cảnh thực tế đa dạng, giúp đánh giá hiệu suất và tính linh hoạt của các thuật toán.

Trong các bộ dữ liệu gốc, mỗi item đã được mặc định với giá trị tiện ích là 1, và tất nhiên sẽ không có giá trị tiện ích âm. Đồng thời, các bộ dữ liệu này là dữ liệu chắc chắn nên cũng không tồn tại các giá trị xác suất trong từng giao dịch. Để phù hợp với yêu cầu thực nghiệm nghiệm, các giá trị tiện ích (bao gồm cả âm và dương) được tạo ngẫu nhiên bằng cách sử dụng phân phối đều (uniform distribution), với khoảng giá trị từ -100 đến 100 cho từng item tương ứng. Bên cạnh đó, yếu tố xác suất cũng được thêm vào, được tạo ngẫu nhiên bằng phân phối Gauss (Gaussian distribution) với giá trị trung bình (mean) là 0.5 và độ lệch

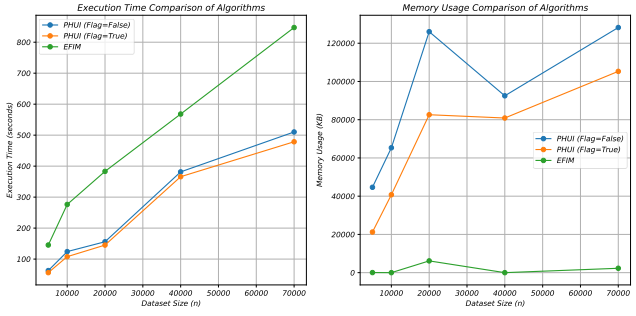
chuẩn (standard deviation) là 0.125 [8]. Việc bổ sung các yếu tố tiện ích và xác suất này giúp mô phỏng dữ liệu thực tế phức tạp hơn, đảm bảo điều kiện thử nghiệm sát với các ứng dụng khai phá dữ liệu không chắc chắn.

B. RUN TIME AND MEMORY USAGE

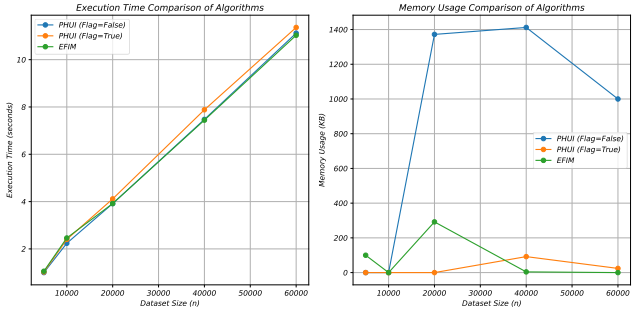
1. ẢNH HƯỞNG CỦA KÍCH THƯỚC DATASET



Hình 5.1: Hiệu suất thời gian và bộ nhớ - Dataset Mushroom



Hình 5.2: Hiệu suất thời gian và bộ nhớ - Dataset Connect

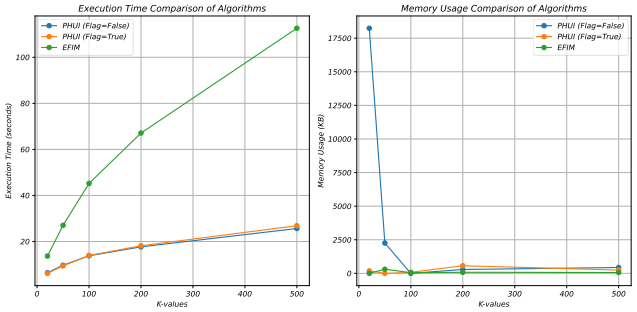


Hình 5.3: Hiệu suất thời gian và bộ nhớ - Dataset BMS

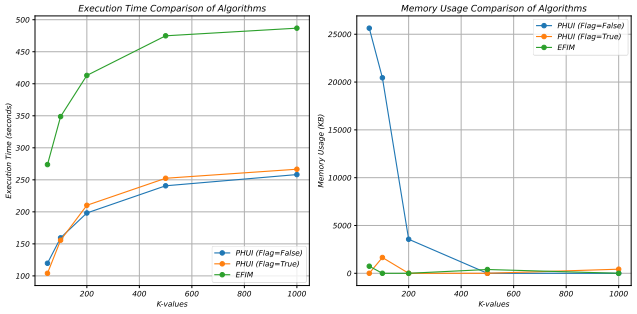
Các kết quả thực nghiệm từ các đồ thị cho thấy sự khác biệt rõ rệt giữa ba thuật toán PHMN, PHMN+, và TKN về thời gian chạy và bộ nhớ sử dụng, phụ thuộc vào đặc điểm thuật toán và cấu trúc dữ liệu. PHMN+, một phiên bản cải tiến từ PHMN, sử dụng chiến lược DU và cấu trúc dữ liệu MList để loại bỏ các tập mục không tiềm năng ngay trước khi khởi tạo, nhờ đó, PHMN+ đạt được thời gian chạy nhanh nhất và tiêu tốn ít bộ nhớ nhất, trong khi đó PHMN thiếu các cải tiến này, dẫn đến việc sinh nhiều tập mục dư thừa hơn, làm tăng cả thời gian chạy lẫn bộ nhớ sử dụng. Và cuối cùng là TKN, mặc dù áp dụng các chiến lược cắt tỉa phức tạp, và tối ưu hơn như local utility và subtree utility, số lượng tập mục ứng viên sinh ra ít hơn hẳn so với hai thuật toán còn lại, nhưng vì không sử dụng cấu trúc lưu trữ đặc biệt, khiến thời gian chạy cao nhất do chi phí tính toán lớn, mặt khác bộ nhớ sử dụng là một ưu điểm cực kì lớn của thuật toán này khi nó không sử dụng bất kì một cấu trúc lưu trữ nào, hoàn toàn đúng với tinh thần của thuật toán EFIM mà nó kế thừa.

Ngoài ra, đặc điểm của các dataset cũng góp phần tạo nên sự khác biệt này. Với Mushroom và Connect, có độ dày đặc cao (19,33% và 33%), không gian tìm kiếm lớn làm nổi bật hiệu quả của PHMN+ trong việc giảm thời gian chạy và bộ nhớ. Trên hai dataset này, sự chênh lệch giữa PHMN+ và hai thuật toán còn lại trở nên rõ rệt hơn khi kích thước bộ dữ liệu ngày càng lớn. Tuy nhiên, với dataset BMS, có độ dày rất thấp (0,51%), không gian tìm kiếm thu hẹp đáng kể khiến thời gian chạy của cả ba thuật toán giảm mạnh. Trong trường hợp này, ngay cả thuật toán TKN vốn được coi là chậm nhất, lại đạt kết quả tương đồng với PHMN và PHMN+ do đặc điểm thưa thớt của dữ liệu giúp giảm bớt chi phí tính toán.

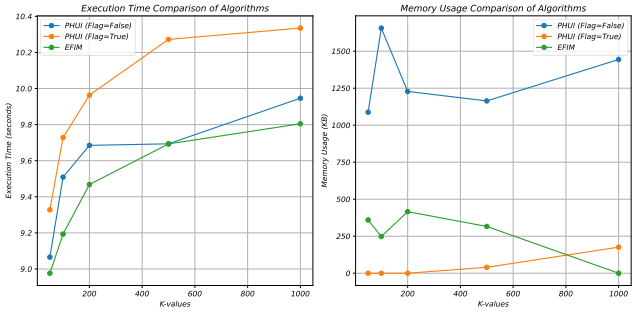
2. ẢNH HƯỞNG CỦA TOP-K



Hình 5.4: Hiệu suất thời gian và bộ nhớ - Dataset Mushroom



Hình 5.5: Hiệu suất thời gian và bộ nhớ - Dataset Mushroom



Hình 5.6: Hiệu suất thời gian và bộ nhớ - Dataset Mushroom

Trên hai tập dữ liệu Mushroom và Connect, có độ dày đặc lần lượt là 20% và 33%, sự gia tăng của k làm tăng

đáng kể thời gian thực thi của tất cả các thuật toán. Trong môi trường này, PHMN và PHMN+ có thời gian thực thi gần như tương đồng, và đáng chú ý là PHMN thậm chí vượt trội hơn PHMN+ khi k tăng cao. Điều này xảy ra do khi k lớn, không gian tìm kiếm được mở rộng, dẫn đến việc sinh ra quá nhiều tập mục tiềm năng. PHMN+, mặc dù được cải tiến với chiến lược DU Pruning để loại bỏ các tập mục không tiềm năng, lại phải kiểm tra chiến lược này cho số lượng tập mục lớn hơn, vô tình làm giảm hiệu năng khi không gian tìm kiếm quá rộng. Trong khi đó, PHMN không áp dụng chiến lược này, nên tiết kiệm được chi phí xử lý khi số lượng tập mục tăng nhanh. Ngược lại, thuật toán TKN tiếp tục thể hiện thời gian thực thi cao nhất do không sử dụng các cấu trúc dữ liệu tối ưu hoặc chiến lược cắt tỉa đặc thù, mặc dù tiêu thụ bộ nhớ ít nhất.

Trên tập dữ liệu BMS, với độ dày đặc chỉ 0.5%, một hiện tượng thú vị được quan sát thấy: thuật toán TKN, vốn thường chậm hơn hẳn trên các tập dữ liệu dày đặc, bất ngờ trở thành thuật toán chạy nhanh nhất. Điều này có thể được giải thích bởi đặc tính cực kỳ thưa của dữ liệu, làm giảm đáng kể số lượng tập mục tiềm năng cần kiểm tra và mở rộng. Khi không gian tìm kiếm bị thu hẹp mạnh mẽ, TKN tận dụng được tính đơn giản trong thiết kế của mình, tập trung vào các chiến lược cắt tỉa như local utility và subtree utility mà không phải xử lý các cấu trúc dữ liệu phức tạp. Ngược lại, PHMN và PHMN+, với chiến lược cắt tỉa mạnh mẽ hơn, lại phải thực hiện nhiều bước kiểm tra bổ sung, đặc biệt là việc tính toán và áp dụng chiến lược DU Pruning trong PHMN+, làm tăng chi phí xử lý trên dữ liệu thưa. Kết quả là TKN không chỉ thu hẹp khoảng cách với PHMN và PHMN+, mà còn vượt qua để trở thành thuật toán chạy nhanh nhất trên tập dữ liệu này.

VI. CONCLUSION

Trong nghiên cứu này, chúng tôi đã đề xuất các thuật toán mới nhằm giải quyết bài toán khai thác Top-K tập mục định kỳ có độ tiện ích cao từ cơ sở dữ liệu không chắc chắn. Các giải pháp này được phát triển dựa trên những thuật toán trước đây, với sự cải tiến về khả năng xử lý và hiệu quả tính toán. Khác với các phương pháp truyền thống, nghiên cứu của chúng tôi không chỉ tập trung vào việc phát hiện các tập mục có tiện ích cao mà còn tích hợp yếu tố định kỳ và tính không chắc chắn — hai yếu tố quan trọng nhưng chưa được chú trọng nhiều trong các nghiên cứu trước. Các thuật toán đề xuất đã chứng minh được khả năng tối ưu hiệu suất xử lý, tiết kiệm bộ nhớ và nâng cao độ chính xác khi khai thác thông tin từ các tập dữ liệu phức tạp, thực tế. Kết quả thực nghiệm cho thấy phương pháp này không chỉ đáp ứng tốt về mặt tính toán mà còn có khả năng mở rộng, tạo nền tảng vững chắc cho việc áp dụng trong các tình huống thực tế đa dạng. Nghiên cứu này không chỉ khắc phục được những hạn chế của các phương pháp truyền thống, mà còn mở ra triển vọng mới cho việc phát triển các giải pháp linh hoạt trong tương lai.

REFERENCES

[1] Souleymane Zida, Philippe Fournier-Viger, Jerry Chun-Wei Lin, Cheng-Wei Wu, and Vincent S. Tseng. EFIM: A Fast and Memory Efficient Algorithm for High-Utility Itemset Mining.
[2] Liu, M., & Qu, J. Mining High Utility Itemsets Without Candidate Generation.
[3] Fournier-Viger, P., Lin, J. C. W., Gomariz, A., & Gueniche, T. FHM: Faster High-Utility Itemset Mining Using Estimated Utility Co-occurrence Pruning.
[4] C. Lee, H. Kim, M. Cho, H. Kim, B. Vo, and J. C. W. Lin, "Incremental top-k high utility pattern mining and analyzing over dynamic database," IEEE Transactions on Knowledge and Data Engineering, vol. 36, no. 4, pp. 589–600, 2024.
[5] H. Cheng, M. Han, N. Zhang, and X. Li, "A survey of incremental high-utility pattern mining based on storage structure," Journal of Intelligent & Fuzzy Systems, vol. 40, no. 7, pp. 15–25, 2023.
[6] M. Ashraf, T. Abdelkader, S. Rady, and T. F. Gharib, "TKN: an efficient approach for discovering top-k high utility itemsets with positive or negative profits," Information Sciences, vol. 621, pp. 25–35, 2022.
[7] S. Kim, H. Kim, M. Cho, H. Kim, and B. Vo, "Efficient approach for mining high-utility patterns on incremental databases with dynamic profits," Knowledge-Based Systems, vol. 252, pp. 123456, 2023.
[8] Z. Li, F. Chen, J. Wu, Z. Liu, and W. Liu, "Efficient weighted probabilistic frequent itemset mining in uncertain databases," Expert Systems, vol. 2020, Mar. 2020, doi: 10.1111/exsy.12551.
[9] K. Singh, R. Kumar, and B. Biswas, "High average-utility itemsets mining: a survey," Applied Intelligence, vol. 65, no. 8, pp. 789–802, 2022.
[10] F. Lai, X. Zhang, G. Chen, and W. Gan, "Mining periodic high-utility itemsets with both positive and negative utilities," Engineering Applications of Artificial Intelligence, vol. 98, pp. 89–103, 2023.
[11] C. Zhang, Z. Du, Y. Yang, W. Gan, and P. S. Yu, "On-shelf utility mining of sequence data," ACM Transactions on Knowledge Discovery from Data, vol. 15, no. 4, pp. 101–112, 2021.
[12] Z. Ren, Y. Xun, J. Cai, and H. Yang, "Stable top-k periodic high-utility patterns mining over multi-sequence," Intelligent Data Analysis, vol. 27, no. 2, pp. 150–162, 2024.
[13] H. Kim, T. Ryu, C. Lee, H. Kim, and B. Vo, "EHMIN: efficient approach of high-utility pattern mining with negative unit profits," Expert Systems with Applications, vol. 210, pp. 115690, 2022.
[14] R. Kumar and K. Singh, "Top-k high utility itemset mining: current status and future directions," The Knowledge Engineering Review, vol. 39, no. 1, pp. 1–15, 2024.
[15] J. M. Luna, R. U. Kiran, P. Fournier-Viger, and S. Ventura, "Efficient mining of top-k high utility itemsets through genetic algorithms," Information Sciences, vol. 556, pp. 59–72, 2023.

[16] L. Chen, W. Gan, Q. Lin, S. Huang, and C. M. Chen, "OHUQI: Mining on-shelf high-utility quantitative itemsets," The Journal of Supercomputing, vol. 78, no. 5, pp. 2145–2162, 2022.
[17] W. Gan, J. C. W. Lin, P. Fournier-Viger, and H. C. Chao, "Beyond frequency: Utility mining with varied item-specific minimum utility," ACM Transactions on Knowledge Discovery from Data, vol. 15, no. 5, pp. 1–16, 2021.
[18] A. Nellutla and N. Srinivasan, "A survey on analysis of data mining algorithms for high utility itemsets," El-Cezeri Journal of Science and Engineering, vol. 9, no. 2, pp. 101–110, 2022.
[19] S. C. Sagare and D. V. Kodavade, "Correlated time-window constrained high-utility itemsets mining with certain and uncertain real-life datasets," Multimedia Tools and Applications, vol. 83, no. 7, pp. 2123–2137, 2024.
[20] P. Fournier-Viger, J. C. W. Lin, and H. C. Chao, "EHMIN: Efficient approach of list-based high-utility pattern mining with negative utilities," IEEE Access, vol. 8, pp. 115673–115690, 2020.
[21] F. Lai, X. Zhang, G. Chen, and W. Gan, "Mining periodic high-utility itemsets with negative utilities," Information Sciences, vol. 550, pp. 78–92, 2023.
[22] R. Kumar and K. Singh, "High utility itemsets mining from transactional databases: A survey," Applied Intelligence, vol. 73, no. 3, pp. 897–912, 2023.
[23] H. Kim, H. Kim, and B. Vo, "Efficient approach of high average utility pattern mining with indexed list-based structure in dynamic environments," Information Sciences, vol. 612, pp. 113–128, 2024.