So Sánh Các Phương Pháp Khai Thác Tiện Ích Cao Trên Tập Dữ Liệu Không Chắc Chắn

Chung Thái Kiệt
Ton Duc Thang University
Quan 7, Ho Chi Minh
52200140

Giản Hoàng Huy
Ton Duc Thang University
Quan 7, Ho Chi Minh
52200147

***Tóm tắt*—Giải thuật đóng vai trò quan trọng trong trong việc phân tích, khai phá dữ liệu và các bài toán tối ưu, giải thuật được ứng dụng trong hầu hết các lĩnh vực như khoa học máy tính, học máy, và lý thuyết đồ thị. Nghiên cứu này tập trung vào việc phát triển và cải tiến các giải thuật một cách hiệu quả để giải quyết các bài toán phức tạp nhằm tối ưu thời gian và không gian tính toán. Các phương pháp truyền thống như giải thuật tham lam, tìm kiếm theo chiều rộng, tìm kiếm theo chiều sâu đã chứng minh được tính hiệu quả trong nhiều tình huống, tuy nhiên những phương pháp này vẫn còn hạn chế khi đối mặt với các vấn đề có không gian tìm kiếm lớn hoặc dữ liệu không chắc chắn. Trong nghiên cứu này, chúng tôi đề xuất một giải thuật mới kết hợp giữa các kỹ thuật tối ưu hóa hiện đại như thuật toán di truyền và tìm kiếm nhánh cắt. Kết quả thực nghiệm cho thấy giải thuật được đề xuất vượt trội về mặt thời gian thực thi và độ chính xác so với các giải thuật truyền thống trong các bài toán thực tế. Chúng tôi cũng phân tích các yếu tố ảnh hưởng đến hiệu suất của giải thuật và đề xuất các hướng nghiên cứu tiếp theo nhằm cải thiện khả năng mở rộng và ứng dụng của giải thuật trong các bài toán phức tạp hơn.**

***Từ khóa*—Khai thác tập tiện ích cao, ngưỡng hữu ích tối thiểu, độ hữu ích, CSDL giao dịch, giải thuật di truyền, thuật toán ngắt ngưỡng.**

1. Giới thiệu

Khai thác dữ liệu, một lĩnh vực quan trọng của trí tuệ nhân tạo (AI), tập trung vào việc phân tích và khai phá các thông tin hữu ích từ dữ liệu lớn, đặc biệt trong bối cảnh dữ liệu ngày càng bùng nổ. Quá trình này, còn được gọi là khám phá tri thức trong cơ sở dữ liệu, nhằm tạo ra các tri thức hợp lệ, dễ hiểu, và có tính ứng dụng cao.

Một số phương pháp phổ biến trong khai thác dữ liệu bao có thể kể đến như luật kết hợp, phân loại, và gom cụm. Tuy nhiên, thách thức lớn nằm ở việc khai thác các mẫu thường xuyên hàng đầu (top-K frequent patterns) trong cơ sở dữ liệu không chắc chắn và các mẫu có tiện ích cao nhất (top-K high utility patterns).

Khác với mẫu thường xuyên, mẫu có tiện ích cao không chỉ xét đến tần suất xuất hiện mà còn đánh giá mức độ ưu tiên hay các yếu tố giá trị như tiền tệ. Nghiên cứu này tập trung phát triển các phương pháp xử lý dữ liệu không chắc chắn và tích hợp yếu tố tiện ích, hướng tới việc tìm ra các mẫu vừa phổ biến vừa có giá trị trong thực tiễn.

2. Công trình liên quan

2.1. Khai thác các mẫu Top-K phổ biến

Phương pháp khai thác Top-K quan trọng trong các hệ thống thông minh có thể chỉ khai thác các mẫu Top-K mà không cần tạo ra tất cả các mẫu. Đến nay, đã có nhiều phương pháp được đề xuất để khai thác các mẫu Top-K: Apriori [1] mở rộng khai thác các mẫu phổ biến nhất, nhưng tốn thời gian do phải quét cơ sở dữ liệu nhiều lần. Thuật toán FP-tree [2] khai thác các mẫu Top-K có độ dài tối thiểu nhất định. Thuật toán CRMN [3] giảm số lượng kết hợp mẫu trên một đường dẫn cây tiền tố, giúp tối ưu hóa quá trình khai thác. Thuật toán ETARM [4] sử dụng hai thuộc tính cắt tỉa để khai thác các mẫu Top-K. Thuật toán FTARM [5] cải tiến ETARM bằng kỹ thuật RGPP giúp giảm không gian tìm kiếm, tăng tốc quá trình khai thác. Ngoài các mẫu phổ biến, các thuật toán này cũng được áp dụng để khai thác các mẫu Top-K theo chuỗi, các mẫu với giá trị cao nhất, các mẫu đóng Top-K, và các mẫu Top-K trên các luồng dữ liệu.

2.2. Khai thác các mẫu tiện ích cao

Khai thác các mẫu có tiện ích cao là quá trình tìm kiếm những mẫu dữ liệu có giá trị lớn hoặc đóng vai trò quan trọng trong một tập dữ liệu. Trong bối cảnh dữ liệu chứa đựng mức độ không chắc chắn, khi mỗi mẫu được liên kết với một giá trị chưa chắc chắn, việc tìm kiếm và khai thác các mẫu này trở nên đặc biệt hữu ích và cần thiết.

Nhiều nghiên cứu đã được thực hiện để giải quyết vấn đề khai thác các tập mẫu có tiện ích cao (HUIs). Một ví dụ điển hình là thuật toán Two Phase [6] kết hợp với mô hình TWU (Transaction Weight Utility), được xem là một phương pháp hiệu quả trong việc khai thác HUIs. Thêm vào đó, thuật toán HUP-Growth [7] sử dụng cấu trúc cây để tối ưu hóa bộ nhớ trong quá trình khai thác. Một phương pháp khác là thuật toán PB, được phát triển dựa trên việc cài đặt các chỉ mục và áp dụng chiến lược cắt tỉa để nâng cao hiệu suất khai thác các tập mẫu có tiện ích cao.

2.3. Khai thác các mẫu phổ biến không chắc chắn

Các thuật toán khai thác mẫu không chắc chắn dựa trên nhiều phương pháp khác nhau. Thuật toán dựa trên Apriori sử dụng phương pháp tạo và kiểm tra ứng viên, gây tốn thời gian do phải quét cơ sở dữ liệu nhiều lần. Thuật toán dựa trên

H-mine khai thác mẫu thông qua cấu trúc liên kết siêu liên kết. Thuật toán FP-growth là thuật toán cây đầu tiên, nhưng yêu cầu bộ nhớ lớn do kích thước cây quá khổ. Thuật toán dựa trên Eclat sử dụng cấu trúc dữ liệu dọc để tìm mẫu. Thuật toán ràng buộc cận trên vẫn yêu cầu ba lần quét cơ sở dữ liệu và vẫn tạo nhiều dương tính giả. Thuật toán danh sách-based sử dụng UP-Lists và CUP-Lists để lưu trữ thông tin các giao dịch, cải tiến thuật toán LUNA để khai thác các mẫu phổ biến không chắc chắn top-K (Top-K UFPs) bằng cách đặt ngưỡng nội bộ ban đầu là 0 và áp dụng hai chiến lược: tăng ngưỡng và cắt tỉa. Tuy nhiên, TUFP không hiệu quả trong môi trường tương tác vì cần phải quét lại dữ liệu khi giá trị K thay đổi bởi người dùng.

3. Định Nghĩa Bài Toán

Dựa trên những kiến thức thu thập được từ các nghiên cứu liên quan đến việc khai phá K tập phổ biến có tiện ích cao trong cơ sở dữ liệu không chắc chắn, chúng tôi tiến hành phân tích các định nghĩa đầu vào, đầu ra để xây dựng bài toán.

TABLE 1: Danh sách giao dịch và giá trị tiện ích

Giao dịch	Danh sách sản phẩm	Giá trị tiện ích
T1	a: 0.9, c: 0.9, d: 0.6	a, c, d : 9 : 3, 1, 5
T2	a: 0.9, b: 0.9, c: 0.7, d: 0.6, e: 0.4	a, b, c, d, e : 20 : 6, 2, 3, 5, 4
T3	b: 0.5, c: 0.8, d: 0.9, f: 0.2	b, c, d, f : 17 : 2, 6, 5, 4
T4	c: 0.9, e: 0.1, f: 0.5	c, e, f : 15 : 5, 4, 6
T5	a: 0.4, b: 0.5, c: 0.9, d: 0.3	a, b, c, d : 23 : 6, 4, 3, 10
T6	d: 0.9, e: 0.1, f: 0.6	d, e, f : 13 : 5, 4, 4
T7	a: 0.9, b: 0.7	a, b : 5 : 3, 2

Đầu vào:

- $i = \{i_1, i_2, \ldots, i_m\}$ là một tập hợp gồm m sản phẩm khác nhau.
- $D = \{T_1, T_2, \ldots, T_n\}$ là tập hợp các giao dịch, mỗi giao dịch chứa:
  - Tập i có trong giao dịch chứa danh sách các sản phẩm.
  - Xác suất của mỗi sản phẩm tồn tại giao dịch, ký hiệu là $pr(i, T_q) \in D$, là xác suất của sản phẩm i trong giao dịch T_q với $1 \leq q \leq n$, và có giá trị trong khoảng $0 < pr(i, T_q) < 1$.
  - Tiện ích hay lợi nhuận của mỗi sản phẩm trong giao dịch, ký hiệu là $u(i, T_q) \in D$, là tiện ích của sản phẩm i trong giao dịch T_q.
- k là số lượng các tập phổ biến có tiện ích cao nhất được lựa chọn theo yêu cầu của người dùng.
- δ là ngưỡng tiện ích tối thiểu do người dùng đặt ra.

Mục tiêu:

Khai phá hay tìm thấy k tập phổ biến có tiện ích cao nhất dựa trên ngưỡng tiện ích δ từ cơ sở dữ liệu không chắc chắn D.

Các định nghĩa

Định nghĩa 1: Xác suất mong đợi (Expected probability) của một mẫu $\mathbf{x}$ trong giao dịch T_q, ký hiệu là $pr(\mathbf{x}, T_q)$, và được tính theo công thức:

$$pr(\mathbf{x}, T_q) = \prod_{i \in \mathbf{x}} pr(i, T_q)$$

Ví dụ: Với $\mathbf{x} = \{a, b\}$ trong giao dịch T_2, ta có:

$$pr(\mathbf{x}, T_2) = pr(a, T_2) \cdot pr(b, T_2) = 0.9 \cdot 0.9 = 0.81.$$

Định nghĩa 2: Độ hỗ trợ mong đợi (Expected support) của mẫu $\mathbf{x}$ trong tập dữ liệu D, ký hiệu là $\text{expSup}(\mathbf{x})$, được tính theo công thức:

$$\text{expSup}(\mathbf{x}) = \sum_{T_q \in D_{\mathbf{x}}} pr(\mathbf{x}, T_q),$$

trong đó $D_{\mathbf{x}}$ là tập các giao dịch chứa mẫu $\mathbf{x}$ trong cơ sở dữ liệu D.
Ví dụ:

$$\text{expSup}(a) = 0.9 + 0.9 + 0.4 + 0.9 = 3.1,$$

$$\text{expSup}(ab) = (0.9 \cdot 0.9) + (0.4 \cdot 0.5) + (0.9 \cdot 0.7) = 1.64.$$

Định nghĩa 3: R-Utility của một mẫu $\mathbf{x}$ trong cơ sở dữ liệu D, ký hiệu là $r\text{-}utility(\mathbf{x})$, được định nghĩa như sau:

$$r\text{-}utility(\mathbf{x}) = \sum_{T_q \in D_{\mathbf{x}}} \left(\sum_{i \in T_q \setminus \mathbf{x}} u(i, T_q) \right),$$

trong đó $T_q \setminus \mathbf{x}$ là tập các phần tử trong giao dịch T_q nhưng không bao gồm các phần tử trong mẫu $\mathbf{x}$.

Ví dụ: Giả sử mẫu $\mathbf{x} = \{a, c\}$ trong giao dịch T_1, thì $r\text{-}utility(\mathbf{x})$ là tổng tiện ích của các phần tử còn lại trong giao dịch T_1 không thuộc $\mathbf{x}$:

$$r\text{-}utility(\{a, c\}) = u(d, T_1) = 5.$$

Định nghĩa 4: Tiện ích của một mẫu $\mathbf{x}$ trong giao dịch T_q, ký hiệu là $u(\mathbf{x}, T_q)$, được định nghĩa như sau:

$$u(\mathbf{x}, T_q) = \sum_{i \in \mathbf{x}} u(i, T_q).$$

Ví dụ:

$$u(\mathbf{x}, T_1) = u(a, T_1) + u(c, T_1) = 3 + 1 = 4.$$

Định nghĩa 5: Tiện ích của một mẫu $\mathbf{x}$ trong cơ sở dữ liệu D, ký hiệu là $u(\mathbf{x})$, được định nghĩa như sau:

$$u(\mathbf{x}) = \sum_{T_q \in D_{\mathbf{x}}} u(\mathbf{x}, T_q).$$

Ví dụ:

$$u(\{a, c\}) = (3 + 1) + (6 + 3) + (6 + 3) = 22.$$

Định nghĩa 6: Tiện ích của một giao dịch T_q, ký hiệu là $tu(T_q)$, được định nghĩa như sau:

$$tu(T_q) = \sum_{i \in T_q} u(i, T_q).$$

Ví dụ:

$$tu(T_1) = u(a, T_1) + u(c, T_1) + u(d, T_1) = 3 + 1 + 5 = 9.$$

Định nghĩa 7: Tổng tiện ích của cơ sở dữ liệu D, ký hiệu là du, được định nghĩa như sau:

$$du = \sum_{T_q \in D} tu(T_q).$$

Ví dụ:

$$du = 9 + 20 + 17 + 15 + 23 + 13 + 5 = 102.$$

Định nghĩa 8: Mức tiện ích trọng số của một mẫu $\mathbf{x}$, ký hiệu là $twu(\mathbf{x})$, được định nghĩa như sau:

$$twu(\mathbf{x}) = \sum_{\mathbf{x} \subseteq T_q \wedge T_q \in D} tu(T_q).$$

Định nghĩa 9: Local Utility (LU) của phần tử a và b trong cơ sở dữ liệu D, ký hiệu là $lu(a, b)$, được định nghĩa như sau:

$$lu(a, b) = \sum_{T_q \in D} (u(a, T_q) + r\text{-}utility(a, T_q))$$
$$\text{nếu} \quad T_q \in D \text{ chứa cả phần tử } a \text{ và } b. \tag{1}$$

Trong đó: $u(a, T_q)$ là tiện ích của phần tử a trong giao dịch T_q, $r\text{-}utility(a, T_q)$ là $r_utility$ của phần tử a trong giao dịch T_q.

Ví dụ: Giả sử phần tử a và b đều xuất hiện trong giao dịch T_1 và có tiện ích và $r_utility$ lần lượt là:

$$u(a, T_1) = 3, \quad r\text{-}utility(a, T_1) = 5,$$

vậy LU của a và b trong giao dịch T_1 là:

$$lu(a, b) = (u(a, T_1) + r\text{-}utility(a, T_1)) = 3 + 5 = 8.$$

Định nghĩa 10: Mẫu tiện ích cao (HUP) trong cơ sở dữ liệu D là mẫu $\mathbf{x}$ thỏa mãn:

$$u(\mathbf{x}) \geq du \times \delta.$$

Ví dụ: Với $\delta = 20\%$ và $du = 102$, ta có:

$$u(a) = 18 < 102 \times 20\% = 20.4 \quad \text{(không phải HUP)},$$
$$u(\{a, c\}) = 22 > 102 \times 20\% = 20.4 \quad \text{(là HUP)}.$$

Định nghĩa 11: Top-k tập mẫu phổ biến không chắc chắn có tiện ích cao (TUHUFP) trong cơ sở dữ liệu D là tập các mẫu $\mathbf{x}$ thỏa mãn:

$$\mathbf{x} \in HUP \cap \text{expSup}(\mathbf{x}) \geq \text{expSup}_{\min}(TUHUFP),$$

trong đó:

$$\text{expSup}_{\min}(TUHUFP) = \min(\text{expSup}(\mathbf{x}) \mid \mathbf{x} \in TUHUFP).$$

4. Phương Pháp Thực Hiện

Nhiều phương pháp đã được xây dựng nhằm khai thác các tập mẫu có tiện ích cao và các tập mẫu trong môi trường dữ liệu không chắc chắn. Trong báo cáo này, chúng tôi lựa chọn hai thuật toán, SKYFUD và UHUOPM, để giải quyết bài toán khai thác K tập phổ biến có tiện ích cao trong cơ sở dữ liệu không chắc chắn.

4.1. Thuật Toán SKYFUD

SKYFUD (*Skyline Frequent Utility Discovery*) là thuật toán tận dụng khái niệm nhằm vào việc khai thác các tập mẫu phổ biến có tiện ích cao trong bộ dữ liệu không chắc chắn *skyline* và sử dụng IMCUP-List [8].

4.1.1. IMCUP-list

IMCUP-list là một cấu trúc dữ liệu được thiết kế để hỗ trợ việc khai thác tập mẫu không chắc chắn có tiện ích cao top-k. Cấu trúc này bao gồm các thành phần sau:

- Tên tập mẫu.
- Độ hỗ trợ mong đợi (expected support) của mẫu.
- Một danh sách UP-list chứa các mẫu.
- Thuộc tính **max** được sử dụng trong **TEP-List** để lưu trữ giá trị xác suất mong đợi lớn nhất của mẫu.
- Ngưỡng tối đa (*overestimate*) của một tập mẫu XY được xác định bằng công thức:

  $$\text{overestimate}(XY) = \text{expSup}(X) \times \max(Y)$$

  Trong đó:
  - expSup(X): Là giá trị xác suất mong đợi của tập X.
  - max(Y): Là giá trị xác suất lớn nhất của tập Y.

  Bằng cách tính trước ngưỡng tối đa của một tập mẫu và so sánh giá trị này với ngưỡng expSup trong danh sách **top-k**, chúng ta có thể xác định được khả năng tồn tại của tập mẫu đó. Điều này cho phép:
  1) **Cắt giảm không gian lưu trữ**: Loại bỏ các tập mẫu không đủ tiềm năng mà không cần khai thác sâu hơn.
  2) **Nâng cao hiệu suất khai thác**: Tập trung tài nguyên tính toán vào các tập mẫu có triển vọng hơn, giảm thời gian xử lý tổng thể.

  Phương pháp này là một bước đột phá trong việc khai thác các tập mẫu có tiện ích cao trên tập dữ liệu không chắc chắn, giúp tối ưu hóa hiệu quả xử lý dữ liệu lớn và phức tạp. [9]
- Utility sẽ lưu trữ giá trị tiện ích của mẫu trong tập dữ liệu
- Thuộc tính utility còn lại ($r\_utility$) để lưu trữ giá trị tiện ích hay giá trị có khả năng mở rộng của mẫu trong cơ sở dữ liệu.
- Thuộc tính TWU (transaction-weight-utility) là tổng các tiện ích của giao dịch trong UP-List.
- Một tập hợp chứa dữ liệu của mẫu gọi là UP-list.
- Dữ liệu UP-list gồm:
  - TID: Địa chỉ giao dịch.
  - Tiện ích của mẫu trong từng giao dịch.
  - Xác suất mong đợi (expected probability) của mẫu trong mỗi giao dịch.
  - Tiện ích của mỗi giao dịch.
  - Tiện ích còn lại của mỗi giao dịch.

Dựa theo dữ liệu ví dụ ở Bảng 1, Hình 1mô tả cấu trúc IMCUP của các sản phẩm có trong cơ sở dữ liệu, bằng việc lưu trữ như thế, chúng ta có thể tính toán và khai phá một cách hiệu quả.

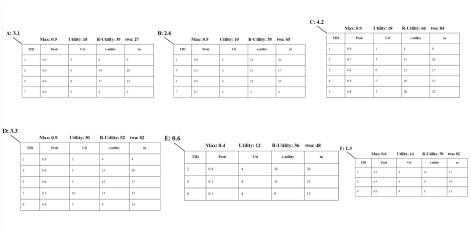

Fig. 1: Mô hình IMCUP-list.

Hình 2 thể hiện quá trình kết hợp mẫu A, B từ IMCUP-list của item A và item B, ta xác định các UP trùng nhau của mẫu từ các TID trùng lặp (2, 5, 7). Sau đó dự vào các định nghĩa 1, 3 và 4 để xác định các giá trị xác suất mong đợi, tiện ích và giá trị tiện ích còn lại (tính bằng giá trị tiện ích còn lại của item b) của mẫu UP-list.

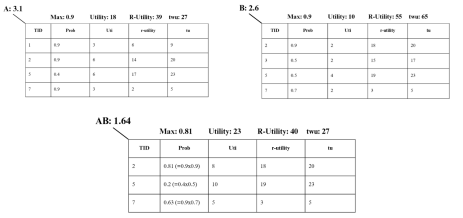

Fig. 2: Mô hình liên kết IMCUP-list cho mẫu AB.

4.1.2. Các chiến lược nâng cao hiệu quả khai thác

Nâng ngưỡng: Danh sách k tập phổ biến có tiện ích cao được duy trì dưới dạng một mảng sắp xếp theo thứ tự giảm dần dựa trên độ hỗ trợ mong đợi (*expSup*). Ngưỡng của top-k được xác định bởi độ hỗ trợ mong đợi của phần tử cuối cùng trong mảng.

Khi một phần tử mới thỏa mãn điều kiện được thêm vào danh sách top-k, phần tử cuối cùng sẽ bị loại bỏ nếu như danh sách đã đủ k phần tử. Đồng thời, ngưỡng hỗ trợ mong đợi sẽ được cập nhật theo giá trị của phần tử cuối sau khi loại bỏ.

Chiến lược này mang lại những lợi ích:

- **Giảm không gian tìm kiếm**: Loại bỏ các tập mẫu không đủ điều kiện.
- **Ưu tiên các tập mẫu tiềm năng**: Tập trung khai thác các tập mẫu có độ hỗ trợ cao hơn.

Cách tiếp cận này giúp tối ưu hóa thời gian xử lý và nâng cao hiệu quả khai thác dữ liệu.

Cắt tỉa theo ngưỡng được nâng: Phương thức *Search* sẽ tạo ra các tập mẫu mới, trong quá trình này, thuật toán sẽ không tạo ra tập mẫu mới nếu độ hỗ trợ mong đợi của tập mẫu tham gia và tập kết hợp mở rộng thấp hơn ngưỡng hỗ trợ mong đợi. Chiến lược này giúp tối ưu thời gian xử lý và không gian lưu trữ của thuật toán.

Cắt tỉa theo TWU: Một *danh sách ứng viên* tham gia sẽ được tạo ra bằng thuật toán trong quá trình khai thác các tập mẫu. Nhằm giảm số lượng ứng viên tham gia khai thác, thuật toán sẽ không thêm tập mẫu vào danh sách này nếu nó giá trị TWU của nó nhỏ hơn ngưỡng tiện ích.

Ngưỡng tối đa: Ngưỡng tối đa (*overestimate*) của một tập XY được xác định bằng $expSup(X) \times max(Y)$. Tập mẫu sẽ bị loại bỏ nếu ngưỡng này nhỏ hơn ngưỡng đặt ra trong top-k. Ngược lại, tập mẫu này sẽ trở thành tập mẫu "tiềm năng" trong top-k, từ đó cắt giảm không gian lưu trữ và nâng cao hiệu suất khai thác [10].

Cắt tỉa theo tiện ích cục bộ: Tiện ích cục bộ (*local utility*) của một tập X được xác định bằng $utility(X)+r\_utility(X)$. Nếu nó nhỏ hơn ngưỡng tiện ích thì không cần mở rộng.

4.1.3. Mã giả thuật toán SKYFUP

Giả sử D là cơ sở dữ liệu và thuật toán yêu cầu tìm kiếm các tập mẫu từ cơ sở dữ liệu. Các bước của thuật toán SKYFUP được mô tả như sau:

1) Đọc cơ sở dữ liệu D và tính min_util bằng công thức $data\_util * percentage$.
2) Tìm kiếm IMCUP-list của tập 1-mẫu (chỉ chứa 1 phần tử, như A hoặc B) trong cơ sở dữ liệu D và sắp xếp theo thứ tự giảm dần độ hỗ trợ mong đợi (expected support).
3) Lấy tối đa k phần tử có độ hỗ trợ mong đợi cao nhất và lưu vào danh sách top-k.
4) Chọn lọc các mẫu tham gia thông qua TWU (Transactional Weighted Utility). Chỉ lấy các mẫu có TWU lớn hơn ngưỡng tiện ích. Đồng thời kiểm tra nếu $utility$ của phần tử đang xét thỏa điều kiện hay lớn hơn min_util thì thêm vào danh top_UHUFP.
5) Thiết lập $threshold(ngng)$ là độ mong đợi tin cậy của phần tử hay $threshold$ tương ứng với phần tử có $expSup$ thấp nhất trong top-k khi top-k có đủ k phần tử.
6) Gọi phương thức `Search`, truyền vào giá trị k và danh sách các mẫu tham gia. Phương thức này áp dụng chiến lược chia để trị nhằm tạo ra các *IMCUP-list* của các tập mẫu lớn hơn từ *UP-list* của các tập mẫu tham gia. Với mỗi tập mẫu, thuật toán kiểm tra các điều kiện:
   - Nếu độ hỗ trợ mong đợi (*expSup*) của tập mẫu thấp hơn ngưỡng hỗ trợ mong đợi hiện tại, tập mẫu sẽ bị loại bỏ.
   - Nếu tổng tiện ích hoặc tiện ích cục bộ của tập mẫu thấp hơn ngưỡng tiện ích, tập mẫu cũng sẽ bị bỏ qua.

   Ngược lại, nếu tập mẫu thỏa mãn các điều kiện, thuật toán sẽ thêm nó vào danh sách kết quả và tiếp tục kết hợp tập mẫu đó để tìm kiếm các tập mẫu lớn hơn.
7) Chiến lược tìm kiếm này sẽ tiếp tục cho đến khi tất cả mẫu được xem xét.

Mã giả thuật toán được mô tả và mã giả Search được mô tả trong thuật toán 1.

Thuật toán SKYFUD

Input:
U: cơ sở dữ liệu không chắc chắn; H: cơ sở dữ liệu tiện ích;
`percentage`: ngưỡng tiện ích của người dùng (%); k: số lượng tập mẫu phổ biến không chắc chắn có tiện ích cao cần tìm.

Output:
k tập mẫu phổ biến không chắc chắn có tiện ích cao.

Bước 1: Đọc cơ sở dữ liệu U và H để tạo danh sách $IMCUP$ (1-mẫu).

Bước 2: Tính tiện ích tổng thể của cơ sở dữ liệu:

$$\text{databaseUtil} \leftarrow \sum_{T_q \in H} \text{tu}(T_q)$$

Bước 3: Tính giá trị ngưỡng tiện ích tối thiểu:

$$\text{minUtil} \leftarrow \text{databaseUtil} \times \texttt{percentage}$$

Bước 4: Sắp xếp lại danh sách $IMCUP$ (1-mẫu) giảm dần theo giá trị `expSup`.

Bước 5: Lọc ra các ứng viên thỏa mãn điều kiện thông qua hàm `create_candidate`:

$$\text{Candidates} \leftarrow \text{Candidates} \cup \{IMCUP_x \in IMCUP \mid CUP_x.\text{TWU} \geq \text{minUtil}\}$$

Bước 6: Lọc ra các tập top_UHUFP có tiện ích cao thông qua hàm `check`:

$$\text{top\_UHUFP} \leftarrow \text{top\_UHUFP} \cup \{IMCUP_x \in IMCUP \mid IMCUP_x.\text{utility} \geq \text{minUtil}\}$$

Bước 7: Cập nhật `threshold` dựa trên phần tử cuối trong $IMCUP$:

$$\text{threshold} \leftarrow \text{phần tử cuối của } IMCUP_x \in IMCUP, \text{ với } (\text{top\_UHUFP}).\text{size} = k$$

Bước 8: Gọi hàm đệ quy `Search(Candidates, k)` để tiếp tục tìm kiếm với danh sách ứng viên.

Thuật toán Search

Input:
P: một danh sách chứa các IMCUP, và k là số lượng UHUFP cần tìm.

Output:
k tập mẫu phổ biến không chắc chắn có tiện ích cao.

Nếu $|P| > 1$, thực hiện:
  Với mỗi $Px \in P$:
    Nếu $P_x.utility + P_x.r\_utility > minUtil$
      extendIMCUP $\leftarrow \emptyset$
      Với mỗi $Py \in P, y > x$, thực hiện:
        overestimate $\leftarrow Px$.expSup $\times Py$.max
        Nếu overestimate $<$ threshold hoặc Px.expSup $<$ threshold thì:
          Tìm kiếm với Px tiếp theo
          Kết thúc vòng lặp.
        combined $\leftarrow$ combine_cup(Px, Py)
        Nếu combined.expSup $>$ threshold, thì:
          Thêm vào kết quả khi combined là top_UHUFP và đặt lại threshold khi cần.
          check(combined, k)
          Nếu combined.TWU $\geq$ minUtil, thì:
            extendIMCUP $\leftarrow$ extendIMCUP $\cup$ combined
          Kết thúc.
        Kết thúc.
      Gọi lại Search(extendIMCUP, k) để tiếp tục tìm kiếm.
    Kết thúc.

Algorithm 1: Thuật Toán SKYFUD

4.2. Thuật Toán UHUOPM

Điểm hạn chế của thuật toán SKYFUP cần phải truyền vào một ngưỡng tiện ích. Việc xác định ngưỡng tiện ích một cách chính xác là rất khó. Nếu ngưỡng quá cao thì quá trình khai thác làm mất đi nhiều mẫu tiện ích tốt dẫn đến kết quả không tốt. Nếu ngưỡng quá thấp thì thời gian và không gian khai thác rất lớn. Do đó chúng tôi đề xuất thuật toán UHUOPM. UHUOPM (*Uncertain High Utility Optimal Pattern Mining*) là một thuật toán tối ưu hóa trong việc khai thác các mẫu có tiện ích cao trong dữ liệu không chắc chắn, và sử dụng cấu trúc dữ liệu PFU-table. Thuật toán sẽ khai thác các ngưỡng tiện ích cao không chắn chắn mà không cần truyền vào ngưỡng tiện ích đồng thời cũng cân bằng giữa tiện ích và xác suất.

4.2.1. PFU-table

PFU-table là cấu trúc dữ liệu được sử dụng để phục vụ khai thác top-k tập mẫu tiện ích cao trên tập dữ liệu không chắc chắn. Cấu trúc PFU-table bao gồm:

- Tên tập mẫu.
- Số lần xuất hiện của vật phẩm sup.
- Xác suất pro của mẫu.
- PUO-list là một tập hợp chứa các thông tin của mẫu.
- Thuộc tính utility để lưu giá trị tiện ích của mẫu.
- Thuộc tính utility còn lại ($r\_utility$) để lưu trữ giá trị tiện ích hay giá trị có khả năng mở rộng của mẫu trong cơ sở dữ liệu.
- Thuộc tính zeros utility ($zrutils$) để lưu trữ giá trị tiện ích hay giá trị có khả năng mở rộng của mẫu trong cơ sở dữ liệu.
- Thuộc tính TWU (transaction-weight-utility): tổng các tiện ích của giao dịch trong PUO-List.
- Dữ liệu PUO-list gồm:
  - TID: Đại diện cho địa chỉ giao dịch.
  - Tiện ích của mẫu trong mỗi giao dịch.
  - Xác suất mong đợi (expected probability) của mẫu trong từng giao dịch.
  - Tiện ích mỗi giao dịch.
  - Tiện ích còn lại của mỗi giao dịch.
  - Tiện ích của mẫu khi $r\_utility = 0$.

Dựa theo dữ liệu ví dụ ở Bảng 1, Hình 3 mô tả cấu trúc dữ liệu PFU-table của sản phẩm A và B trong cơ sở dữ liệu.

Hình 4 thể hiện cách tính các giá trị của việc kết hợp mẫu A, B từ PFU-table của item A và item B, ta xác định các PUO trùng nhau của mẫu từ các TID trùng lặp (2, 5, 7). Sau đó dự vào các định nghĩa 1, 3 và 4 để xác định các giá trị xác suất mong đợi, tiện ích, giá trị tiện ích còn lại (tính bằng giá trị tiện ích còn lại của item b) và giá trị tiện ích không (tính bằng giá trị tiện tích không của item b) của mẫu PUO-list.

4.2.2. Các chiến lược khai thác nâng cao hiệu suất

Tính độ hội tụ: Trong quá trình khởi tạo ứng viên để khai phá, tính độ hội tụ bằng cách nếu $TWU(A, B) == TWU(A)$ thì item B có độ hội tụ thấp không cần phải khai phá.

Nâng ngưỡng: Nếu $|PHUOP| > k$ thì xóa đi phần tử có $utility$ và pro thấp nhất. Đồng thời cập nhật $min\_utility$ và min_pro tương ứng.

A

Sup: 4	Pro: 3.1	Utility: 18	R-Utility: 39	twu: 57	zrutility: 0
TID	prob	Uti	r-uti	tu	zr-uti
1	0.9	3	6	9	0
2	0.9	8	14	20	0
3	0.4	8	17	23	0
7	0.9	3	2	5	0

(a) Cấu trúc PFU-table của item A.

B

Sup: 4	Pro: 3.1	Utility: 10	R-Utility: 55	twu: 65	zrutility: 0
TID	prob	Uti	r-uti	tu	zr-uti
2	0.9	2	18	20	0
3	0.5	2	15	17	0
5	0.5	4	19	23	0
7	0.7	2	3	5	0

(b) Cấu trúc PFU-table của item B.

Fig. 3: Cấu trúc dữ liệu PFU-table với hai phần tử A và B.

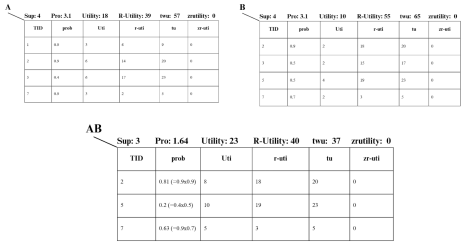

Fig. 4: Mô hình liên kết UHUOPM-list cho mẫu AB.

Cắt tỉa theo ngưỡng được nâng: Trong quá trình khai thác nếu item đang xét có $pro < min\_pro$ thì dừng tìm kiếm. Kiểm tra $utility + r\_utility - zrutil$ nếu thấp hơn $min\_utility$ thì không mở rộng phần tử này. Nếu pro của item cần kết hợp nhỏ hơn min_pro hoặc $local\_utility(A, B)$ thấp hơn $min\_utility$ thì không cần mở rộng.

Cắt tỉa các phần tử ứng viên: Trong quá trình khai thác, thuật toán sẽ có một danh sách chứa các tập mẫu là *danh sách ứng viên* tham gia khai phá. Thuật toán sẽ không thêm các phần tử đã kết hợp vào danh sách ứng viên nếu pro hay $utility+r\_utility$ thấp hơn min_pro hoặc $min\_utility$ tương ứng. Do đó giảm thiểu việc khai phá các phần tử không triển vọng.

Ngưỡng tối đa: Ngưỡng tối đa (*overestimate*) của một tập XY được xác định bằng $expSup(X) \times max(Y)$. Ngưỡng tiện ích tối đa của một tập XY được xác định bằng $utility(X) + max(Y)$. Nếu các ngưỡng này nhỏ hơn ngưỡng hiện tại thuật toán thì không cần phải kết hợp với phần tử đang xét Ngược lại, tập mẫu này sẽ trở thành tập mẫu "tiềm năng" trong top-k, từ đó cắt giảm không gian lưu trữ và nâng cao hiệu suất khai thác.

Cắt tỉa theo ngưỡng mở rộng: Các thuật toán sử dụng cấu trúc dữ liệu $UP - list$ và thuật toán $TWO - Phase$ để tạo ra tổ hợp ABC cần phải kết hợp hoặc tạo ra AB và AC. Nếu AC không phải *tiện ích cao* thì việc tạo ra nó là vô nghĩa và ảnh hưởng bộ nhớ. Do đó cần phải tính $utility(B) + r\_utility(B) + utility(X)$ với B là phần tử cần kết hợp với phần tử đang xét và X là tiền tố của phần tử, trong trường hợp này là A. Nếu phép tính này nhỏ hơn $min\_utility$ thì không cần kết hợp, đồng thời kiểm tra $utility(B) + r\_utility(B)$ và $pro(B)$ nếu thỏa các điều kiện, thì thêm B vào *danh sách ứng viên*, trong tường hợp này là C thay vì AC như thuật toán khác.

Thay đổi thuật toán construct: Thay đổi cách kết hợp hai phần tử A và B từ sử dụng giải thuật 5Construct [11] sang

Algorithm 1 Construct utility-list

Input:
UL(P), the utility-list of itemset *P*;
UL(Px), the utility-list of itemset *Px*;
UL(Py), the utility-list of itemset *Py*

Output: *UL(Pxy)*, the utility-list of itemset *Pxy*.

```
UL(Pxy) = NULL;
for each (tuple ex ∈ UL(Px)) do
  if (∃ey ∈ UL(Py) and ex.tid = ey.tid) then
    if (UL(P) is not empty) then
      Search element e ∈ UL(P) such that e.tid = ex.tid;
      exy ⟵ (ex.tid; ex.iutil + ey.iutil - e.iutil; ey.rutil);
    else
      exy ⟵ (ex.tid; ex.iutil + ey.iutil; ey.rutil);
    end if
    UL(Pxy) ⟵ UL(Pxy) ∪ exy;
  end if
end for
return UL(Pxy);
```

Fig. 5: Construct.

giải thuật Iconstruct 6iConstruct [11]. Cho thấy hiệu quả hay tốc độ tính toán được cải thiện. Độ phức tạp của thuật toán Construct là $O(N^2)$, trong khi của giải thuật iConstruct là $O(N)$, cho thấy giải thuật mới có khả năng tính toán hiệu quả hơn khi khai phá các tập dữ liệu lớn.

fhdgfjsdkg

Algorithm 2 **iConstruct** Algorithm

Input:
UL(P), the utility-list of itemset *P*;
UL(Px), the utility-list of itemset *Px*;
UL(Py), the utility-list of itemset *Py*

Output: *UL(Pxy)*, the utility-list of itemset *Pxy*.

```
UL(Pxy) = NULL; Let i, j = 0;
Let total = UL(Px).sumIutils + UL(Py).sumIutils + UL(Px).sumRutils + UL(Py).sumRutils; //for the EA strategy
while (i < UL(Px).size and j < UL(Py).size) do
  if ( UL(Px)[i].tid = UL(Py)[j].tid) then
    UL(Pxy) ⟵ (UL(Px)[i].tid; UL(Px)[i].iutil + UL(Py)[j].iutil; UL(Py)[j].rutil);
    i = i + 1; j = j + 1;
  else if (UL(Px)[i].tid < UL(Py)[j].tid) then
    total = total - (UL(Px)[i].iutil + UL(Px)[i].rutil);
    i = i + 1;
  else
    total = total - (UL(Py)[j].iutil + UL(Py)[j].rutil);
    j = j + 1;
  end if
  if (total < min_util) then
    return null; //EA strategy
  end if
end while
if (UL(P) ≠ ∅ and UL(Pxy) ≠ ∅) then
  Let i = j = 0;
  while (i < UL(Pxy).size and j < UL(P).size) do
    if (UL(Pxy)[i].tid = UL(P)[j].tid) then
      UL(Pxy)[i++].iutil -= UL(P)[j].iutil;//Subtract utilities of tuples existing in prefix UL(P)
    end if
    j = j + 1;
  end while
end if
return UL(Pxy);
```

Fig. 6: iConstruct.

4.2.3. *Mã giả thuật toán UHUOPM*

Mã giả thuật toán được mô tả thông qua giải thuật 2.

5. Thực nghiệm và Phân tích kết quả

Các mẫu được khai thác sẽ được đánh giá lại nhằm đảm bảo tính chính xác và tối ưu của thuật toán trước khi xuất ra kết quả cuối cùng.

5.1. *Tập dữ liệu*

Các thuật toán được thực nghiệm trên các tập dữ liệu khác nhau với các thông tin tương ứng bao gồm: số lượng giao dịch, số lượng vật phẩm, trung bình, độ dài tối đa, và kiểu dữ liệu của chúng.

Dataset	#Trans	#Item	#Avg	#Maxlen	#Type
Foodmart	21,557	1,550	4	11	sparse
Chess	3,196	76	37	37	dense
Retail	88,162	3,999	10	28	sparse

Fig. 7: Tập dữ liệu thực nghiệm.

Thuật toán SKYFUP sẽ được chạy trên các tập dữ liệu Foodmart, Retail và Chess với các ngưỡng tiện tích tương ứng là 0.0004, 0.0001, 0.001 và số lượng top K mẫu lần lượt là 100, 300, 500, 700, 900.

Thuật toán UHUOPM cũng sẽ được thực nghiệm trên các tập dữ liệu Foodmart, Retail, Chess với số lượng Top K mẫu tương ứng là 100, 300, 500, 700, và 900.

Các thuật toán được thực nghiệm trên môi trường Colab, sử dụng Python 3 với phần cứng CPU và dung lượng RAM khả dụng là 13.61 GB

5.2. *So sánh các thuật toán*

5.2.1. *So sánh TUHUFP và SKYFUP*

Thuật toán SKYFUP vượt trội hơn **TUHUFP** cả về mọi mặt: ứng viên, tổng tiện ích, thời gian và bộ nhớ. Điều này được thể hiện rõ ở tập dữ liệu **Foodmart**. Đối với tập dữ liệu **Chess** và **Retail**, SKYFUP vượt trội hơn TUHUFP ở mặt tiết kiệm thời gian và bộ nhớ rất nhiều.

Foodmart:

K	Candidate		Sum Utility		Time		Memory	
	TUHUFP	SKYFUP	TUHUFP	SKYFUP	TUHUFP	SKYFUP	TUHUFP	SKYFUP
100	239	**228**	1124279	1124279	2	2	0.06	**0.04**
300	1860	**1518**	3131398	3131398	5	**3**	0.15	**0.14**
500	4974	**3862**	5130303	5130303	6	**5**	0.18	**0.17**
700	9208	**6753**	6881503	**6901035**	8	8	0.25	**0.22**
900	11511	**9700**	8473791	**8498406**	13	13	0.35	**0.24**

Fig. 8: So sánh SKYFUP và TUHUFP trên tập dữ liệu Foodmart.

Đối với tập dữ liệu Foodmart

- **Ứng viên: (Candidate)** Ứng viên của SKYFUP luôn nhỏ hơn TUHUFP, và chênh lệch ngày càng rõ rệt khi K tăng lên. (VD: K = 900, TUHUFP: 11511, trong khi SKYFUP chỉ có 9700).
- **Tổng tiện ích (Sum Utility):** Tổng tiện ích đạt được của cả hai thuật toán gần như tương đương nhau, cho thấy dù có sự khác biệt về số lượng ứng viên, cả hai phương pháp đều hiệu quả trong việc xác định các itemset có tiện ích cao.

- **Thời gian thực thi và bộ nhớ sử dụng: (Time and Memory)** Trong hầu hết mọi trường hợp, thuật toán SKYFUP đều tốt hơn TUHUFP rất nhiều ở việc tiết kiệm được thời gian và bộ nhớ.

Chess:

K	Candidate		Sum Utility		Time		Memory	
	TUHUFP	SKYFUP	TUHUFP	SKYFUP	TUHUFP	SKYFUP	TUHUFP	SKYFUP
100	216	216	10243630	10243630	96	**85**	26.37	**26.34**
300	1573	1573	27630427	27630427	460	**448**	88.70	**88.70**
500	3936	3936	41545459	41545459	784	**781**	130.45	**130.42**
700	6691	6691	63876687	63876687	1051	**1038**	172.54	**171.78**
900	9536	9536	96388425	96388425	1354	1354	220.15	220.15

Fig. 9: So sánh SKYFUP và TUHUFP trên tập dữ liệu Chess.

Đối với tập dữ liệu Chess

- **Ứng viên và tổng tiện ích:** Cả hai thuật toán gần như không có sự khác biệt.
- **Thời gian thực thi:** SKYFUP có thời gian chạy ngắn hơn TUHUFP trong các giá trị K nhỏ.
- **Bộ nhớ sử dụng:** SKYFUP tiêu thụ bộ nhớ gần như tương đương hoặc thấp hơn TUHUFP một chút, nhưng chênh lệch không đáng kể.

Retail:

K	Candidate		Sum Utility		Time		Memory	
	TUHUFP	SKYFUP	TUHUFP	SKYFUP	TUHUFP	SKYFUP	TUHUFP	SKYFUP
100	129	129	6452283	6452283	594	**532**	18.48	18.48
300	375	375	9925041	9925041	780	**712**	22.46	**22.44**
500	657	657	12331117	12331117	1062	**906**	27.91	**27.91**
700	921	921	14035375	14035375	1322	**1170**	31.40	**31.39**
900	1201	1201	15668304	15668304	1565	**1402**	34.59	**34.58**

Fig. 10: So sánh SKYFUP và TUHUFP trên tập dữ liệu Retail.

Đối với tập dữ liệu Retail

- **Ứng viên và tổng tiện ích:** Tương tự như tập dữ liệu Chess, cả hai thuật toán đều có số lượng Candidate và giá trị Sum Utility giống nhau.
- **Thời gian thực thi:** SKYFUP luôn nhanh hơn TUHUFP, với sự chênh lệch rõ ràng khi K tăng.
- **Bộ nhớ sử dụng:** Bộ nhớ tiêu thụ của hai thuật toán gần như tương đương nhau trên toàn bộ các giá trị K.

5.2.2. So sánh SKYFUP và UHUOPM

Đối với tập dữ liệu Foodmart, thuật toán **UHUOPM** tốt hơn thuật toán **SKYFUP** ở các mặt như ứng viên, tổng tiện ích và bộ nhớ rất nhiều. Ở tập dữ liệu Chess, thuật toán UHUOPM tốt hơn ở ứng viên, tổng tiện ích và thời gian. Tuy nhiên bộ nhớ mà thuật toán này sử dụng lại lớn hơn rất nhiều so với SKYFUP Tuy nhiên, thuật toán UHUOPM lại tốn thời gian xử lý hơn thuật toán SKYFUP rất nhiều do phải sử dụng nhiều tính toán.

Foodmart:

K	Candidate		Sum Utility		Time		Memory	
	SKYFUP	UHUOPM	SKYFUP	UHUOPM	SKYFUP	UHUOPM	SKYFUP	UHUOPM
100	228	**100**	1124279	**1728188**	2	2	0.04	0.04
300	1518	**300**	3131398	**4309252**	3	**2**	0.14	**0.13**
500	3862	**500**	5130303	**6352140**	5	**3**	0.17	**0.14**
700	6753	**700**	6901035	**8025064**	8	**5**	0.22	**0.15**
900	9700	**900**	8498406	**9373173**	13	**9**	0.24	**0.20**

Fig. 11: So sánh SKYFUP và UHUOPM trên tập dữ liệu Foodmart.

Đối với Tập dữ liệu Foodmart

- **Candidate:** UHUOPM tạo ra số lượng Candidate ít hơn đáng kể so với SKYFUP trong mọi trường hợp.
- **Tổng tiện ích (Sum Utility):** Giá trị Sum Utility của UHUOPM cao hơn SKYFUP trong tất cả các trường hợp.
- **Thời gian thực thi (Time):** UHUOPM có thời gian xử lý nhanh hơn SKYFUP trên tập dữ liệu này
- **Bộ nhớ sử dụng (Memory):** UHUOPM tiêu thụ ít bộ nhớ hơn SKYFUP trong hầu hết các trường hợp.

Chess:

K	Candidate		Sum Utility		Time		Memory	
	SKYFUP	UHUOPM	SKYFUP	UHUOPM	SKYFUP	UHUOPM	SKYFUP	UHUOPM
100	216	276	10243630	**15422196**	85	134	26.34	1103.00
300	1573	**586**	27630427	**65242081**	448	**275**	88.70	2309.92
500	3936	**734**	41545459	**68829537**	781	**426**	130.42	3333.67
700	6691	**643**	63876687	**75358311**	1038	**392**	171.78	3187.87
900	9536	**643**	96388425	**75358311**	1354	**351**	220.15	3188.01

Fig. 12: So sánh SKYFUP và UHUOPM trên tập dữ liệu Chess.

Đối với tập dữ liệu Chess

- **Candidate và Sum Utility:** UHUOPM tạo ra số lượng Candidate thấp hơn SKYFUP trong hầu hết mọi trường hợp. Giá trị Sum Utility của UHUOPM cao hơn đáng kể so với SKYFUP trong mọi trường hợp.
- **Thời gian thực thi (Time):** SKYFUP có thời gian thực thi ngắn hơn nhiều so với UHUOPM trên tập dữ liệu Chess, do UHUOPM thực hiện nhiều tính toán hơn.
- **Bộ nhớ sử dụng (Memory):** UHUOPM tiêu tốn bộ nhớ cao hơn rất nhiều so với SKYFUP.

Đối với tập dữ liệu Retail

- **Candidate:** UHUOPM tạo ra số lượng Candidate nhiều hơn SKYFUP trong mọi trường hợp.
- **Sum Utility:** Giá trị Sum Utility của UHUOPM cao hơn đáng kể so với SKYFUP.
- **Thời gian thực thi (Time):** SKYFUP tiếp tục vượt trội hơn về tốc độ so với UHUOPM.
- **Bộ nhớ sử dụng (Memory):** SKYFUP tiêu thụ ít bộ nhớ hơn so với UHUOPM trên tập dữ liệu này.

Retail:

K	Candidate		Sum Utility		Time		Memory	
	SKYFUP	UHUOPM	SKYFUP	UHUOPM	SKYFUP	UHUOPM	SKYFUP	UHUOPM
100	129	149	6452283	**7122260**	532	**492**	18.48	82.95
300	375	461	9925041	**11284757**	712	817	22.44	139.78
500	657	774	12331117	**13824661**	906	1188	27.91	170.19
700	921	1093	14035375	**15753183**	1170	1475	31.39	191.92
900	1201	1406	15668304	**17462048**	1402	1854	34.58	214.66

Fig. 13: So sánh SKYFUP và UHUOPM trên tập dữ liệu Retail.

6. KẾT LUẬN VÀ HƯỚNG ĐI TRONG TƯƠNG LAI

Cả ba thuật toán đều thực hiện việc tạo UP List và ứng viên, tuy nhiên, cả hai vẫn chưa đạt được sự khai phá toàn diện. Dưới đây là phân tích kết quả các thuật toán:

- **Thuật toán TUHUFP:**
  - Chưa tối ưu về tất cả các mặt, lẫn cả ứng viên, tổng tiện ích, thời gian và bộ nhớ.
  - Tốn nhiều bộ nhớ hơn, cần giảm bớt việc lưu trữ dữ liệu dư thừa.
- **Thuật toán SKYFUP:**
  - Tối ưu hơn TUHUFP về cả 4 mặt trên cả 3 tập dữ liệu.
  - Tuy nhiên vấn đề của thuật toán này yêu cầu người dùng phải truyền vào một ngưỡng tiện ích, dẫn đến sự phụ thuộc và khó khăn trong một số trường hợp.
- **Thuật toán UHUOPM:**
  - Thuật toán UHUOPM đã giải quyết được vấn đề yêu cầu người dùng truyền vào ngưỡng tiện ích.
  - UHUOPM đã cho thấy được sự tối ưu về mặt ứng viên, tổng tiện ích, và thời gian trên 2 tập dữ liệu là Foodmart và Chess.
  - Tuy nhiên, đối với tập dữ liệu lớn như Retail, để cân bằng giữa xác suất và tiện ích, sẽ cần khai phá nhiều hơn nên sẽ tốn thời gian và bộ nhớ hơn SKYFUP rất nhiều.
- **Nếu chỉ xét về Ứng viên và Tổng tiện ích:** Thuật toán **UHUOPM** sẽ là thuật toán tốt nhất trong cả 3 thuật toán.
- **Nếu chỉ xét về Thời gian:** Thuật toán **SKYFUP** và **UHUOPM**, sẽ tối ưu thời gian hơn 2 thuật toán còn lại. Tuy nhiên tốt nhất là SKYFUP.
- **Nếu chỉ xét về Bộ nhớ:** Thuật toán **SKYFUP** và, sẽ tối ưu nhất trong 3 thuật toán, do **TUHUFP** tốn quá nhiều thời gian để duyệt, còn **UHUOPM** lại tốn quá nhiều bộ nhớ cho việc tính toán.

Kết luận: Như đã thấy thuật toán SKYFUP đã cải thiện được hiện suất bộ nhớ, thời gian, ứng cữ viên cho thuật toán TUHUFP. Đồng thời thuật toán UHUOPM đã giải quyết được vấn đề phải truyền ngưỡng tiện ích của thuật toán SKYFUP. Dựa vào kết quả nếu người dùng hiểu rõ về cơ sỡ dự liệu hay tiện ích thì có thể sử dụng thuật toán $SKYFUP$ để nhanh chóng đạt kết quả. Khi muốn chỉ truyền k thì sử dụng thuật toán $UHUOPM$ để đạt được các tiện ích cao. Nhưng do phải cân bằng giữa xác suất và tiện ích, vì thế thời gian cũng như tiêu thụ bộ nhớ rất lớn đặc biệt ở các tập lớn như retail. Nhưng thuật toán cũng cho thấy đạt được hiệu quả tiện ích, thời gian, ứng cử viên so với thuật toán $SKYFUP$.

Cả 3 thuật toán đều sử dụng cấu trúc dữ liệu gọi chung là $UP-list$ và các chiến lược ngắt ngưỡng dựa trên toán học. Nhưng như đã trình bày các chiến lược thì khác giống nhau. Do đó, cho thấy cách thức này đã đạt ngách.

Chính vì thế, nhóm chúng em đề xuất hai thuật toán **Bio-HUIF** [12] và **MOEA-HEUPM** [13], 2 thuật toán này dựa trên cơ chế sinh học và tập tính động vật, kết hợp với các thuật toán tiến hóa như:

- **GA** (Genetic Algorithm),
- **PSO** (Particle Swarm Optimization),
- **BA** (Bat Algorithm).

Những thuật toán này kỳ vọng sẽ cải thiện hiệu quả khai phá và khắc phục các hạn chế hiện tại của các thuật toán trước.

REFERENCES

[1] R. S. R. Agrawal, "Fast algorithms for mining association," 1998. [Online]. Available: https://www.vldb.org/conf/1994/P487.PDF

[2] . Z. J. Grahne, G., "Fast algorithms for frequent itemset mining using fp-trees," 2005. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1501819

[3] U. Pyun, G. Yun, "Mining top-k frequent patterns with combination reducing techniques," 2014. [Online]. Available: https://link.springer.com/article/10.1007/s10489-013-0506-9

[4] B. V. L. T. T. N. P. F.-V. . A. S. Linh T., T. Nguyen, "Etarm: an efficient top-k association rule mining algorithm," 2018. [Online]. Available: https://link.springer.com/article/10.1007/s10489-017-1047-4

[5] P. F.-V. Xiangyu Liu, Xinzheng Niu, "Fast top-k association rule mining using rule generation property pruning," 2020. [Online]. Available: https://link.springer.com/article/10.1007/s10489-020-01994-9

[6] L. W. C. A. Liu, Y., "A two-phase algorithm for fast discovery of high utility itemsets," 2005. [Online]. Available: https://link.springer.com/chapter/10.1007/11430919_79

[7] H. T.-P.-L. W.-H. Lin, C.-W., "An effective tree structure for mining high utility itemsets," 2011. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0957417410014454

[8] R. Davashi, "Itufp: A fast method for interactive mining of top-k frequent patterns from uncertain data." 2023. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0957417422021741

[9] L. W. C.-A. Liu, Y., "A two-phase algorithm for fast discovery of high utility itemsets." 2005. [Online]. Available: https://link.springer.com/chapter/10.1007/11430919_79

[10] V. B. H.-V.-N. N.-N. B. S. Le, T., "Mining top-k frequent patterns from uncertain databases." 2020. [Online]. Available: https://link.springer.com/article/10.1007/s10489-019-01622-1

[11] T. L. a. Chun-Jung Chu a, Vincent S. Tseng b, "An efficient algorithm for mining high utility itemsets with negative item values in large databases," 2015. [Online]. Available: https://www.philippe-fournier-viger.com/spmf/huinivmine.pdf

[12] C. H. WEI SONG, "Mining high utility itemsets using bio-inspired algorithms: A diverse optimal value framework," 2018. [Online]. Available: https://www.researchgate.net/publication/324022168_Mining_High_Utility_Itemsets_Using_Bio-Inspired_Algorithms_A_Diverse_Optimal_Value_Framework

[13] U. A. J. C.-W. L. G. S. R. Y. Y. Djenouri, "An evolutionary model to mine high expected utility patterns from uncertain databases," 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9122027

Thuật toán UHUOPM

Input:
U: cơ sở dữ liệu không chắc chắn; H: cơ sở dữ liệu tiện ích
k: số lượng tập mẫu phổ biến không chắc chắn có tiện ích cao cần tìm.

Output:
k tập mẫu phổ biến không chắc chắn có tiện ích cao.

Bước 1: Đọc cơ sở dữ liệu U và H để tạo danh sách $PFUTABLE$ (1-mẫu).

Bước 2: Lấy k phần tử có $utility$ cao nhất rồi xếp theo TWU sau đó tính hàm hội tụ thông qua hàm `coverage` để tính toán khả năng khai phá của danh sách C

Bước 3: Tính các giá trị ngưỡng tiện ích:

$$\text{min\_utility} = \min(C_{\text{utility}})$$

$$\text{min\_pro} = \min(C_{\text{pro}})$$

Bước 4: Lọc ra các ứng viên thỏa mãn điều kiện thông qua hàm `judge(i, k)`: **Bước 5:** Sắp xếp lại danh sách $PFUTABLE$ (1-mẫu) giảm dần theo giá trị `pro`.

Bước 6: Gọi hàm đệ quy `PHUOP_Search(X, Candidates, k)` để tiếp tục tìm kiếm với danh sách ứng viên.

Thuật toán PHUOP_Search

Input:
P: một danh sách chứa các PFU, và X là tiền tố
k là số lượng UHUFP cần tìm.

Output:
k tập mẫu phổ biến không chắc chắn có tiện ích cao.

Nếu $|P| > 1$, thực hiện:

Với mỗi $Px \in P$:
Nếu $P_x.pro < min\_pro$:
Thuật toán tìm kiếm với Px tiếp theo
Nếu $P_x.utility + P_x.r\_utility - P_x.zrutils > minUtil$
extend $\leftarrow \emptyset$
Với mỗi $Py \in P, y > x$, thực hiện:
Nếu Px.pro < min_pro hoặc $lu(x, y)$ < min_pro:
Thuật toán dừng tìm kiếm
overestimate $\leftarrow Px$.expSup $\times Py$.max
over $\leftarrow Px$.utility + Py.maxUtility
Nếu overestimate < min_pro hoặc over < min_utility thì:
Tìm kiếm với Px tiếp theo
Kết thúc vòng lặp.
$du \leftarrow Py$.utility + Px.r_utility + PX.utility
Nếu $du > min\_utility$ thì:
xy $\leftarrow$ construct(x, y)
Nếu xy.pro > min_pro và xy.utility > min_utilitythì:
judge(xy, k)
Nếu xy.utility + xy.r_utility $\geq min\_utility$ và xy.pro $\geq min\_pro$ thì:
extend $\leftarrow$ extend $\cup$ xy
Kết thúc.
Kết thúc.
Nếu Py.utility + Py.r_utility $\geq min\_utility$ và Py.pro $> min\_pro$:
extend $\leftarrow$ extend $\cup$ y
Kết thúc.
Kết thúc.
Gọi lại PHUOP_Search(x, extend, k) để tiếp tục tìm kiếm.

Algorithm 2: Thuật Toán UHUOPM