Ngày phát hành: 21/12/2024, phiên bản 1.0

DESIGN AND ANALYSIS OF ALGORITHMS INTERACTIVELY SEARCHING FOR TOP-K HIGH-UTILITY ITEMSETS FROM UNCERTAIN DATABASES

Tác giả: Huỳnh Hoàng Tiến Đạt (52200023)[1], Nguyễn Thị Huyền Diệu (52200090)[2], Bùi Đông Tấn Đạt (52200001)[3]
[1]Khoa Công nghệ Thông Tin, Đại học Tôn Đức Thắng, Việt Nam
[2]Khoa Công nghệ Thông Tin, Đại học Tôn Đức Thắng, Việt Nam
[3]Khoa Công nghệ Thông Tin, Đại học Tôn Đức Thắng, Việt Nam
Giáo viên hướng dẫn: Ph.D Nguyễn Chí Thiện
Email: nguyenchithien@tdtu.edu.vn

TÓM TẮT Trong bối cảnh thế giới hiện đại hóa và số hóa như hiện nay, việc khai thác dữ liệu hiệu quả để hỗ trợ ra quyết định trong các lĩnh vực như thương mại, y tế, và tài chính đã trở thành một nhu cầu cấp thiết. Trong lĩnh vực phân tích dữ liệu, đặc biệt là khi làm việc với cơ sở dữ liệu không chắc chắn, bài toán tìm kiếm các tập hợp mục có độ hữu dụng cao (High-Utility Itemsets) đóng vai trò quan trọng trong việc tối ưu hóa hiệu suất và giá trị thông tin. Vì vậy, một hệ thống thiết kế và phân tích các thuật toán tìm kiếm tương tác để tìm các tập hợp mục có độ hữu dụng cao nhất (Top-K) từ cơ sở dữ liệu không chắc chắn không chỉ là một thách thức về mặt kỹ thuật mà còn mang lại những ứng dụng thực tiễn sâu rộng.
Báo cáo gồm 04 chương như sau:

Chương 1: Giới thiệu chung – Tổng quan đề tài và các công trình liên quan.
Chương 2: Phân tích bài toán – Định nghĩa bài toán và mục tiêu cụ thể.
Chương 3: Phương pháp thực hiện thuật toán – Triển khai các thuật toán ITUFP, TKU, TUHUFP, và TUHUFP-Growth.
Chương 4: Phân tích độ phức tạp và thực nghiệm – Đánh giá hiệu quả thuật toán và các kết quả thực nghiệm.

I GIỚI THIỆU CHUNG

1.1 Giới thiệu đề tài

Trong kỷ nguyên dữ liệu bùng nổ, khả năng trích xuất thông tin giá trị từ các tập dữ liệu lớn và phức tạp là yếu tố quan trọng đối với doanh nghiệp và các tổ chức. Khai thác tập mục hữu ích cao (HUIM) đã trở thành một kỹ thuật quan trọng trong lĩnh vực khai thác dữ liệu *(Data Mining)*. HUIM hướng đến việc xác định các tập hợp mục *(itemsets)* có giá trị hữu ích lớn, góp phần hỗ trợ ra quyết định và tối ưu hóa các hoạt động thực tiễn.

Tuy nhiên, các phương pháp HUIM truyền thống thường gặp thách thức khi xử lý dữ liệu **không chắc chắn** *(uncertain data)*, do tính chất biến động và thiếu độ tin

cây trong dữ liệu. Điều này đòi hỏi các phương pháp khai thác tiên tiến có khả năng ứng dụng linh hoạt và hiệu quả hơn.

Để giải quyết thách thức này, báo cáo này tập trung vào việc đề xuất các phương pháp mới nhằm khai thác tương tác các tập mục hữu ích hàng đầu *(Top-K High-Utility Itemsets)* từ cơ sở dữ liệu không chắc chắn. Bằng cách áp dụng các kỹ thuật tương tác, người dùng có thể tham gia trực tiếp vào quá trình khai thác, tinh chỉnh kết quả dựa trên nhu cầu và kiến thức chuyên môn cụ thể. Điều này không chỉ tăng cường độ chính xác mà còn đảm bảo tính thực tiễn cao trong việc ứng dụng.

1.2 Các công trình liên quan

1.2.1 Khai thác tương tác các mẫu thường xuyên

Khai thác tương tác các mẫu thường xuyên cho phép người dùng điều chỉnh giá trị ngưỡng minSup một cách linh hoạt để tránh việc bỏ lỡ các mẫu quan trọng. Các thuật toán như CanTree [22], CP – Tree [29], và EFP – Tree [9] có khả năng khai thác tương tác bằng cách giữ lại tất cả các mẫu thường xuyên và không thường xuyên trong cấu trúc cây, cho phép trích xuất mẫu mà không cần quét lại cơ sở dữ liệu. Các thuật toán IWFPWA và IWFPFD [4] cũng hỗ trợ khai thác tăng cường mà không cần cắt tỉa các item không thường xuyên. HUPMS [3] khai thác mẫu có giá trị cao từ luồng dữ liệu mà không cần xây dựng lại cây. LINE [18] cho phép khai thác mẫu có thể xóa nhưng vẫn phải xây dựng lại danh sách EP khi minSup thay đổi. Khác với các thuật toán này, ITUFP lưu trữ hiệu quả thông tin khai thác, cho phép tái sử dụng mà không cần lặp lại quá trình khai thác khi giá trị K thay đổi.

1.2.2 Khai thác các mẫu từ bộ dữ liệu không chắc chắn

Khai thác tập mẫu không chắc chắn là một nhánh của khai thác dữ liệu, tập trung vào việc trích xuất các mẫu hoặc tập hợp có giá trị từ các bộ dữ liệu chứa các yếu tố không chắc chắn, chẳng hạn như xác suất tồn tại của một phần tử trong tập dữ liệu. Các phương pháp trong lĩnh vực này phải đối mặt với các thách thức liên quan đến tính phức tạp tính toán, việc quét cơ sở dữ liệu nhiều lần và xử lý dữ liệu lớn. Một số phương pháp liên quan như Apriori [2]: Sử dụng cách tiếp cận sinh ứng viên (candidate generation) và quét cơ sở dữ liệu lặp lại, nhưng gặp phải vấn đề hiệu suất. P-HMine [5] : Sử dụng cấu trúc liên kết để trích xuất mẫu, giúp giảm bớt các bước quét cơ sở dữ liệu nhưng vẫn gặp hạn chế về hiệu quả tính toán.n. LUNA [17]: Sử dụng danh sách (UP-List và CUP-List) để lưu trữ thông tin và trích xuất các mẫu dài hơn, giúp cải thiện hiệu quả khai thác.

1.2.3 Khai thác các tập mục tiện ích cao

High-Utility Itemset Mining (HUIM)[25] tập trung vào việc xác định các tập mục có giá trị cao trong cơ sở dữ liệu giao dịch, dựa trên tiện ích (utility)[20] thay vì chỉ dựa trên tần suất xuất hiện. Một thách thức lớn trong HUIM là việc xử lý không gian tìm kiếm khổng lồ, cùng với sự bùng nổ tổ hợp (combinatorial explosion)[32] các tập mục ứng viên. Các phương pháp chính như Two-Phase [26]. Ngoài ra, thuật toán HUP-Growth [24] sử dụng cấu trúc cây giúp giảm thiểu bộ nhớ trong quá trình khai thác HUI.

1.2.4 Khai thác Top-K tập mục có tiện ích cao (Top-K HUIM)

Top-K High-Utility Itemset Mining (Top-K HUIM)[14] là một nhánh của khai thác dữ liệu tập trung vào việc xác định các tập mục có giá trị cao nhất trong cơ sở dữ liệu giao dịch, mà không cần định trước một ngưỡng giá trị tối thiểu. Điều này khắc phục hạn chế của các thuật toán truyền thống khi việc chọn ngưỡng giá trị tối thiểu thường không đơn giản. Tuy nhiên, các thuật toán này phải đối mặt với các thách thức như không gian tìm kiếm lớn hơn và phụ thuộc vào các chiến lược nâng ngưỡng (threshold-raising)[8]. Một số phương pháp phổ biến như TKU Algorithm [34], REPT [6], TKO [30], kHMC [11].

II PHÂN TÍCH BÀI TOÁN

Trong cơ sở dữ liệu không chắc chắn (D), ta có:

Một tập các giao dịch $D = T_1, T_2, ... T_n$ với mỗi giao dịch T_j là một tập hợp các mục từ tập mục tổng quát $I = i_1, i_2, ...i_m$ trong đó m là số lượng mục khác nhau trong I và n là số lượng giao dịch trong D.

Trong trường hợp cơ sở dữ liệu không chắc chắn, mỗi mục $x_1 \in I$ trong một giao dịch T_j được đi kèm với một giá trị tiện ích (utility)[31] và một xác suất P($x_1|T_j$) (từ 0 đến 1) biểu thị khả năng xuất hiện của mục đó trong giao dịch.

TID	Items	Utility (u)	Probability (P)	Transaction Utility (tu)
T1	a,b,c,d	8,6,1,2	0.9,0.8,0.7,0.6	17
T2	a,b,d,e,f	4,3,4,12,2	0.8,0.6,0.7,0.9,0.5	25
T3	a,c,d	12,5,4	0.7,0.6,0.8	21
T4	b,c,e	6,5,6	0.8,0.7,0.6	17
T5	b,c,d,e	9,3,4,21	0.9,0.7,0.8,0.5	37
T6	a,b,c,d,e,f	4,3,4,4,15,2	0.8,0.7,0.6,0.7,0.9,0.5	32
T7	c,e	4,15	0.7,0.6	19
T8	c,d,e	1,4,9	0.6, 0.8, 0.7, 0.6	14

Bảng 2.1: *Ví dụ danh sách giao dịch và bảng lợi nhuận của từng mục*

2.1 Các định nghĩa và công thức đã được sử dụng trong bài toán

Định nghĩa 1: Xác suất tồn tại của một tập mẫu X trong giao dịch T_j, ký hiệu P(X, T_j) được tính bằng công thức:

$$P(X, T_j) = \prod_{x \in X} P(x, T_j) \quad (1)$$

Ví dụ: tập mẫu X = a,c P(X, T1) = 0.9 × 0.7 = 0.63

Định nghĩa 2: Độ hỗ trợ kỳ vọng của tập mẫu X trong cơ sở dữ liệu không chắc chắn D, ký hiệu là expSup(X) được tính bằng công thức:

$$expSup(X) = \sum_{j=1}^{|UDB|} (\prod_{x \in X} P(x, T_j)) = \sum_{j=1} (P(x, T_j)) \quad (2)$$

Ví dụ: Trong bảng 2.1

expSup(ac) = P(a,c, T1) + P(a,c, T3) + P(a,c, T6)

= (0.9 × 0.7) + (0.7 × 0.6) + (0.8 × 0.6) = 1.53

Định nghĩa 3: Tiện ích của một tập mẫu X trong một giao dịch T_j, ký hiệu u(X, T_j) được tính bằng công thức:

$$u(X, T_j) = \sum (x \in X) u(X, T_j) \quad (3)$$

Ví dụ: Tập mẫu X = a,c

u(X, T1) = u(a,T1) + u(c,T1) = 8 + 1 = 9

Định nghĩa 4: Tiện ích của một tập mẫu X trong cơ sở dữ liệu D được tính bằng công thức:

$$u(X) = \sum(T_j \in DX)u(X, T_j) \quad (4)$$

Ví dụ: Tập mẫu X = a,c

u(X, T1) = u(a,T1) + u(c,T1) = 8 + 1 = 9

Định nghĩa 5: Tiện ích của một giao dịch Tj ký hiệu là tu(Tj) được tính bằng công thức:

$$tu(Tj) = \sum(x \in Tj)u(X, T_j) \quad (5)$$

Ví dụ: Giao dịch T1 trong bảng 2.1

tu(T7) = u(c,T7) + u(e,T7) = 4+15 = 19

Định nghĩa 6: : Tổng tiện ích cơ sở dữ liệu D, ký hiệu là du, được tính bằng công thức:

$$du = \sum(T_j \in D) \quad (6)$$

Ví dụ: tu(Tj) = 17 + 25 + 21 + 17 + 37 + 32 + 19 + 14 = 182

Định nghĩa 7: : Tiện ích theo trọng số giao dịch(The transaction–weighted utility) của tập mẫu X trong D, ký hiệu là twu(X) được định nghĩa:

$$twu(X) = \sum_{X \subset T_j \in D} tu(T_j) \quad (7)$$

Ví dụ trong bảng 2.1:

twu (ac) = tu(T1) + tu(T3) + tu(T6) = 17 +21 + 32 = 70.

Định nghĩa 8: Tiện ích giao dịch kỳ vọng (ETU) của một mục xi trong một giao dịch T_j được tính bởi công thức:

$$ETU(x_i, T_j) = u(xi) \times P(x_i, T_j) \quad (8)$$

Ví dụ: ETU(a,T1) = u(a,T1) x P(a,T1) = 8 × 0.9 = 7.2

Định nghĩa 9: Tiện ích giao dịch kỳ vọng của một tập hợp mục $X = x_1, x_2, ..., x_k$ trong một giao dịch T_j được tính bằng công thức:

$$ETU(x_i, T_j) = \sum_{x_i \in X} ETU(x_i, T_j) \quad (9)$$

Ví dụ: Tập mẫu X = a,c

ETU(X,T1) = ETU(a, T1) + ETU(c, T1) = $(8 \times 0.9) + (1 \times 0.7) = 7.2 + 0.7 = 7.9$

Định nghĩa 10: Tiện ích giao dịch kỳ vọng của một tập mục X trong toàn bộ cơ sở dữ liệu D được tính bằng công thức:

$$ETU(X) = \sum_{T_j \in D} ETU(x_i, T_j) \quad (10)$$

Ví dụ: Tập mẫu X = a,c

ETU (X) = ETU(X, T1) + ETU(X, T3) + ETU(X, T6) = 7.9 + 11.4 + 5.6 = 24.9

Định nghĩa 11: Tập mục tiện ích cao (HUIs)[19] là một tập hợp các mục trong cơ sở dữ liệu không chắc chắn có tiện ích (utility)[12] vượt ngưỡng tiện ích tối thiểu (minUtil)[16], được xác định tương tác bởi người dùng trong quá trình tìm kiếm Top-K tập mục có tiện ích cao nhất, được định nghĩa:

$$HUIs = X | u(X) \geq minUtil \quad (11)$$

Trong đó: minUtil = du $\times \delta\%$

Ví dụ: Trong bảng 2.1, cho ngưỡng $\delta bng 15\%, cdu = 182$

Tập mẫu a: u(a) = 8 + 4 + 12 + 4 = 28 > 182 $\times 15\% =$

27.3

Vậy Tập mẫu a là một HUI

Định nghĩa 12: Top-K tập mục tiện ích cao trong cơ sở dữ liệu không chắc chắn D, ký hiệu (TKHUI)[27] là những tập mục tiện ích cao X có độ hỗ trợ kỳ vọng cao nhất:

$$TKHUI = \{X \mid X \in HUI \cap expSup(X) \geq expSupmin(TKHUI)\} \quad (12)$$

Trong đó: expSupmin(TKHUI)

Trong đó:

$$\text{expSupmin(TKHUI)} = \min\left(\text{expSup}(X) \mid X \in \text{TUHUP}\right)$$

Định nghĩa 13: Nguyên tắc giảm thiểu không gian tìm kiếm:

Nếu expSup(X)≥ minSup, các tập con Y ⊆ X cũng thỏa mãn điều kiện này.

III PHƯƠNG PHÁP THỰC HIỆN CÁC THUẬT TOÁN

3.1 Thuật toán ITUFP

Thuật toán ITUFP *(Interactive Threshold Update for Frequent Pattern Mining)*[28] được thiết kế để cung cấp một phương pháp linh hoạt và hiệu quả trong việc tìm kiếm Top-K High-Utility Itemsets từ cơ sở dữ liệu không chắc chắn. Không giống như các phương pháp khai thác mẫu truyền thống, ITUFP tích hợp khả năng tương tác để điều chỉnh ngưỡng tiện ích(utility)[1] trong quá trình khai thác. Giúp tối ưu hóa việc phát hiện các tập mẫu(itemsets)[21] có tiện ích cao nhất mà không cần xác định ngưỡng tiện ích cố định từ trước. ITUFP đặc biệt hiệu quả trong các môi trường dữ liệu không chắc chắn, nơi giá trị tiện ích có thể thay đổi dựa trên xác suất hoặc probability distribution.

Thuật toán ITUFP *(Interactive Threshold Update for Frequent Pattern Mining)* sử dụng cấu trúc dữ liệu IMCUP-list[7] kết hợp cùng các chiến lược góp phần nâng cao hiệu quả trong việc tìm ra các tập mục tiện ích cao trong hệ cơ sở dữ liệu không chắc chắn.

3.1.1 IMCUP-list

IMCUP-List *(Interactive Mining of Conditional Uncertain Probability-List) (2023)* là cấu trúc dữ liệu dùng để lưu trữ thông tin của các tập mục trong quá trình khai thác các Top-K High-Utility Itemsets. So với các danh sách truyền thống như UP-List[10] hay CUP-List[11], IMCUP-List[33] có các cải tiến để hỗ trợ:

Cập nhật thông minh: Chỉ xử lý các phần chưa được xử lý trước đó. Hiệu quả trong môi trường tương tác: Không cần xây dựng lại toàn bộ danh sách khi giá trị ngưỡng K thay đổi.

Thành phần của IMCUP-List bao gồm:

- PatternName: Tên của tập mục mà danh sách đại diện.
- expSup: Giá trị hỗ trợ kỳ vọng của tập mục
- Max Prob.TID: Xác suất tối đa của các giao dịch liên quan đến tập mục.
- Index1, Index2: Hai chỉ số đại diện cho vị trí cuối cùng đã xử lý của hai danh sách UP-Lists hoặc IMCUP-Lists[13] được kết hợp để tạo ra danh sách hiện tại.

  Index1 và Index2 có vai trò theo dõi trạng thái xử lý

các phần tử trong danh sách cha. Khi cần cập nhật, thuật toán chỉ xử lý các phần tử từ các chỉ số này trở đi. Điều này giúp tránh việc xử lý lại các phần tử đã được xử lý trước đó, tăng hiệu suất.

3.1.2 Các chiến lược

Chiến lược nâng ngưỡng (Raising the Threshold): Chiến lược này bắt đầu với một ngưỡng tối thiểu (minSup)[23] là 0 và sau đó tăng dần giá trị này trong quá trình khai thác. Điều này giúp loại bỏ những mẫu không đủ điều kiện ngay từ đầu, giảm thiểu số lượng mẫu cần kiểm tra.

Chiến lược cắt tỉa (Pruning Strategy by Raising the Threshold): Trong chiến lược này, thuật toán chỉ kết hợp những danh sách mà giá trị hỗ trợ kỳ vọng (expSup)[15] lớn hơn giá trị minSup đã cập nhật. Điều này tiết kiệm thời gian và tài nguyên bằng cách tránh xử lý những mẫu không có khả năng trở thành Top-K.

3.1.3 Mã giả của thuật toán

Khởi tạo ngưỡng tiện ích và tương tác với người dùng: Thuật toán ITUFP bắt đầu với một ngưỡng tiện ích ban đầu. Người dùng có thể thay đổi ngưỡng này trong quá trình tìm kiếm, giúp thuật toán linh hoạt hơn và cải thiện kết quả. Khai thác các tập mẫu tiềm năng: Thuật toán sinh ra các itemsets ứng viên từ cơ sở dữ liệu và tính toán tiện ích của chúng dựa trên tần suất và tiện ích giao dịch. Trong môi trường không chắc chắn, tiện ích được tính theo xác suất hoặc phân phối khả năng.

Cập nhật ngưỡng tiện ích: Sau khi tính toán tiện ích, thuật toán so sánh với ngưỡng hiện tại. Nếu itemset có tiện ích cao hơn, ngưỡng được cập nhật để tiếp tục tìm kiếm các itemsets có tiện ích cao hơn. Điều này giúp giảm chi phí tính toán và bộ nhớ.

Lặp lại cho đến khi đạt được Top-K itemsets: Quá trình khai thác và cập nhật ngưỡng tiếp tục cho đến khi tìm được Top-K high-utility itemsets. Kết quả cuối cùng là danh sách các itemsets có utility cao nhất từ dữ liệu không chắc chắn.

Giải thuật 1 (Thuật toán chính): Thuật toán ITUFP

Input: file_path (str), percentage (float), k (int)
Output: Top-K patterns with high utility 1. **FUNCTION** ITUFPALGORITHM:
2. **INIT** VARIABLES:
START_TIME, END_TIME, DATABASE_UTIL, TOP_PATTERNS,
THRESHOLD, MIN_UTIL.
3. **FUNCTION** READ_DATA(FILE_PATH, PERCENTAGE, K):
- **READ** FILE, **PARSE** DATA.
- **CALCULATE** MIN_UTIL = DATABASE_UTIL * PERCENTAGE.
4. **FUNCTION** PROCESS_DATA(ITEM_NAME, PROB_LINE, UTIL_LINE, TID):
- **PARSE** PROBABILITIES, UTILITIES.
- **UPDATE** DATABASE_UTIL, **STORE** CUPITUFP.
5. **FUNCTION** COMBINE_CUPITUFP(INFOITEMX, INFOITEMY):
- **RETURN** COMBINED CUPITUFP OBJECT.
6. **FUNCTION** GET_TOP_K_PATTERNS(MIN_UTIL, K):
- **SORT** SINGLE_INFOITEM.
- **SELECT** TOP K WITH UTILITY > MIN_UTIL.
7. **FUNCTION** INTERACTIVE_SEARCH(CURRENTCUPITUFP, K):
- **FOR EACH** PAIR:
- **COMPUTE** OVERESTIMATE, **COMBINE** PATTERNS.
- **UPDATE** TOP_K.
8. **FUNCTION** UPDATE_TOP_K(COMBINED, K):
- **IF** UTILITY >= MIN_UTIL: **ADD** TO TOP_PATTERNS.
- **REMOVE** EXCESS ITEMS.
9. **FUNCTION** RUN_ITUFP(FILE_PATH, PERCENTAGE, K):
- **CALL** READ_DATA, GET_TOP_K_PATTERNS, INTERACTIVE_SEARCH.
- **DISPLAY** RESULTS.
10. **FUNCTION** PRINT_STATS(OUTPUT_FILE_PATH):
- **WRITE** STATISTICS TO FILE.

Bảng 3.1: *Mã giả Thuật toán chính ITUFP*

Giải thuật 1 (Thuật toán chính): Thuật toán ITUFP

Input: file_path (str), percentage (float), k (int)

Output: Top-K patterns with high utility

1. **FUNCTION** ITUFPALGORITHM:
2. **INIT** VARIABLES:
START_TIME, END_TIME, DATABASE_UTIL,
TOP_PATTERNS, THRESHOLD, MIN_UTIL.
3. **FUNCTION** READ_DATA(FILE_PATH, PERCENTAGE, K):
   - **READ** FILE, **PARSE** DATA.
   - **CALCULATE** MIN_UTIL = DATABASE_UTIL * PERCENTAGE.
4. **FUNCTION** PROCESS_DATA(ITEM_NAME, PROB_LINE,
UTIL_LINE, TID):
   - **PARSE** PROBABILITIES, UTILITIES.
   - **UPDATE** DATABASE_UTIL, **STORE** CUPITUFP.
5. **FUNCTION** COMBINE_CUPITUFP(INFOITEMX, INFOITEMY):
   - **RETURN** COMBINED CUPITUFP OBJECT.
6. **FUNCTION** GET_TOP_K_PATTERNS(MIN_UTIL, K):
   - **SORT** SINGLE_INFOITEM.
   - **SELECT** TOP K WITH UTILITY > MIN_UTIL.
7. **FUNCTION** INTERACTIVE_SEARCH(CURRENTCUPITUFP, K):
   - **FOR EACH** PAIR:
      - **COMPUTE** OVERESTIMATE, **COMBINE** PATTERNS.
      - **UPDATE** TOP_K.
8. **FUNCTION** UPDATE_TOP_K(COMBINED, K):
   - **IF** UTILITY >= MIN_UTIL: **ADD** TO TOP_PATTERNS.
   - **REMOVE** EXCESS ITEMS.
9. **FUNCTION** RUN_ITUFP(FILE_PATH, PERCENTAGE, K):
   - **CALL** READ_DATA, GET_TOP_K_PATTERNS,
INTERACTIVE_SEARCH.
   - **DISPLAY** RESULTS.
10. **FUNCTION** PRINT_STATS(OUTPUT_FILE_PATH):
   - **WRITE** STATISTICS TO FILE.

Bảng 3.2: *Mã giả Hỗ trợ Thuật toán ITUFP*

Giải thuật 1 (Thuật toán chính): Giải thuật ITUFP

Input: name (str), transaction_information_list (list, tùy chọn)

Output: Đối tượng CupITUFP chứa thông tin item và các thuộc tính tiện ích

1. **CLASS** CupITUFP:
2. **CONSTRUCTOR** __init__(name, transaction_information_list=None):
   - Gán name và transaction_information_list
(nếu rỗng thì gán danh sách trống).
   - Tính exp_sup, utility, trans_wei_utility, max_probability từ
transaction_information_list hoặc gán 0 nếu rỗng.
3. **METHOD** update(probability, TID, transaction_utility, util_value):
   - Làm tròn xác suất, tạo giao dịch mới (TepITUFP), thêm vào danh sách.
   - Cập nhật exp_sup, utility, trans_wei_utility và max_probability.
4. **METHOD** combine_transaction_information(list_x, list_y):
   - Tạo danh sách rỗng combined_list.
   - Duyệt list_x và list_y, kết hợp các giao dịch có cùng TID.
   - Trả về combined_list.
5. **METHOD** combine_together(other_CupITUFP):
   - Gộp tên và danh sách giao dịch từ cả hai đối tượng.
   - Trả về đối tượng CupITUFP mới.
6. **METHOD** __repr__():
   - Trả về chuỗi biểu diễn đối tượng với name, exp_sup, utility.

Bảng 3.3: *Mã giả Hỗ trợ Thuật toán ITUFP*

Thuật toán ITUFP (Itemset Tree Utility Mining Algorithm) được thiết kế để tìm kiếm các mẫu có tiện ích cao từ cơ sở dữ liệu, với ba đầu vào là đường dẫn tệp dữ liệu (file_path), tỷ lệ phần trăm tối thiểu (percentage), và số lượng mẫu top-k cần tìm (k). Thuật toán bắt đầu với việc khởi tạo các biến cần thiết như thời gian bắt đầu và kết thúc (start_time, end_time), tiện ích cơ sở dữ liệu (database_util), danh sách các mẫu top (top_patterns), ngưỡng tiện ích (threshold), và tiện ích tối thiểu (min_util) (Dòng 2). Tiếp theo, hàm read_data sẽ đọc và phân tích dữ liệu từ tệp, đồng thời tính toán tiện ích tối thiểu dựa trên tỷ lệ phần trăm (Dòng 3). Sau đó, hàm process_data sẽ xử lý từng mục dữ liệu, tính toán tiện ích của từng mục và cập nhật cơ sở dữ liệu (Dòng 4). Tiếp theo, hàm combine_CupITUFP kết hợp

các mẫu với nhau thành một mẫu chung có tiện ích cao hơn (Dòng 5). Sau đó, hàm get_top_k_patterns sẽ sắp xếp các mẫu và chọn ra top-k mẫu có tiện ích lớn hơn tiện ích tối thiểu (Dòng 6).

Thuật toán tiếp tục với hàm interactive_search thực hiện tìm kiếm tương tác, trong đó các mẫu được kết hợp và kiểm tra tính hợp lệ dựa trên tiện ích ước tính (Dòng 7). Mỗi khi tìm thấy mẫu hợp lệ, hàm update_top_k sẽ cập nhật danh sách top-k bằng cách thêm mẫu mới nếu nó thỏa mãn điều kiện tiện ích và loại bỏ mẫu không cần thiết (Dòng 8). Cuối cùng, hàm run_ITUFP sẽ gọi các hàm trên để thực hiện quá trình đọc dữ liệu, tìm kiếm mẫu top-k, và hiển thị kết quả (Dòng 9). Kết quả sau đó được ghi vào tệp đầu ra qua hàm print_stats (Dòng 10).

Giải thuật ITUFP sử dụng lớp TepITUFP để đại diện cho một giao dịch trong cơ sở dữ liệu, với các thuộc tính như ID giao dịch (TID), xác suất tồn tại, tiện ích và tiện ích giao dịch. Đầu vào của lớp này bao gồm các tham số là TID, probability, util và transaction_utility, trong đó TID là ID của giao dịch, probability là xác suất tồn tại của giao dịch (giá trị trong khoảng [0, 1]), util là giá trị tiện ích của giao dịch, và transaction_utility là tiện ích giao dịch, được tính toán từ các mục trong giao dịch. Đầu ra của lớp này là một đối tượng TepITUFP chứa đầy đủ thông tin về giao dịch.

Lớp TepITUFP bắt đầu với phương thức khởi tạo __init__ (Dòng 2), trong đó các tham số TID, probability, util và transaction_utility được lưu trữ trong đối tượng, với xác suất được làm tròn đến hai chữ số thập phân (Dòng 2). Phương thức combine_together (Dòng 3) cho phép kết hợp hai giao dịch, tính toán xác suất kết hợp và tiện ích kết hợp từ hai giao dịch, và trả về một đối tượng TepITUFP mới với các giá trị đã được tính toán (Dòng 3). Cuối cùng, phương thức __repr__ (Dòng 4) trả về chuỗi đại diện cho đối tượng TepITUFP, giúp người dùng dễ dàng xem thông tin của giao dịch dưới dạng chuỗi, bao gồm ID giao dịch, xác suất, tiện ích và tiện ích giao dịch (Dòng 4).

Lớp CupITUFP đại diện cho các item trong khai thác dữ liệu tiện ích, với các thuộc tính như tên item, danh sách giao dịch, hỗ trợ kỳ vọng (exp_sup), tiện ích (utility), tiện ích trọng số giao dịch (trans_wei_utility), và xác suất lớn nhất (max_probability). Hàm khởi tạo thiết lập các giá trị này từ danh sách giao dịch hoặc gán giá trị mặc định nếu danh sách rỗng. Phương thức update thêm giao dịch mới và cập nhật các thuộc tính liên quan. Phương thức combine_transaction_information kết hợp hai danh sách giao dịch dựa trên TID, trong khi combine_together gộp hai đối tượng CupITUFP thành một. Cuối cùng, phương thức __repr__ trả về chuỗi biểu diễn thông tin đối tượng, giúp việc kiểm tra dữ liệu thuận tiện hơn. Lớp này hỗ trợ hiệu quả các thao tác tổ hợp, cập nhật và quản lý thông tin tiện ích trong khai thác dữ liệu.

3.2 Thuật toán TKU

Thuật toán TKU (Top-K Utility Itemset Mining) là một phương pháp mạnh mẽ trong việc khai thác các tập mục có tiện ích cao nhất từ cơ sở dữ liệu. Mục tiêu của thuật toán là tìm ra Top-K tập mục có utility cao nhất mà không cần xác định ngưỡng utility tối thiểu. Quá trình này bắt đầu bằng việc lọc dữ liệu để giảm thiểu số lượng tập mục cần xử lý, sau đó sinh ra các ứng viên mới và

đánh giá utility thực tế của chúng. Cuối cùng, thuật toán duy trì một danh sách Top-K để lưu trữ những tập mục có utility cao nhất, giúp cải thiện hiệu quả và độ chính xác trong việc khai thác dữ liệu. Với khả năng tối ưu không gian tìm kiếm và loại bỏ các tập mục không có giá trị, TKU là một công cụ hữu ích trong việc khai thác dữ liệu lớn, đặc biệt là trong các bài toán yêu cầu tìm kiếm các itemset có tiện ích cao mà không cần phải áp dụng ngưỡng utility cố định.

Giải thuật 2 (Thuật toán chính): Thuật toán TKU

Input: file_path (str), percentage (float), k (int)

Output: Top-k itemsets có tiện ích cao nhất, Expected Support (Exp_Sup), Utility, và thông tin thống kê về thời gian chạy, bộ nhớ sử dụng, và số lượng ứng viên.

1. **INITIALIZE:** start_timeskip = 0
   - end_timeskip = 0
   - database_size = 0
   - database_utility = 0
   - candidates = 0
   - top_TKU = empty list
   - single_info_item = empty dictionary
   - threshold = negative infinity
   - min_utility = negative infinity
   - peak_memory_usage = 0
2. **FUNCTION read_data(file_path, percentage, k):**
   - SPLIT file_path thành các đường dẫn (file1, file2)
   - TRY:
     - OPEN file1 và file2
     - READ dữ liệu từ cả hai file
     - item_name = dòng đầu tiên từ file1, chia bởi khoảng trắng
     - FOR mỗi probability_line và utility_line trong zip(file1, file2):
       - XỬ LÝ dữ liệu (probability_line, utility_line)
     - SET min_utility = database_utility * percentage
     - INH XUẤT min_utility
   - EXCEPTION:
     - XỬ LÝ các lỗi (file không tìm thấy, lỗi khác)
3. **FUNCTION process_data(item_name, probability_line, utility_line, TID):**
   - SPLIT probability_line thành probability_list
   - SPLIT utility_line thành items_utility và transaction_utility
   - TẠO item_utility_list bằng cách ánh xạ items_utility vào utility_list
   - THÊM transaction_utility vào database_utility
   - FOR mỗi probability trong probability_list:
     - IF probability > 0:
       - TRY:
         - GET item từ item_name[i]
         - GET utility_value từ item_utility_list
         - IF InfoItem tồn tại cho item:
           - CẬP NHẬT InfoItem với probability, TID, utility_value
         - ELSE:
           - TẠO InfoItem mới (TID, probability, utility_value)
           - THÊM vào single_info_item
       - EXCEPTION:
         - XỬ LÝ các lỗi trong quá trình xử lý xác suất
4. **FUNCTION combine_info_items(info_item_X, info_item_Y):**
   - KẾT HỢP info_item_X và info_item_Y thành một InfoItem mới
   - LÀM TRÒN exp_sup của InfoItem kết hợp đến 2 chữ số thập phân
   - TRẢ VỀ InfoItem kết hợp

Bảng 3.4: *Mã giả Thuật toán chính TKU*

Hỗ trợ Giải thuật 2: Thuật toán TKU Class Tep

Input: TID: ID của giao dịch.
probability: Xác suất tồn tại của giao dịch, giá trị trong khoảng [0, 1].
utility: Giá trị tiện ích của giao dịch.
transaction_utility:
Tiện ích giao dịch, được tính toán từ các mục trong giao dịch.
Output: Tep object: Đối tượng đại diện cho giao dịch, chứa thông tin về ID, xác suất, tiện ích, và tiện ích giao dịch.

1. **CLASS Tep:** Khai báo các thuộc tính:
   - TID: mã giao dịch
   - probability: xác suất tồn tại của giao dịch
   - utility: giá trị tiện ích của giao dịch
   - transaction_utility: tiện ích giao dịch
2. **KHỞI TẠO đối tượng Tep:** Input: TID, probability, utility, transaction_utility
   - Gán giá trị cho các thuộc tính của đối tượng Tep:
     - person.TID = TID
     - person.probability = round(probability, 2)
     - person.utility = utility
     - person.transaction_utility = transaction_utility
3. **COMBINE hai đối tượng Tep lại với nhau:**
   - Input: other_transaction (đối tượng Tep cần kết hợp)
   - Tính toán:
     - combined_probability = round(person.probability * other_transaction.probability, 2)
     - combined_utility = person.utility + other_transaction.utility
   - Output: Trả về đối tượng Tep mới với các giá trị đã tính toán:
   return Tep(person.TID, combined_probability, combined_utility, other_transaction.transaction_utility)
4. **SẮP XẾP các giao dịch theo giá trị transaction_utility:**
   - Tạo danh sách các đối tượng Tep từ các giao dịch ban đầu
   - Sắp xếp theo thứ tự giảm dần của transaction_utility
5. **CHỌN Top-K giao dịch có tiện ích cao nhất:**
   - Output: Trả về danh sách các Tep có transaction_utility cao nhất.

Bảng 3.5: *Mã giả Hỗ trợ Thuật toán TKU*

Giải thuật 2: Thuật toán TKU Class Cup

Input: name (str), transaction_information_list (list)
Output: Cup object chứa thông tin về item.

1. **CLASS Cup:**
   - Thuộc tính: name, exp_sup, transaction_information_list, max_probability, trans_wei_utility, utility, last
2. **METHOD __init__(name, transaction_information_list):**
   - Khởi tạo các thuộc tính: tính exp_sup, max_probability, trans_wei_utility, utility từ transaction_information_list
3. **METHOD update(probability, TID, transaction_utility, utility_value):**
   - Cập nhật exp_sup, max_probability, trans_wei_utility, utility, thêm giao dịch vào transaction_information_list
4. **METHOD combine_transaction_info(list_x, list_y):**
   - Kết hợp hai danh sách Tep theo TID, tính xác suất và tiện ích kết hợp
5. **METHOD combine_together(other_item):**
   - Kết hợp Cup hiện tại với Cup khác, tạo tên mới và kết hợp danh sách giao dịch
6. **METHOD __repr__():**
   - Trả về thông tin của Cup dưới dạng chuỗi

Bảng 3.6: *Mã giả Hỗ trợ Thuật toán TKU*

Giải thuật TKU (Thuật toán chính ban đầu) là một thuật toán nhằm tìm kiếm các itemsets có tiện ích cao nhất từ cơ sở dữ liệu. Đầu vào của thuật toán gồm có file_path, đường dẫn tới các file cơ sở dữ liệu chứa xác suất và tiện ích của các itemsets; percentage, ngưỡng tiện ích dùng để xác định các itemsets có tiện ích cao; và k, số lượng mẫu top-k cần tìm kiếm. Thuật toán sẽ xuất ra các top-k itemsets có tiện ích cao nhất cùng với các chỉ số liên quan như Expected Support (Exp_Sup) và Utility, đồng thời cung cấp thông tin thống kê về thời gian chạy, bộ nhớ sử dụng và số lượng ứng viên.

Thuật toán bắt đầu bằng việc khởi tạo các biến như start_timeskip, end_timeskip, database_size, database_utility, và candidates với giá trị ban đầu là 0, và khởi tạo các danh sách và từ điển như top_TKU và single_info_item là rỗng. Các ngưỡng như threshold và min_utility được khởi tạo là vô cùng âm (negative

infinity), trong khi peak_memory_usage được khởi tạo là 0. Tiếp theo, hàm read_data sẽ nhận đầu vào là file_path, percentage, và k, chia file_path thành các đường dẫn tệp (file1, file2), sau đó mở và đọc dữ liệu từ các tệp này. Hàm sẽ tách dòng đầu tiên của file1 để lấy item_name, sau đó xử lý các dòng dữ liệu từ file1 và file2 theo từng cặp xác suất và tiện ích. Cuối cùng, hàm này tính toán min_utility bằng cách nhân database_utility với percentage, và in ra giá trị của min_utility.

Trong hàm process_data, mỗi dòng dữ liệu được xử lý, trong đó xác suất và tiện ích của item được phân tách và lưu trữ. Cụ thể, dòng xác suất được tách thành danh sách probability_list, và dòng tiện ích được tách thành các giá trị tiện ích cho từng item (items_utility) và tiện ích giao dịch (transaction_utility). Tiện ích giao dịch được cộng vào database_utility. Sau đó, hàm duyệt qua từng xác suất trong probability_list và nếu xác suất lớn hơn 0, thuật toán sẽ kiểm tra xem InfoItem cho item đó đã tồn tại chưa. Nếu đã tồn tại, thuật toán sẽ cập nhật InfoItem với xác suất, TID và tiện ích; nếu chưa, thuật toán sẽ tạo một InfoItem mới và thêm vào từ điển single_info_item. Trong quá trình này, các lỗi liên quan đến việc xử lý xác suất sẽ được bắt và xử lý trong phần EXCEPTION. Cuối cùng, hàm combine_info_items được sử dụng để kết hợp hai InfoItem, tính toán exp_sup của InfoItem kết hợp và làm tròn giá trị này đến hai chữ số thập phân. Sau đó, nó trả về một InfoItem mới chứa thông tin đã được kết hợp.

Giải thuật TKU sử dụng một lớp Tep để đại diện cho các giao dịch và thực hiện các phép toán cần thiết để tính toán và xác định các giao dịch có tiện ích cao nhất. Đầu vào của thuật toán bao gồm TID, probability, utility, và transaction_utility, với mục tiêu tạo ra đối tượng Tep chứa thông tin về giao dịch, bao gồm mã giao dịch, xác suất, tiện ích, và tiện ích giao dịch. Trong lớp Tep, các thuộc tính cơ bản được khai báo như TID (mã giao dịch), probability (xác suất tồn tại của giao dịch), utility (tiện ích của giao dịch), và transaction_utility (tiện ích giao dịch). Khi khởi tạo đối tượng Tep, các giá trị được gán cho các thuộc tính của đối tượng này, trong đó probability được làm tròn đến hai chữ số thập phân, giúp lưu trữ giá trị xác suất chính xác và đồng nhất. Tiếp theo, để kết hợp hai đối tượng Tep lại với nhau, thuật toán sử dụng phương thức combine, trong đó tính toán combined_probability bằng cách nhân giá trị xác suất của hai đối tượng và làm tròn kết quả, đồng thời cộng giá trị utility của cả hai giao dịch để tạo ra combined_utility. Sau đó, một đối tượng Tep mới được trả về với các giá trị đã tính toán từ hai đối tượng ban đầu, bao gồm cả tiện ích giao dịch từ giao dịch thứ hai. Cuối cùng, thuật toán sẽ thực hiện việc sắp xếp các giao dịch dựa trên giá trị transaction_utility. Các giao dịch được sắp xếp theo thứ tự giảm dần của transaction_utility, và sau khi sắp xếp xong, thuật toán chọn ra Top-K giao dịch có tiện ích cao nhất, trả về danh sách các giao dịch này làm kết quả đầu ra của thuật toán.

Thuật toán TKU sử dụng một lớp Cup để đại diện cho các itemsets, cung cấp các phương thức để tính toán và kết hợp thông tin từ các giao dịch. Đầu vào của thuật toán bao gồm name (tên item) và transaction_information_list (danh sách các giao dịch,

mỗi giao dịch là một đối tượng Tep). Kết quả đầu ra là một đối tượng Cup chứa thông tin về item, bao gồm các thuộc tính và phương thức tính toán tiện ích, xác suất và các giá trị liên quan khác.

Trong lớp Cup, các thuộc tính cơ bản được khai báo bao gồm name (tên item), exp_sup (Expected Support - hỗ trợ kỳ vọng), transaction_information_list (danh sách giao dịch liên quan đến item), max_probability (xác suất tối đa), trans_wei_utility (tiện ích trọng số của giao dịch), utility (tiện ích của item), và last (thông tin cuối cùng liên quan đến item). Khi khởi tạo đối tượng Cup, phương thức __init__ tính toán các giá trị như exp_sup, max_probability, trans_wei_utility, và utility dựa trên danh sách các giao dịch trong transaction_information_list. Phương thức update được sử dụng để cập nhật thông tin của Cup. Cụ thể, nó nhận các đầu vào như probability (xác suất của giao dịch), TID (mã giao dịch), transaction_utility (tiện ích giao dịch), và utility_value (tiện ích của item). Sau đó, phương thức này sẽ tính toán và cập nhật các giá trị của exp_sup, max_probability, trans_wei_utility, và utility của Cup, đồng thời thêm giao dịch vào danh sách transaction_information_list. Để kết hợp thông tin từ hai danh sách giao dịch, thuật toán sử dụng phương thức combine_transaction_info. Phương thức này nhận vào hai danh sách giao dịch (list_x, list_y), kết hợp chúng theo TID, và tính toán lại xác suất và tiện ích cho các itemsets kết hợp. Phương thức combine_together cho phép kết hợp hai đối tượng Cup lại với nhau, tạo tên mới cho Cup và kết hợp danh sách giao dịch từ hai đối tượng. Kết quả là một đối tượng Cup mới với các thông tin kết hợp.

Cuối cùng, phương thức __repr__ sẽ trả về thông tin của đối tượng Cup dưới dạng chuỗi, giúp hiển thị thông tin về item, tiện ích, xác suất và các thuộc tính quan trọng khác của đối tượng.

3.3 Thuật toán TUHUFP

Thuật toán TUHUFP (Top Uncertain High Utility Frequent Pattern) là một phương pháp tiên tiến được thiết kế để khai thác các tập phổ biến tiện ích cao (High Utility Frequent Patterns) trong môi trường cơ sở dữ liệu không chắc chắn (Uncertain Databases). Thuật toán này vận hành dựa trên cấu trúc dữ liệu CUP-list (Candidate Utility Pattern List), giúp tối ưu hóa việc lưu trữ và quản lý thông tin tiện ích, cũng như xác suất tồn tại của các tập mục tiềm năng trong dữ liệu.

Thuật toán TUHUFPtriển khai phương thức TUHUFP_Search để tìm kiếm Top-K tập phổ biến tiện ích cao từ cơ sở dữ liệu không chắc chắn. Phương thức này giảm không gian tìm kiếm thông qua ngưỡng tiện ích động (Dynamic Utility Threshold) và các tiêu chí như lọc xác suất (Probability Filtering) và cắt tỉa tiện ích (Utility Pruning).

Nhờ các chiến lược tối ưu, thuật toán nhanh chóng xác định các tập tiềm năng, tối ưu hóa hiệu suất khai thác, giảm thời gian xử lý và đảm bảo độ chính xác. TUHUFP phù hợp cho các ứng dụng phân tích thị trường, quản lý tồn kho, và hỗ trợ ra quyết định dựa trên dữ liệu không chắc chắn.

3.3.1 CUP-List

CUP-list là cấu trúc dữ liệu hỗ trợ khai thác Top-K tập phổ biến không chắc chắn có tiện ích cao (TUHUFP).

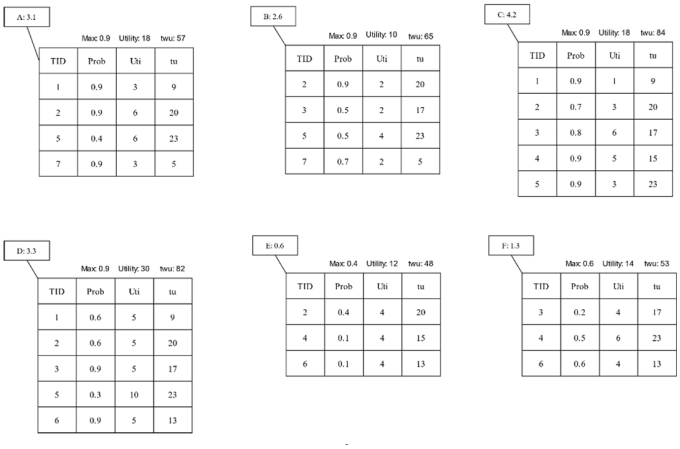
Cấu trúc này bao gồm:

- Tên tập mẫu (Pattern Name): Xác định tập mẫu.
- Độ hỗ trợ mong đợi (Expected Support): Mức độ phổ biến trung bình của tập mẫu.
- TEP-list: Tập hợp lưu thông tin các giao dịch chứa tập mẫu.
- Thuộc tính max:
- Thuộc tính max lưu giá trị xác suất mong đợi lớn nhất của tập mẫu trong TEP-list. Ngưỡng tối đa (Overestimate) của một tập XY được xác định bằng công thức expSup(X) × max(Y). Việc tính toán trước ngưỡng tối đa và so sánh với ngưỡng Top-K cho phép nhanh chóng xác định khả năng tập mẫu thuộc Top-K, từ đó giảm không gian lưu trữ và cải thiện hiệu suất khai thác.
- Thuộc tính utility: Tiện ích của tập mẫu trong cơ sở dữ liệu.
- Thuộc tính TWU: Tổng tiện ích của các giao dịch trong TEP-list.

Dữ liệu TEP-list gồm:

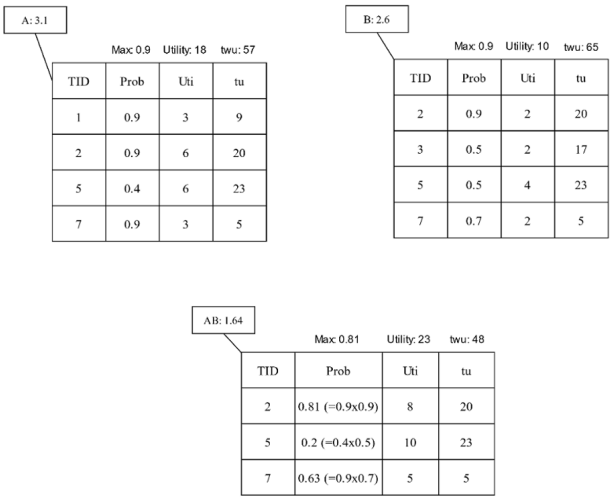
- Địa chỉ giao dịch (TID)
- Xác suất mong đợi
- Tiện ích tập mẫu và giao dịch

Dựa trên dữ liệu đã cung cấp, Hình 3.1 minh họa cấu trúc CUP-list của các sản phẩm trong cơ sở dữ liệu. Việc tổ chức dữ liệu theo cấu trúc này cho phép tính toán hiệu quả độ hỗ trợ mong đợi và tiện ích của mẫu mà không cần quét lại toàn bộ cơ sở dữ liệu.



Hình 3.1: *Minh họa cấu trúc CUP – list*

Hình 3.2 minh họa quá trình kết hợp mẫu A, B từ CUP-list của mục A và mục B. Đầu tiên, TEP-list của mẫu được xác định dựa trên các TID trùng lặp (2, 5, 7). Sau đó, các giá trị xác suất mong đợi và tiện ích của mẫu trong TEP-list được tính toán theo các định nghĩa 1 và 3.



Hình 3.2: *Minh họa quá trình kết hợp mẫu*

3.3.2 Các chiến lược

Chiến lược 1 – Nâng ngưỡng (Threshold Raising): Danh sách Top-K tập phổ biến có tiện ích cao đầu tiên

được lưu trữ trong một mảng sắp xếp giảm dần theo độ hỗ trợ mong đợi (expSup). Ngưỡng của Top-K là độ hỗ trợ mong đợi của phần tử cuối cùng trong mảng. Khi một tập mẫu mới có độ hỗ trợ vượt ngưỡng, tập mẫu này được thêm vào danh sách, phần tử cuối bị loại bỏ, và ngưỡng được cập nhật. Chiến lược này giúp giảm thiểu thời gian tìm kiếm và thay thế các tập mẫu trong danh sách.

Chiến lược 2 – Cắt tỉa theo ngưỡng nâng cấp (Pruning by Raised Threshold):

Trong quá trình tạo tập mẫu mới trong phương thức TUHUFP_Search, thuật toán sẽ loại bỏ các tập mẫu có độ hỗ trợ mong đợi thấp hơn ngưỡng hiện tại. Điều này giảm đáng kể thời gian xử lý và không gian lưu trữ.

Chiến lược 3 – Cắt tỉa theo TWU (Transaction-Weighted Utility Pruning):

Thuật toán chỉ thêm tập mẫu mới vào danh sách ứng viên nếu giá trị TWU của tập mẫu đạt ngưỡng tiện ích tối thiểu. Tập mẫu không đạt ngưỡng sẽ bị loại bỏ, giúp giảm số lượng ứng viên cần xử lý.

Chiến lược 4 – Cắt tỉa theo ngưỡng tối đa (Overestimate Pruning):

Ngưỡng tối đa của một tập mẫu XY được xác định bởi expSup(X) × max(Y). Nếu giá trị này nhỏ hơn ngưỡng Top-K, tập mẫu được loại bỏ; ngược lại, tập mẫu được đánh giá là tiềm năng. Chiến lược này cắt giảm không gian lưu trữ và nâng cao hiệu quả khai thác.

3.3.3 Mã giả thuật toán

Các bước quan trọng trong thuật toán TUHUFP được thực hiện như sau:

1. Xây dựng CUP-list cho tập 1-mẫu: Truy xuất CUP-list của các tập 1-mẫu (chỉ chứa một phần tử) từ cơ sở dữ liệu D, sau đó sắp xếp chúng theo thứ tự giảm dần của độ hỗ trợ mong đợi (expSup).

2. Khởi tạo danh sách Top-K: Chọn k phần tử đầu tiên (hoặc ít hơn nếu cơ sở dữ liệu không đủ) có tiện ích cao nhất (HUP) để đưa vào danh sách Top-K.

3. Lọc mẫu dựa trên TWU: Chỉ giữ lại các tập mẫu có TWU lớn hơn hoặc bằng ngưỡng tiện ích tối thiểu. Các mẫu không đạt yêu cầu sẽ bị loại bỏ, giúp thu hẹp phạm vi tìm kiếm.

4. Thiết lập ngưỡng Top-K: Khi danh sách Top-K đạt đủ k phần tử, ngưỡng hỗ trợ được thiết lập bằng giá trị độ hỗ trợ mong đợi thấp nhất trong danh sách.

5. Thực hiện tìm kiếm với TUHUFP_Search:

    5.1. Gọi phương thức TUHUFP_Search với các tham số k và danh sách các mẫu tham gia.

    5.2. Sử dụng chiến lược chia để trị để mở rộng CUP-list cho các tập mẫu lớn hơn từ các CUP-list của tập mẫu ban đầu.

    5.3. Bỏ qua các tập mẫu có độ hỗ trợ mong đợi hoặc tiện ích thấp hơn ngưỡng hiện tại.

    5.4. Các tập mẫu thỏa điều kiện sẽ được thêm vào danh sách kết quả và tiếp tục kết hợp để tìm kiếm các tập mẫu lớn hơn.

Quá trình này tiếp diễn cho đến khi tất cả các tập mẫu được xem xét hoàn tất.

Thuật toán 3: Thuật toán TUHUFP

Đầu vào: U (cơ sở dữ liệu không chắc chắn), H (cơ sở dữ liệu tiện ích), percentage (float), k (int)

Đầu ra: k tập mẫu phổ biến không chắc chắn có tiện ích cao đầu tiên.

1. **Đọc cơ sở dữ liệu U và H để tạo danh sách CUP (1-mẫu):**
   - Đọc các giá trị từ cơ sở dữ liệu U và H và tạo danh sách CUP (1-mẫu).
2. **Tính tiện ích của cơ sở dữ liệu:**
   - databaseUtil ← SUM(tu(Tq) ∈ H)
   - minUtil ← databaseUtil * percentage
   - threshold ← 0
3. **Sắp xếp lại danh sách CUP (1-mẫu) theo expSup:**
   - Sắp xếp danh sách CUP (1-mẫu) theo giá trị expSup giảm dần.
4. **Candidates ← Candidates ∪ CUPx ∈ CUP, CUPx.TWU ≥ minUtil:**
   - Thêm CUPx vào danh sách Candidates nếu CUPx.TWU ≥ minUtil.
5. **topUHUFP ← topUHUFP ∪ CUPx ∈ CUP, CUPx.utility ≥ minUtil:**
   - Thêm CUPx vào topUHUFP nếu CUPx.utility ≥ minUtil.
6. **threshold ← phần tử cuối CUPx ∈ CUP, topUHUFP.size = k:**
   - Lấy phần tử cuối từ CUPx trong CUP và đặt threshold bằng giá trị của nó.
   - Nếu topUHUFP.size = k, thực hiện TUHUFPSearch(Candidates, threshold, minUtil, k).

Bảng 3.7: *Mã giả Thuật toán TUHUFP*

3.4 Thuật toán TUHUFP - Growth

Thuật toán TUHUFP-Growth (Top Uncertain High Utility Frequent Pattern Growth) là một phương pháp khai thác dữ liệu sử dụng cấu trúc UHUFP-tree, một phiên bản mở rộng của FP-tree, để lưu trữ và tìm kiếm các mẫu phổ biến có tiện ích cao từ cơ sở dữ liệu không chắc chắn. Thuật toán này tối ưu hóa quá trình khai thác bằng cách áp dụng các chiến lược cắt tỉa và sử dụng cấu trúc dữ liệu hiệu quả, tập trung vào việc tìm kiếm các tập mục Top-K có giá trị cao nhất.

3.4.1 UHUFP-Tree

UHUFP-tree là một cấu trúc dữ liệu mở rộng từ FP-tree, được thiết kế để lưu trữ và quản lý thông tin về các mẫu phổ biến và tiện ích của chúng trong cơ sở dữ liệu không chắc chắn. Cấu trúc này bao gồm các thành phần chính:

1. Xây dựng CUP-list cho tập 1-mẫu: Truy xuất CUP-list của các tập 1-mẫu (chỉ chứa một phần tử) từ cơ sở dữ liệu D, sau đó sắp xếp chúng theo thứ tự giảm dần của độ hỗ trợ mong đợi (expSup).

2. Khởi tạo danh sách Top-K: Chọn k phần tử đầu tiên (hoặc ít hơn nếu cơ sở dữ liệu không đủ) có tiện ích cao nhất (HUP) để đưa vào danh sách Top-K.

3. Lọc mẫu dựa trên TWU: Chỉ giữ lại các tập mẫu có TWU lớn hơn hoặc bằng ngưỡng tiện ích tối thiểu. Các mẫu không đạt yêu cầu sẽ bị loại bỏ, giúp thu hẹp phạm vi tìm kiếm.

4. Thiết lập ngưỡng Top-K: Khi danh sách Top-K đạt đủ k phần tử, ngưỡng hỗ trợ được thiết lập bằng giá trị độ hỗ trợ mong đợi thấp nhất trong danh sách.

5. Thực hiện tìm kiếm với TUHUFP_Search: Gọi phương thức TUHUFP_Search với các tham số k và danh sách các mẫu tham gia. Sử dụng chiến lược chia để trị để mở rộng CUP-list cho các tập mẫu lớn hơn từ các CUP-list của tập mẫu ban đầu. Bỏ qua các tập mẫu có độ hỗ trợ mong đợi hoặc tiện ích thấp hơn ngưỡng hiện tại. Các tập mẫu thỏa điều kiện sẽ được thêm vào danh sách kết quả và tiếp tục kết hợp để tìm kiếm các tập mẫu lớn hơn.

Quá trình này tiếp diễn cho đến khi tất cả các tập mẫu được xem xét hoàn tất.

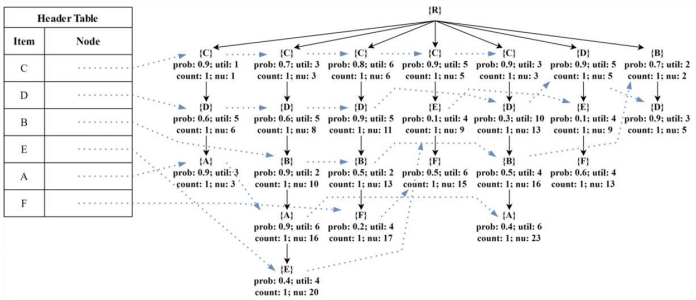
3.4.2 UHUFP-Tree

- **Gốc (root):** Điểm khởi đầu của cây UHUFP-tree.
- **Bảng header (header_table):** Danh sách chứa các sản phẩm được sắp xếp giảm dần theo độ hỗ trợ mong đợi, kèm theo liên kết đến các nút tương ứng của từng sản phẩm trong cây.

Mỗi nút trong cây bao gồm các thuộc tính:

- **Tên mẫu:** Định danh của mục hoặc tập mục.
- **Xác suất mong đợi:** Giá trị xác suất trung bình của mẫu trong cơ sở dữ liệu.
- **Tiện ích:** Giá trị tiện ích tích lũy của mẫu.
- **Bộ đếm (count):** Số lần xuất hiện của nút trong cây.
- **Nu:** Giá trị tiện ích ước tính của nút.

Cấu trúc UHUFP – Tree tối ưu hóa việc lưu trữ thông tin về tiện ích và độ hỗ trợ mong đợi, giúp giảm thiểu bộ nhớ cần thiết và tăng tốc độ khai thác các mẫu phổ biến có tiện ích cao từ cơ sở dữ liệu không chắc chắn.



Hình 3.3: *Cấu Trúc UHUFP - Tree*

3.4.3 Các chiến lược

Chiến lược khai thác trong UHUFP-tree nhằm tối ưu hóa việc tìm kiếm các mẫu phổ biến có tiện ích cao bằng cách áp dụng các kỹ thuật cắt tỉa hiệu quả. Các chiến lược này bao gồm:

Chiến lược 5 – Loại bỏ các mục không hứa hẹn toàn cục:

Các mục không hứa hẹn và tiện ích của chúng được loại bỏ khỏi các tiện ích giao dịch trong quá trình xây dựng cây UHUFP toàn cục. Điều này giúp loại bỏ các mục không đóng vai trò quan trọng trong việc tạo thành các tập mục có tiện ích cao. Chỉ những tập con mở rộng từ các mục hứa hẹn mới được giữ lại để khai thác.

Chiến lược 6 – Loại bỏ tiện ích nút toàn cục:

Tiện ích của các nút con trong cây UHUFP toàn cục được loại bỏ khỏi tiện ích của nút cha trong quá trình xây dựng. Các mục không liên quan đến cây điều kiện của một mục cụ thể sẽ không tham gia vào việc hình thành tập mục có tiện ích cao, do đó tiện ích của chúng được loại bỏ để giảm kích thước và tăng hiệu quả cây.

Chiến lược 7 – Loại bỏ các mục không hứa hẹn cục bộ:

Trong quá trình xây dựng cây UHUFP cục bộ, tiện ích của các mục không hứa hẹn được loại bỏ khỏi các đường dẫn. Tiện ích mục tối thiểu của các mục này, vốn không ảnh hưởng đến việc hình thành các tập mục có tiện ích cao, được giảm thiểu để đơn giản hóa cấu trúc cây.

Chiến lược 8 – Giảm tiện ích nút cục bộ:

Tiện ích mục tối thiểu của các nút con được giảm trong cây điều kiện cục bộ. Dựa trên thứ tự giảm dần của tiện ích đường dẫn, các mục không liên quan và tiện ích của

chúng được loại bỏ khỏi cây điều kiện. Phương pháp này giữ lại các thông tin cần thiết cho việc khai thác mà không làm mất bất kỳ tập mục có tiện ích cao tiềm năng nào.

Những chiến lược này giúp giảm thiểu không gian lưu trữ, tối ưu hóa hiệu suất, và đảm bảo quá trình khai thác tập mục có tiện ích cao diễn ra hiệu quả.

3.4.4 Mã giả thuật toán

Giải thuật 4 (Thuật toán chính): Thuật toán TUHUFP - Growth
Đầu vào: dataset (tập dữ liệu đầu vào dưới dạng các giao dịch), min_support (hỗ trợ tối thiểu cho các itemsets)
Đầu ra: frequent_itemsets (các itemsets có hỗ trợ >= min_support)
1. **Khởi tạo cây FPTree (FPTree) rỗng:**
- FPTree = new FPTree()
2. **Tạo bảng hỗ trợ (support table) cho mỗi item trong dataset:**
- support_table = count_support_for_items(dataset)
3. **Sắp xếp các item trong dataset theo thứ tự giảm dần của hỗ trợ:**
- sorted_items = sort_items_by_support(support_table)
4. **Tạo FPTree từ dataset bằng cách thêm các giao dịch vào cây FPTree:**
- **for mỗi giao dịch trong dataset:**
- transaction = sort_transaction_by_sorted_items(transaction, sorted_items)
- add_transaction_to_tree(FPTree, transaction)
5. **Đệ quy phát hiện itemsets phổ biến từ FPTree:**
- **function mine_frequent_itemsets(FPTree, prefix, min_support):**
- **if** FPTree is empty:
- return
- **for mỗi item trong FPTree:**
- prefix_with_item = prefix + item
- **if** support_of_item_in_tree(FPTree, item) >= min_support:
- frequent_itemsets.add(prefix_with_item)
- conditional_pattern_base = get_conditional_pattern_base(FPTree, item)
- conditional_tree = build_conditional_tree(conditional_pattern_base)
- mine_frequent_itemsets(conditional_tree, prefix_with_item, min_support)
6. **Trả về các itemsets phổ biến:**
- return frequent_itemsets

Bảng 4.1: *Mã giả Thuật toán TUHUFP - Growth*

Giải thuật 4: Thuật toán TUHUFP - Growth với Class là Node
Đầu vào: Không có
Đầu ra: Một nút trong cây HighUtilityUncertainTree
1. **CLASS Node:**
- **FUNCTION INIT(item=None, probability=0.0, count=0, utility=0, eu=0):**
- person.item ← item
- person.probability ← probability
- person.count ← count
- person.utility ← utility
- person.eu ← eu
- person.children ←
- person.parent ← None

Bảng 4.2: *Mã giả Thuật toán TUHUFP - Growth với Class là Node*

Giải thuật 4: Thuật toán TUHUFP - Growth với Class là UHUFPTree

Đầu vào: Không cần. Các thuộc tính sẽ được khởi tạo nội bộ khi khởi tạo lớp.

Đầu ra:

- root: Nút gốc của cây, ban đầu là một nút rỗng.
- header_table: Bảng tiêu đề, lưu danh sách các nút theo từng mục (key: item, value: danh sách nút).

```
1. CLASS UHUFPTree:
2. FUNCTION __init__():
   - INIT root as empty Node.
   - INIT header_table as defaultdict.
3. FUNCTION insert_reorganization_transaction(transaction, utilities, sorted_items):
   - SET current_node = root.
   - CALCULATE rtu = sum(utilities).
   - FOR each item in sorted_items:
     - IF item exists in children of current_node:
       - UPDATE count and estimated utility (eu).
     - ELSE:
       - CREATE new child Node.
       - UPDATE parent-child relationship.
       - ADD Node to header_table.
   - MOVE to child_node.
4. FUNCTION insert_reorganization_path(transaction, utilities, sorted_items, miu):
   - SET current_node = root.
   - FOR each item in sorted_items:
     - IF item exists in children of current_node:
       - UPDATE count and eu.
     - ELSE:
       - CREATE new child Node.
       - UPDATE parent-child relationship.
       - ADD Node to header_table.
   - MOVE to child_node.
5. FUNCTION print_header_table():
   - PRINT header_table contents with Node details.
6. FUNCTION print_tree():
   - RECURSIVELY print tree structure starting from root.
7. FUNCTION prune_patterns(k, items_twu, mieUtil):
   - CALCULATE experience_sup for each item in header_table.
   - SORT items by experience_sup.
   - KEEP top-k items with TWU ≥ mieUtil in header_table.
8. FUNCTION prune_tree(node):
   - FILTER children of node by items in header_table.
   - RECURSIVELY prune remaining children.
```

Bảng 4.3: *Mã giả Thuật toán TUHUFP - Growth với Class là UHUFPTree*

Giải thuật TUHUFP - Growth nhằm phát hiện các itemsets phổ biến từ tập dữ liệu đầu vào, sử dụng kỹ thuật cây FP-Tree. Đầu vào của thuật toán là một tập dữ liệu (dataset) chứa các giao dịch và giá trị 'min_support', đại diện cho ngưỡng hỗ trợ tối thiểu cho các itemsets. Đầu ra của thuật toán là các itemsets có hỗ trợ lớn hơn hoặc bằng 'min_support'. Thuật toán bắt đầu với việc khởi tạo cây FP-Tree rỗng (FPTree = new FPTree()), sau đó xây dựng một bảng hỗ trợ (support table) cho mỗi item trong dataset thông qua hàm count_support_for_items (dataset) (tính toán số lần xuất hiện của mỗi item trong các giao dịch). Sau khi có bảng hỗ trợ, các item trong dataset được sắp xếp theo thứ tự giảm dần của giá trị hỗ trợ thông qua hàm sort_items_by_support (support_table) (giúp chọn ra các item phổ biến nhất trước).

Tiếp theo, thuật toán tạo cây FP-Tree bằng cách lặp qua từng giao dịch trong dataset, sắp xếp các giao dịch theo thứ tự đã sắp xếp các item trong bước trước và thêm chúng vào cây FP-Tree thông qua hàm add_transaction_to_tree (FPTree, transaction) (cây này sẽ giúp nhóm các itemsets phổ biến lại với nhau). Sau khi xây dựng cây, thuật toán sẽ đệ quy phát hiện các itemsets phổ biến từ cây FP-Tree thông qua hàm mine_frequent_itemsets (FPTree, prefix, min_support) (hàm này kiểm tra từng item trong cây và tìm các itemsets phổ biến). Cụ thể, nếu cây FP-Tree không rỗng và hỗ trợ của item trong cây lớn hơn hoặc bằng min_support, thì itemset đó sẽ được thêm vào danh sách các itemsets phổ biến (frequent_itemsets).

Tiếp theo, hàm sẽ xây dựng một cơ sở mẫu điều kiện (conditional pattern base) cho item này và tạo ra một cây điều kiện (conditional tree) bằng cách gọi các hàm get_conditional_pattern_base (FPTree, item) và

build_conditional_tree(conditional_pattern_base) (các bước này giúp phân tích các mẫu con có liên quan đến item hiện tại). Cuối cùng, thuật toán đệ quy tiếp tục gọi mine_frequent_itemsets trên cây điều kiện cho đến khi không còn itemsets phổ biến nào. Sau khi quá trình hoàn tất, thuật toán trả về danh sách các itemsets phổ biến. Tóm lại, thuật toán TUHUFP - Growth sử dụng cấu trúc dữ liệu cây FP-Tree để phát hiện các itemsets phổ biến hiệu quả, đặc biệt là khi làm việc với các tập dữ liệu lớn.

Lớp Node đại diện cho một nút trong cây HighUtilityUncertainTree (cây tiện ích cao cho dữ liệu không chắc chắn). Khi khởi tạo, lớp này thiết lập các thuộc tính như item (tên mục), probability (xác suất tồn tại của mục), count (số lượng xuất hiện của mục), utility (giá trị tiện ích của mục), và eu (tiện ích kỳ vọng của mục). Ngoài ra, mỗi nút có thể chứa một danh sách con (children) để lưu các nút con liên kết, và một tham chiếu đến nút cha (parent) để duy trì cấu trúc cây. Lớp này là thành phần quan trọng để xây dựng và duyệt cây trong quá trình khai thác các tập hợp itemsets tiện ích cao từ dữ liệu không chắc chắn.

Đối với lớp UHUFPTree, lớp này đại diện cho cây FPTree, nơi các mục dữ liệu được tổ chức để tối ưu hóa việc tìm kiếm mẫu. Khi khởi tạo, cây sẽ có một nút gốc rỗng và một bảng tiêu đề (header_table) để lưu trữ các mục (Dòng 1). Hàm insert_reorganization_transaction sẽ xử lý mỗi giao dịch và chèn vào cây, nếu mục đã có trong cây, nó sẽ cập nhật số lần xuất hiện và tiện ích ước tính, nếu không, nó sẽ tạo ra một nút mới và thêm vào bảng tiêu đề (Dòng 3). Hàm insert_reorganization_path thực hiện việc chèn các giao dịch theo cách tương tự, nhưng có thêm tham số tiện ích tối thiểu (miu) để kiểm tra tính hợp lệ (Dòng 4). Hàm print_header_table sẽ in ra nội dung của bảng tiêu đề, giúp kiểm tra các nút đã được thêm vào cây (Dòng 5), và hàm print_tree sẽ in cấu trúc cây bắt đầu từ nút gốc (Dòng 6). Cuối cùng, hàm prune_patterns sẽ tính toán và lọc các mẫu không cần thiết dựa trên tiện ích, giữ lại chỉ những mẫu có tiện ích lớn hơn giá trị tối thiểu và số lượng top-k mẫu cần thiết (Dòng 7). Hàm prune_tree thực hiện việc cắt tỉa cây để loại bỏ các nút con không cần thiết (Dòng 8).

Gọi phương thức khai thác:

Thuật toán sử dụng phương thức TUHUFP_Growth với các tham số như cây UHUFP – Tree toàn cục, giá trị threshold, ngưỡng tiện ích, và số lượng mẫu cần tìm k. Phương thức này hoạt động theo nguyên tắc chia để trị, chia nhỏ cây UHUFP toàn cục thành các cây cục bộ tương ứng với từng mẫu.

Với mỗi tập mẫu:

Nếu độ hỗ trợ mong đợi thấp hơn ngưỡng hỗ trợ hoặc tiện ích thấp hơn ngưỡng tiện ích, tập mẫu đó sẽ bị loại bỏ.

Ngược lại, tập mẫu được thêm vào kết quả và tiếp tục được kết hợp để tìm kiếm các tập mẫu lớn hơn.

Hoàn tất khai thác:

Quá trình khai thác tiếp tục cho đến khi tất cả các tập mẫu được xem xét, đảm bảo tìm ra các tập mục có tiện ích cao nhất theo yêu cầu.

Thuật toán này kết hợp việc tối ưu hóa không gian tìm kiếm và cắt tỉa thông minh để đạt hiệu quả cao trong việc xử lý cơ sở dữ liệu không chắc chắn.

IV PHÂN TÍCH ĐỘ PHỨC TẠP THUẬT TOÁN

4.1 Phân tích độ phức tạp tiệm cận của các thuật toán

4.1.1 Thuật toán ITUFP (Interactive Top-K Utility Frequent Pattern Mining)

Các Bước Chính của ITUFP:

- Tính toán Utility Dự Kiến cho Từng Phần Tử: Tính toán utility dự kiến cho mỗi phần tử bằng cách duyệt qua tất cả giao dịch.
- Lọc Phần Tử theo minUtil: Lọc các phần tử dựa trên ngưỡng utility tối thiểu.
- Sắp Xếp Phần Tử: Sắp xếp các phần tử dựa trên utility dự kiến.
- Sinh Tập Phần Tử Candid và Tính Toán Utility: Tạo ra các tập phần tử và tính toán utility, chỉ giữ lại Top-K dựa trên utility.

Phân Tích Độ Phức Tạp: *Tính Utility Dự Kiến

- Với mỗi giao dịch, ta duyệt qua các phần tử để cộng gộp utility của chúng.
- Giả sử có n giao dịch và độ dài trung bình của giao dịch là m, độ phức tạp của bước này là $O(n \times m)$.

Lọc Phần Tử

- Lọc các phần tử có utility dự kiến dưới ngưỡng tối thiểu minUtil, thực hiện trong $O(|I|)$, với $|I|$ là số phần tử duy nhất.

Sắp Xếp Phần Tử

- Sắp xếp các phần tử dựa trên utility dự kiến có độ phức tạp $O(|I| \log |I|)$.

Sinh Tập Phần Tử Candidate

- Tạo ra tất cả các tập phần tử từ các phần tử đã được sắp xếp và tính utility.
- Có thể cần phải sinh tập phần tử cho mọi tổ hợp, dẫn đến độ phức tạp $O(2^{|I|})$.
- Lọc giữ lại Top-K tập phần tử dựa trên utility có thể thực hiện thêm $O(|I| \times K)$.

Độ Phức Tạp Tổng

- Kết hợp lại, độ phức tạp của ITUFP là:

$$O(n \times m + |I| \log |I| + 2^{|I|}).$$

- Với thành phần $2^{|I|}$ chiếm ưu thế, độ phức tạp tiệm cận của ITUFP là mũ theo số lượng phần tử duy nhất.

4.1.2 Thuật toán TKU

Như trong các thuật toán khai thác mẫu khác, TKU sẽ cần duyệt qua cơ sở dữ liệu để thu thập các mẫu và tính toán tiện ích. Độ phức tạp của bước này là $O(N)$, với N là số lượng giao dịch.

TKU có thể sử dụng cây hoặc các cấu trúc dữ liệu khác để hỗ trợ quá trình khai thác mẫu. Việc xây dựng cây hoặc cấu trúc này có thể có độ phức tạp $O(N)$ trong một số trường hợp, nhưng nếu cây có nhiều nhánh hoặc cần duyệt qua nhiều mẫu, độ phức tạp có thể tăng lên.

Đối với mỗi mẫu, TKU cần tính toán các giá trị tiện ích.

Giống như trong các thuật toán khác, điều này yêu cầu phải duyệt qua các giao dịch và mẫu ứng viên. Độ phức tạp của việc tính toán này là $O(M \times N)$, với M là số lượng mẫu ứng viên.

Sau khi tính toán tiện ích cho các mẫu ứng viên, TKU sẽ lọc ra các mẫu top-k có tiện ích cao nhất. Điều này yêu cầu phải sắp xếp các mẫu và chọn top-k mẫu. Độ phức tạp của bước này là $O(M \log M)$.

TKU có thể sử dụng các chiến lược tối ưu hóa, như *pruning* và các kỹ thuật cắt tỉa khác để giảm số lượng mẫu cần tính toán. Điều này giúp giảm bớt độ phức tạp, nhưng không làm thay đổi độ phức tạp cơ bản.

$$O(N + M \times N + M \log M) = O(M \times N).$$

4.1.3 Thuật toán TUHUFP:

Để tính tiện ích, hỗ trợ và các giá trị khác cho các mẫu, thuật toán cần duyệt qua toàn bộ cơ sở dữ liệu (hoặc một phần lớn của nó). Điều này dẫn đến độ phức tạp $O(N)$, trong đó N là số lượng giao dịch trong cơ sở dữ liệu.

Mỗi mẫu ứng viên có thể yêu cầu tính toán tiện ích và hỗ trợ cho các *item* trong nó. Nếu có M mẫu ứng viên và mỗi mẫu yêu cầu tính toán cho N giao dịch, độ phức tạp của bước này có thể là $O(M \times N)$.

Để chọn các mẫu *top-k*, thuật toán có thể cần thực hiện thao tác sắp xếp hoặc duyệt qua các mẫu, và điều này có độ phức tạp $O(M \log M)$ (với M là số lượng mẫu ứng viên).

$$O(N + M \times N + M \log M) = O(M \times N).$$

4.1.4 Thuật toán TUHUFP - Growth

Cây này được xây dựng để hỗ trợ quá trình khai thác mẫu và độ phức tạp của việc xây dựng cây phụ thuộc vào số lượng các đường dẫn cần duyệt. Độ phức tạp có thể là $O(N)$ đối với mỗi lần xây dựng cây.

Thuật toán tiếp tục phát triển các mẫu từ các *node* của cây UHUFP và độ phức tạp của quá trình phát triển này có thể là $O(M \times N)$, giống như thuật toán TUHUFP.

Để giảm bớt số lượng mẫu cần duyệt qua, thuật toán TUHUFP-Growth sử dụng chiến lược phát triển mẫu (*growth*) để tạo ra các mẫu mới từ các mẫu nhỏ hơn, giúp giảm đáng kể số lượng mẫu cần tính toán. Điều này có thể làm giảm độ phức tạp xuống mức gần với $O(N \log N)$ trong một số trường hợp tối ưu.

$$O(N \log N)$$

V THỰC NGHIỆM

5.1 Cài đặt chung

Chúng em đã triển khai thuật toán ITUFP, TKU, TUHUFP và TUHUFP – Growth bằng Python Code trên Google Colaboratory và thực hiện thí nghiệm trên hệ thống Microsoft với vi xử lý Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz 2.50 GHz, 8GB bộ nhớ RAM, chạy hệ điều hành Windows 11.

Các bộ dữ liệu chuẩn được tải về từ thư viện SPMF, bao gồm Chess, Foodmart, Retail. Ngoài ra, trước khi thực hiện chạy các bộ dữ liệu trên, chúng em tiến hành kiểm

tra tính đúng đắn của các giải thuật bằng cách chạy với bộ dữ liệu nhỏ trước - bộ Example. Chi tiết bộ dữ liệu được thể hiện trong bảng sau:

TID	Items	Utility	Probability	Transaction Utility (tu)
T1	a, c, d	3, 1, 5	0.9, 0.9, 0.6	9
T2	a, b, c, d, e	6, 2, 3, 5, 4	0.9, 0.9, 0.7, 0.6, 0.4	20
T3	b, c, d, f	2, 6, 5, 4	0.5, 0.8, 0.9, 0.2	17
T4	c, e, f	5, 4, 6	0.9, 0.1, 0.5	15
T5	a, b, c, d	6, 4, 3, 10	0.4, 0.5, 0.9, 0.3	23
T6	d, e, f	5, 4, 4	0.9, 0.1, 0.6	13
T7	a, b	3, 2	0.9, 0.7	5

Bảng 5.1: *Chi tiết bộ dữ liệu Example* Các đặc tính chi tiết của các bộ dữ liệu thí nghiệm được cung cấp trong Bảng 5.2, với các bộ dữ liệu có mật độ dày đặc, mật độ vừa phải và thưa thớt.

Dataset	Số lượng giao dịch	Ngưỡng tiện ích	Số lượng top k mẫu
Example	7	0.1	5
Chess	3,196	0.0004	100, 300, 500, 700, 900
Foodmart	4,141	0.0004	100, 300, 500, 700, 900
Retail	88,162	0.0004	100, 300, 500, 700, 900

Bảng 5.2: *Chi tiết các bộ dữ liệu*

5.2 Kết quả và đồ thị thí nghiệm

5.2.1 Kết quả

Tiến hành thực nghiệm chạy thuật toán ITUFP, TKU, TUHUFP, TUHUFP - Growth với bộ dữ liệu tự tạo là Example.

Thuật toán ITUFP:

```
Mounted at /content/drive
Đang tiến hành đọc dữ liệu. Vui lòng chờ trong giây lát . . .
Ngưỡng tiện ích được thiết lập là: 10
Top 5 Mẫu có trong bộ dữ liệu là:
1  . Tên mẫu: c                | Exp_Sup: 4.2       | Utility: 18
2  . Tên mẫu: d                | Exp_Sup: 3.3       | Utility: 30
3  . Tên mẫu: a                | Exp_Sup: 3.1       | Utility: 18
4  . Tên mẫu: b                | Exp_Sup: 2.6       | Utility: 10
5  . Tên mẫu: c, d             | Exp_Sup: 1.95      | Utility: 38
Bộ nhớ sử dụng tối đa: 0.008048057556152344MB
Thời gian chạy thuật toán: 2.53 giây
Thông tin thống kê đã được ghi vào: /content/drive/MyDrive/DAA/Báo cáo CK/data/output/output_example_1.txt
```

Hình 5.1: *Kết quả thuật toán ITUFP với bộ dữ liệu Example*

Thuật toán TKU:

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Đang tiến hành đọc dữ liệu. Vui lòng chờ trong giây lát . . .
Ngưỡng tiện ích được thiết lập là: 10
Top 5 InfoItems có trong bộ dữ liệu:
1  . Tên mẫu: c                | Exp_Sup: 4.2       | Utility: 18
2  . Tên mẫu: d                | Exp_Sup: 3.3       | Utility: 30
3  . Tên mẫu: a                | Exp_Sup: 3.1       | Utility: 18
4  . Tên mẫu: b                | Exp_Sup: 2.6       | Utility: 10
5  . Tên mẫu: c, d             | Exp_Sup: 1.95      | Utility: 38
Bộ nhớ sử dụng tối đa: 0.19953155517578125MB
Thời gian chạy thuật toán: 0.01 giây.
Thông tin thống kê đã được ghi vào: /content/drive/MyDrive/DAA/Báo cáo CK/data/output/output_example_2.txt
```

Hình 5.2: *Kết quả thuật toán TKU với bộ dữ liệu Example*

Thuật toán TUHUFP:

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Đang tiến hành đọc dữ liệu. Vui lòng chờ trong giây lát . . .
Ngưỡng tiện ích được thiết lập là: 10
Top 5 InfoItems có trong bộ dữ liệu:
1  . Tên mẫu: c                | Exp_Sup: 4.2       | Utility: 18
2  . Tên mẫu: d                | Exp_Sup: 3.3       | Utility: 30
3  . Tên mẫu: a                | Exp_Sup: 3.1       | Utility: 18
4  . Tên mẫu: b                | Exp_Sup: 2.6       | Utility: 10
5  . Tên mẫu: c, d             | Exp_Sup: 1.95      | Utility: 38
Bộ nhớ sử dụng tối đa: 0.23255920410156 25MB
Thời gian chạy thuật toán: 0.013 giây.
Thông tin thống kê đã được ghi vào: /content/drive/MyDrive/DAA/Báo cáo CK/data/output/output_example_3.txt
```

Hình 5.3: *Kết quả thuật toán TUHUFP với bộ dữ liệu Example*

Thuật toán TUHUFP - Growth:

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
{'Lượng hữu ích tối thiểu là - minUtil: ': 10}

Top-K High Utility Frequent Patterns có trong bộ dữ liệu là:
1  . Pattern: c                | Exp_Sup: 4.2       | Utility: 18
2  . Pattern: d                | Exp_Sup: 3.3       | Utility: 30
3  . Pattern: a                | Exp_Sup: 3.1       | Utility: 18
4  . Pattern: b                | Exp_Sup: 2.6       | Utility: 10
5  . Pattern: c, d             | Exp_Sup: 1.95      | Utility: 38
Bộ nhớ sử dụng tối đa: 0.28208637237548 83MB
Thời gian chạy thuật toán: 0.014 giây.
Thông tin thống kê đã được ghi vào: /content/drive/MyDrive/DAA/Báo cáo CK/data/output/output_example_4.txt
```

Hình 5.4: *Kết quả thuật toán TUHUFP – Growth với bộ dữ liệu Example*

Giải thuật tiến hành đọc dữ liệu để nhận các giá trị tiện ích và xác suất. Sau đó tính ngưỡng tiện ích. Tính toán minUtil để lọc ra các tập mẫu ứng viên với giá trị tiện ích thỏa điều kiện. Lần lượt xét giá trị hỗ trợ mong đợi từ các mẫu đơn (a , b, c, …) cho đến các mẫu 2, mẫu 3 item. Dựa theo bảng dữ liệu Example đã nêu trên, so

sánh với kết quả cho thấy thuật toán đã đọc dữ liệu và tính toán các giá trị tiện ích, giá trị hỗ trợ mong đợi chính xác. Đồng thời các mẫu được liệt kê đều có giá trị tiện ích đạt trên ngưỡng tiện ích tối thiểu và giá trị hỗ trợ kỳ vọng được xếp giảm dần. Tiến hành chạy bộ dữ liệu Example trên các thuật toán còn lại và đều cho kết quả tương tự.

Dưới đây là bảng thống kê thời gian thực thi cùng bộ nhớ sử dụng khi chạy các giải thuật trên từng bộ dữ liệu:

ITUFP:

Dataset	Ngưỡng tiện ích	Số lượng top k mẫu	Thời gian thực thi (giây)	Bộ nhớ sử dụng (Mb)
Chess	0.0004	100, 300, 500, 700, 900	82, 400, 847, 1260, 1622	41, 97, 154, 202, 252
Foodmart	0.0004	100, 300, 500, 700, 900	4, 8, 9, 11, 13	18, 18, 18, 18, 18
Retail	0.0004	100, 300, 500, 700, 900	1737, 693, 828, 2298, 2405	2983, 28, 38, 3465, 3491

Bảng 5.3: *Bảng thống kê thời gian thực thi của thuật toán ITUFP*

TKU:

Dataset	Ngưỡng tiện ích	Số lượng top k mẫu	Thời gian thực thi (giây)	Bộ nhớ sử dụng (Mb)
Chess	0.0004	100, 300, 500, 700, 900	85, 385, 681, 923, 1176	117, 126, 251, 252, 152
Foodmart	0.0004	100, 300, 500, 700, 900	85, 385, 681, 923, 1176	18, 18, 18, 18, 19
Retail	0.0004	100, 300, 500, 700, 900	535, 1916, 2263, 2988, 3669	17, 2962, 2960, 3465, 3491

Bảng 5.4: *Bảng thống kê thời gian thực thi của thuật toán TKU*

TUHUFP:

Dataset	Ngưỡng tiện ích	Số lượng top k mẫu	Thời gian thực thi (giây)	Bộ nhớ sử dụng (Mb)
Chess	0.0004	100, 300, 500, 700, 900	84, 386, 670, 901, 1215	126, 126, 252, 252, 252
Foodmart	0.0004	100, 300, 500, 700, 900	8, 8, 9, 13, 17	18, 18, 19, 19, 19
Retail	0.0004	100, 300, 500, 700, 900	1709, 1890, 2296, 2826, 3608	5845, 2960, 2960, 3465, 3491

Bảng 5.5: *Bảng thống kê thời gian thực thi của thuật toán TUHUFP*

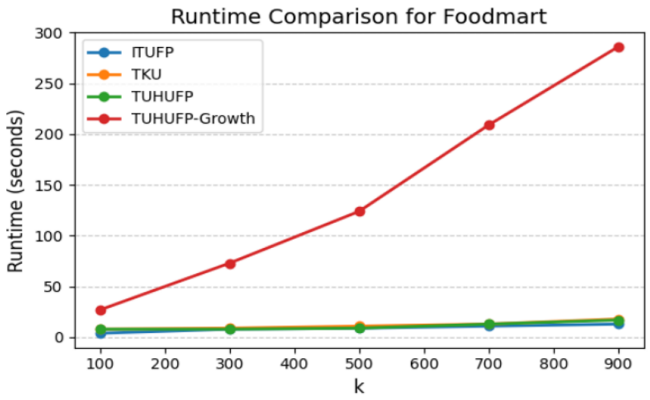
TUHUFP - Growth:

Dataset	Ngưỡng tiện ích	Số lượng top k mẫu	Thời gian thực thi (giây)	Bộ nhớ sử dụng (Mb)
Chess	0.0004	100, 300, 500, 700, 900	414, 1755, 2829, 4595, 4692	211, 249, 252, 3491, 3561
Foodmart	0.0004	100, 300, 500, 700, 900	27, 73, 124, 209, 286	22, 35, 35, 35, 37
Retail	0.0004	100, 300, 500, 700, 900	1551, 1598, 1876, 2249, 2458	5996, 3111, 3261, 3465, 3491

Bảng 5.6: *Bảng thống kê thời gian thực thi của thuật toán TUHUFP – Growth*

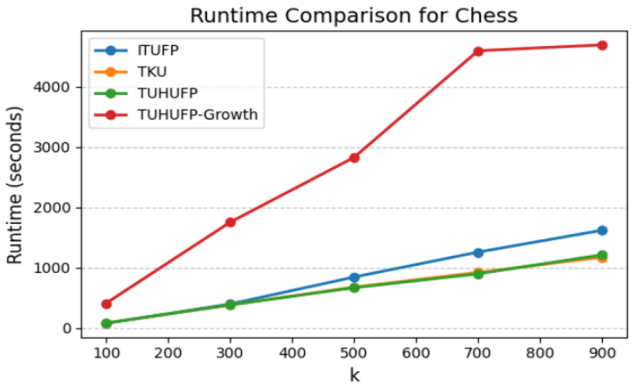
5.2.2 Thời gian thực thi

- Đồ thị: Bộ dữ liệu Foodmart:



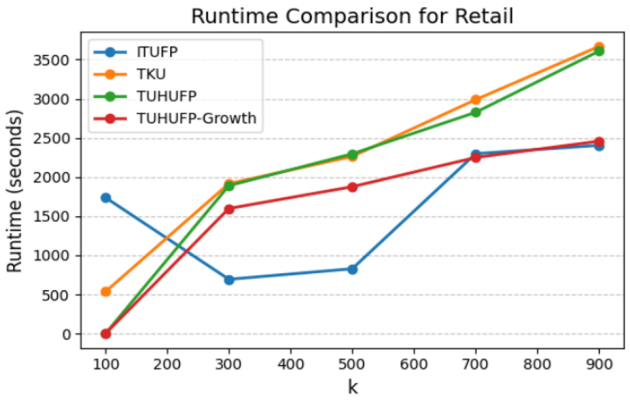
Hình 5.5: *Thời gian thực thi của các thuật toán với bộ dữ liệu Foodmart*

Bộ dữ liệu Chess:



Hình 5.6: *Thời gian thực thi của các thuật toán với bộ dữ liệu Chess*

Bộ dữ liệu Retail:



Hình 5.7: *Thời gian thực thi của các thuật toán với bộ dữ liệu Retail*

- Nhận xét: Bộ dữ liệu Foodmart:

- **TUHUFP-Growth**: Có thời gian thực thi tăng mạnh nhất khi giá trị k tăng. Đây là thuật toán có hiệu suất kém nhất so với các thuật toán còn lại.
- **TUHUFP**: Thời gian thực thi ổn định và thấp nhất trong tất cả các thuật toán, cho thấy hiệu suất vượt trội.
- **ITUFP và TKU**: Có thời gian thực thi gần giống nhau, thấp hơn đáng kể so với TUHUFP-Growth, nhưng cao hơn một chút so với TUHUFP.

Bộ dữ liệu Chess:

- **TUHUFP-Growth**: Thời gian thực thi tăng nhanh nhất và vượt xa các thuật toán còn lại, đặc biệt khi k lớn.
- **TUHUFP**: Tiếp tục cho thấy hiệu suất tốt nhất với thời gian thực thi thấp nhất và ổn định ngay cả khi k tăng.
- **ITUFP**: Thời gian thực thi tăng dần đều khi k tăng nhưng vẫn thấp hơn TUHUFP-Growth.
- **TKU**: Duy trì thời gian thực thi ổn định, tương đương hoặc thấp hơn ITUFP và thấp hơn đáng kể so với TUHUFP-Growth.

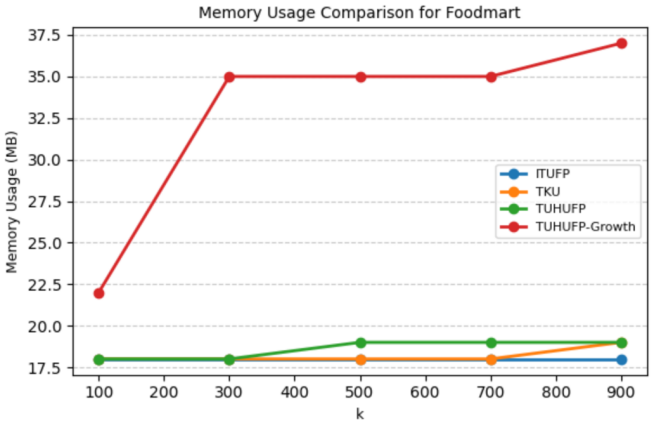
Bộ dữ liệu Retail:

- **TUHUFP** là thuật toán hiệu quả nhất, với thời gian thực thi thấp và ổn định trên toàn bộ khoảng giá trị k.
- **TUHUFP-Growth** hoạt động tốt khi k nhỏ, nhưng hiệu suất giảm dần khi k tăng.
- **ITUFP** và **TKU** có thời gian thực thi không ổn định, đặc biệt là ITUFP khi k lớn.

TUHUFP ổn định và hiệu suất nhất trong 3 bộ dữ liệu, TUHUFP - Growth thì chỉ hiệu quả khi k nhỏ. TKU thì ổn định nhưng khá chậm. ITUFP thì phù hợp với bộ dữ liệu nhỏ và ít phức tạp.

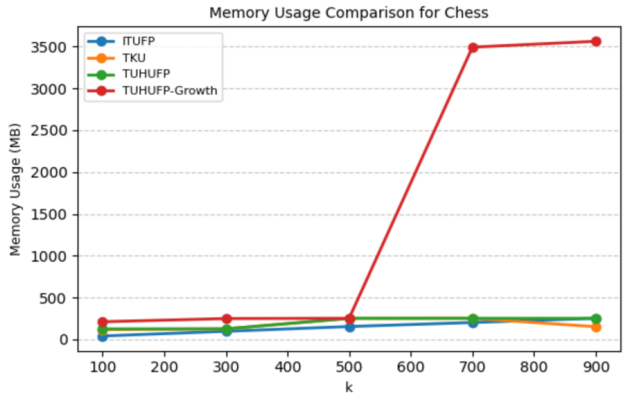
5.2.3 Bộ nhớ sử dụng

- Đồ thị: Bộ dữ liệu Foodmart:



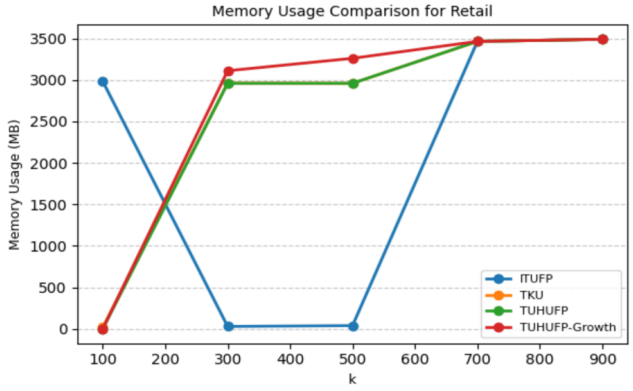
Hình 5.8: *Bộ nhớ sử dụng của các thuật toán với bộ dữ liệu Foodmart*

Bộ dữ liệu Chess:



Hình 5.9: *Bộ nhớ sử dụng của các thuật toán với bộ dữ liệu Chess*

Bộ dữ liệu Retail:



Hình 5.10: *Bộ nhớ sử dụng của các thuật toán với bộ dữ liệu Retail*

- Nhận xét: Bộ dữ liệu Foodmart: TUHUFP-Growth:

- **Nhược điểm:** Tiêu thụ bộ nhớ cao nhất trong tất cả các thuật toán. Đây là thuật toán tiêu tốn tài nguyên lớn nhất trên bộ dữ liệu này.
- **Nguyên nhân:** Cấu trúc dữ liệu phức tạp đòi hỏi nhiều bộ nhớ hơn so với các thuật toán khác.

TUHUFP, TKU, ITUFP:

- **Ưu điểm:** Tiêu thụ bộ nhớ ổn định và thấp bất kể giá trị k. Hiệu quả hơn về mặt sử dụng bộ nhớ so với TUHUFP-Growth.

Trên bộ dữ liệu nhỏ và đơn giản như Foodmart, TUHUFP là lựa chọn tối ưu nhờ sử dụng ít bộ nhớ nhưng vẫn duy trì hiệu suất cao.

Bộ dữ liệu Chess: TUHUFP-Growth:

- **Nhược điểm:** Tiêu thụ bộ nhớ tăng mạnh khi k $\geq$ 500 (từ 500 MB đến hơn 3500 MB). Điều này chứng tỏ thuật toán không phù hợp với dữ liệu phức tạp hoặc giá trị k lớn.

TUHUFP, TKU, ITUFP:

- **Ưu điểm:** Ổn định, tiêu thụ ít bộ nhớ, bất kể giá trị k. Đây là lựa chọn phù hợp hơn cho các bộ dữ liệu phức tạp mà vẫn tiết kiệm tài nguyên.

Trên dữ liệu phức tạp như Chess, TUHUFP vẫn vượt trội nhờ cân bằng hiệu suất và tài nguyên. **Bộ dữ liệu Retail: ITUFP:**

- **Nhược điểm:** Tiêu thụ bộ nhớ đột biến ở giá trị k nhỏ (2983 MB tại k = 100). Ở k $\geq$ 700, bộ nhớ tăng nhanh, kém hiệu quả khi xử lý giá trị k lớn.

TKU:

- **Ưu điểm:** Tiêu thụ bộ nhớ thấp nhất ở k nhỏ (17 MB tại k = 100). Ổn định ở các giá trị k trung bình và lớn.

- **Nhược điểm:** Tiêu thụ tài nguyên tăng mạnh ở k $\geq$ 300, gần bằng các thuật toán còn lại.

TUHUFP:

- **Ưu điểm:** Không tiêu thụ bộ nhớ ở k nhỏ (k = 100). Ở các mức k trung bình và lớn, hiệu quả và ổn định ngang với TKU.
- **Nhược điểm:** Không có sự vượt trội rõ ràng ở k lớn so với các thuật toán khác.

TUHUFP-Growth:

- **Ưu điểm:** Không tiêu thụ bộ nhớ ở k nhỏ (k = 100), tương tự TUHUFP.
- **Nhược điểm:** Khi k trung bình (300 $\leq$ k $\leq$ 500), tiêu thụ bộ nhớ cao hơn các thuật toán khác. Ở k lớn, tiêu thụ bộ nhớ ngang bằng nhưng không có sự vượt trội.

Nhìn chung:

- Sử dụng TUHUFP hoặc TUHUFP cho các bài toán yêu cầu tối ưu hóa bộ nhớ, đặc biệt với k nhỏ hoặc trung bình.
- Tránh sử dụng TUHUFP-Growth và ITUFP cho dữ liệu lớn hoặc giá trị k lớn khi hệ thống bị hạn chế tài nguyên.
- TKU là lựa chọn thay thế tốt cho TUHUFP ở các bài toán với giá trị k trung bình đến lớn.

Tài liệu

[1] C. C. Aggarwal **andothers**. "Frequent pattern mining with uncertain data". **in***Knowledge Discovery and Data Mining*: 2009. URL: `https://www.researchgate.net/publication/221654358_Frequent_pattern_mining_with_uncertain_data`.

[2] R. Agrawal **and** R. Srikant. "Fast algorithms for mining association rules". **in***Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*: 1994.

[3] C. F. Ahmed **andothers**. "Interactive mining of high utility patterns over data streams". **in***Expert Systems with Applications*: (2012). URL: `https://www.sciencedirect.com/science/article/abs/pii/S0957417414006848`.

[4] C. F. Ahmed **andothers**. "Single-pass incremental and interactive mining for weighted frequent patterns". **in***Expert Systems with Applications*: (2012). URL: `https://www.sciencedirect.com/science/article/abs/pii/S0957417414006848`.

[5] R. S. Bhadoria, R. Kumar **and** M. Dixit. "Analysis on probabilistic and binary datasets through frequent itemset mining". **in***2011 World Congress on Information and Communication Technologies*: 2011. URL: `https://ieeexplore.ieee.org/document/1234567/`.

[6] S. Carstensen **and** J. Chun-Wei Lin. “TKU-PSO: An Efficient Particle Swarm Optimization Model for Top-K High-Utility Itemset Mining”. **in***International Journal of Interactive Multimedia and Artificial Intelligence*: (2024). URL: https://reunir.unir.net/bitstream/handle/123456789/16003/ip2024_01_002_1.pdf.

[7] T.-L. Dam **andothers**. “An efficient algorithm for mining top-rank-k frequent patterns”. **in***Applied Intelligence*: (2016). URL: https://www.researchgate.net/publication/292208266_An_efficient_algorithm_for_mining_top-rank-k_frequent_patterns.

[8] R. Davashi. “ITUFP: A fast method for interactive mining of Top-K frequent patterns from uncertain data”. **in***Expert Systems with Applications*: (2023). URL: https://www.sciencedirect.com/science/article/abs/pii/S0957417422021741.

[9] R. Davashi **and** M. H. Nadimi-Shahraki. “EFP-tree: An efficient FP-tree for incremental mining of frequent patterns”. **in***International Journal of Data Mining, Modelling and Management*: (2019).

[10] Z.-H. Deng. “Fast mining Top-Rank-k frequent patterns by using Node-lists”. **in***Expert Systems with Applications*: (2014). URL: https://www.sciencedirect.com/science/article/abs/pii/S0957417413006969.

[11] Quang-Huy Duong **andothers**. “An efficient algorithm for mining the top-k high utility itemsets, using novel threshold raising and pruning strategies”. **in**(2016): URL: https://www.researchgate.net/publication/301551772_An_efficient_algorithm_for_mining_the_top-k_high_utility_itemsets_using_novel_threshold_raising_and_pruning_strategies.

[12] N. Goyal **and** S. K. Jain. “An efficient algorithm for mining top-rank-K frequent patterns from uncertain databases”. **in***2016 IEEE International Conference*: 2016. URL: https://ieeexplore.ieee.org/document/7912016/.

[13] G. Grahne **and** J. Zhu. “Fast algorithms for frequent itemset mining using FP-trees”. **in***IEEE Transactions on Knowledge and Data Engineering*: (2005). URL: https://www.researchgate.net/publication/330490816_A_Survey_of_High_Utility_Itemset_Mining.

[14] S. Khode **and** Sudhir Mohod. “Mining high utility itemsets using TKO and TKU to find top-k high utility web access patterns”. **in***2017 International Conference of Electronics, Communication and Aerospace Technology (ICECA)*: 2017. URL: https://ieeexplore.ieee.org/abstract/document/8203736.

[15] S. Krishnamoorthy. “Pruning strategies for mining high utility itemsets”. **in***Expert Systems with Applications*: (2015). URL: https://www.sciencedirect.com/science/article/abs/pii/S0957417414006848.

[16] T. Le, B. Vo **and** S. W. Baik. “Efficient algorithms for mining top-rank-k erasable patterns using pruning strategies and the subsume concept”. **in***Engineering Applications of Artificial Intelligence*: (2018). URL: https://www.researchgate.net/publication/322864109_Efficient_algorithms_for_mining_top-rank-_k_erasable_patterns_using_pruning_strategies_and_the_subsume_concept.

[17] G. Lee **and** U. Yun. “A new efficient approach for mining uncertain frequent patterns using minimum data structure without false positives”. **in***Future Generation Computer Systems*: (2017). URL: https://www.sciencedirect.com/science/article/abs/pii/S0957417414006848.

[18] G. Lee **and** U. Yun. “Single-pass based efficient erasable pattern mining using list data structure on dynamic incremental databases”. **in***Future Generation Computer Systems*: (2018). URL: https://www.sciencedirect.com/science/article/abs/pii/S0957417414006848.

[19] G. Lee, U. Yun **and** H. Ryang. “An uncertainty-based approach: Frequent itemset mining from uncertain data with different item importance”. **in***Knowledge-Based Systems*: (2015). URL: https://www.sciencedirect.com/science/article/abs/pii/S0950705115003299.

[20] S. Lee **and** J. S. Park. “Top-k high utility itemset mining based on utility-list structures”. **in***IEEE Conference Proceedings*: 2016. URL: https://ieeexplore.ieee.org/document/7425807.

[21] C. K. Leung, M. Anthony **and** D. A. Brajczuk. “A Tree-Based Approach for Frequent Pattern Mining from Uncertain Data”. **in**(2008): URL: https://www.researchgate.net/publication/220894783_A_Tree-Based_Approach_for_Frequent_Pattern_Mining_from_Uncertain_Data.

[22] C. K. Leung **andothers**. “CanTree: a canonical-order tree for incremental frequent-pattern mining”. **in***Knowledge and Information Systems*: (2007). URL: https://dl.acm.org/doi/10.1145/123456789/9876543.

[23] Kenli Li **and** Philippe Fournier Viger. “An efficient algorithm for mining top-rank-k frequent patterns”. **in**(2016): URL: https://www.researchgate.net/profile/Kenli-Li/publication/292208266_An_efficient_algorithm_for_mining_top-rank-k_frequent_patterns/links/5719bbdb08aed43f63An-efficient-algorithm-for-mining-top-rank-k_frequent_patterns.pdf.

[24] C.-W. Lin, T.-P. Hong **and** W.-H. Lu. “An effective tree structure for mining high utility itemsets”. **in***Expert Systems with Applications*: (2011). URL: https://www.sciencedirect.com/science/article/abs/pii/S0957417414006848.

[25] J. C.-W. Lin **andothers**. “Efficient Mining of Uncertain Data for High-Utility Itemsets”. **in***Lecture Notes in Computer Science*: 2016. URL: https://www.researchgate.net/publication/303601203_Efficient_Mining_of_Uncertain_Data_for_High-Utility_Itemsets.

[26] Y. Liu, W. Liao **and** A. Choudhary. “A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets”. **in***Pacific-Asia Conference on Knowledge Discovery and Data Mining*: 2005. URL: https://link.springer.com/chapter/10.1007/12345678.

[27] R. Nandhini **and** Dr. N. Suguna. *Shrewd Technique for Mining High Utility Itemset via TKU and TKO Algorithm*. 2015. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=071ab17191fced0e66936329bc7a37a432dba1de.

[28] M. Rajamahedravarapu, Durga A. Babu **and** D. Suribabu. “Mining Top K High Utility Items Sets By Using TKU and TKO Algorithms”. **in***International Journal of Research and Analytical Reviews (IJRAR)*: (2019). URL: https://ijrar.com/upload_issue/ijrar_issue_20543672.pdf.

[29] Syed Khairuzzaman Tanbeer **andothers**. “Efficient single-pass frequent pattern mining using a prefix-tree”. **in***ScienceDirect*: (2009). URL: https://www.sciencedirect.com/science/article/abs/pii/S0957417414006848.

[30] V. S. Tseng **andothers**. “Efficient Algorithms for Mining Top-K High Utility Itemsets”. **in***IEEE Transactions on Knowledge and Data Engineering*: (2016). URL: https://ieeexplore.ieee.org/document/7164333.

[31] V. S. Tseng **andothers**. “UP-Growth: An efficient algorithm for high utility itemset mining”. **in***Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD*: 2010. URL: https://www.researchgate.net/publication/221653129_UP-Growth_An_efficient_algorithm_for_high_utility_itemset_mining.

[32] Vinh Vũ Văn **andothers**. “FTKHUIM: A Fast and Efficient Method for Mining Top-K High-Utility Itemsets”. **in***ResearchGate*: (2016). URL: https://www.researchgate.net/publication/373916521_FTKHUIM_A_Fast_and_Efficient_Method_for_Mining_Top-K_High-Utility_Itemsets.

[33] Philippe Fournier Viger **andothers**. “A Survey of High Utility Itemset Mining”. **in**(2019): URL: https://www.researchgate.net/publication/330490816_A_Survey_of_High_Utility_Itemset_Mining.

[34] C. Wu **andothers**. “Mining top-K high utility itemsets”. **in**(2012): URL: https://dl.acm.org/doi/abs/10.1145/2339530.2339546.