BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC KINH TẾ THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG CÔNG NGHỆ VÀ THIẾT KẾ

ĐỒ ÁN MÔN HỌC

TÌM HIỂU THUẬT TOÁN ECLAT (EQUIVALENCE CLASS CLUSTERING AND BOTTOM-UP LATTICE TRAVERSAL) TRÊN BỘ DỮ LIỆU MARKET BASKET ANALYSIS 2

Lớp - Chuyên ngành : DS001 - Khoa Học Dữ Liệu
Khóa : K47
Lớp học phần : 23C1INF50904401
Môn học : Khai phá dữ liệu

Giảng viên: TS. Nguyễn An Tế

TP Hồ Chí Minh, 2023

THÀNH VIÊN NHÓM

STT	Họ tên	MSSV	Nội dung	Đóng góp
1	Đinh Trọng Hữu	31211027643	Tìm hiểu thuật toán Eclat	100%
2	Nguyễn Nhật Huy	31211027641	Tìm hiểu thuật toán Eclat	100%
3	Nguyễn King	31211023531	Ứng dụng thuật toán Eclat	100%
4	Lê Xuân Hưởng	35221020914	Ứng dụng thuật toán Eclat	100%
5	Võ Ngọc Mỹ Kim	31211027646	Tìm hiểu thuật toán Eclat	100%

Mục lục

Lời mở đầu

Trong thời đại công nghệ thông tin phát triển như vũ bão, lượng dữ liệu được tạo mỗi ngày là vô cùng khổng lồ. Theo thống kê "Annual size of real time data in the global datasphere from 2010 to 2025" của Petroc Taylor dự đoán thế giới sẽ tạo ra 175 zettabytes vào năm 2025, tức 175 nghìn tỷ terabyte. Với lượng dữ liệu đa dạng và khổng lồ đó, việc xử lý thông tin nhanh chóng để hỗ trợ đưa ra quyết định là một thách thức lớn đối với các doanh nghiệp và tổ chức. Từ đó những kỹ thuật và công cụ mới đã ra đời để đáp ứng nhu cầu, trong đó có khai phá dữ liệu (Data mining).

Khai phá dữ liệu là một lĩnh vực khoa học máy tính nhằm phát hiện các mô hình và mối quan hệ tiềm ẩn trong dữ liệu. Khai phá dữ liệu có thể được sử dụng để hỗ trợ các hoạt động kinh doanh, nghiên cứu khoa học, và nhiều lĩnh vực khác. Có nhiều kỹ thuật khai phá dữ liệu khác nhau, mỗi kỹ thuật có những ưu điểm và nhược điểm riêng. Trong bài báo cáo này sẽ nghiên cứu về thuật toán ECLAT, một thuật toán khai thác tập phổ biến hiệu quả và nhanh chóng. ECLAT được sử dụng để tìm ra các luật kết hợp mạnh, là các luật có độ tin cậy và độ hỗ trợ cao. Các luật kết hợp mạnh có thể được sử dụng để hiểu hành vi của khách hàng, dự đoán xu hướng thị trường, và nhiều ứng dụng khác.

CHƯƠNG 1: GIỚI THIỆU

1.1 Mục tiêu nghiên cứu

- Tìm hiểu khái quát về khai thác dữ liệu, các bước, các ứng dụng và thách thức của nó.
- Tìm hiểu sâu về thuật toán ECLAT: một thuật toán khai thác tập phổ biến hiệu quả và nhanh chóng.
- Khai phá dữ liệu và nghiên cứu xây dựng thuật toán dựa trên dữ liệu thực tế của Marketing Basket Analysis.

1.2 Đối tượng nghiên cứu và phạm vi nghiên cứu

- Thuật toán ECLAT: cách thức hoạt động và ứng dụng vào thực tế.
- Phạm vi nghiên cứu: Nghiên cứu giới hạn ở việc sử dụng thuật toán để khai thác các luật kết hợp mạnh từ bộ dữ liệu.

1.3 Phương pháp nghiên cứu và tài nguyên sử dụng

1.3.1 Phương pháp nghiên cứu

- Tìm hiểu và áp dụng ECLAT để tìm ra các luật kết hợp mạnh.
- So sánh và đánh giá kết quả đạt được.

1.3.2 Tài nguyên sử dụng

- Ngôn ngữ lập trình: Python.
- Các thư viện sử dụng: Pandas, Numpy, mlxtend, Time, Matplot, Seaborn...
- Bộ dữ liệu Marketing Basket Analysis 2 do giảng viên cung cấp và bộ Course nhóm tự chuẩn bị.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1 Khái niệm chung

2.1.1 Khái niệm khai phá dữ liệu

Khai phá dữ liệu (Data mining) là quá trình khám phá thông tin hữu ích tiềm ẩn bên trong lượng lớn dữ liệu. Data mining là một công cụ phân tích để phân tích dữ liệu, cho phép người dùng phân tích, phân loại và tóm tắt các mối quan hệ giữa dữ liệu. Nó bao gồm một số kỹ thuật phổ biến như gom cụm, luật kết hợp, hồi quy, v.v.

Khai phá dữ liệu là một bước của quá trình khai thác tri thức (*Knowledge Discovery Process*) [3], bao gồm các giai đoạn sau đây:

- Hiểu nghiệp vụ (Business Understanding): Đây là bước quan trọng nhất, bao gồm việc xác định mục tiêu, nhu cầu và nguồn dữ liệu của việc khai phá.
- Hiểu dữ liệu (Data Understanding): xác định dữ liệu cần thiết, cần dữ liệu gì để giải quyết bài toán. Bao gồm việc thu thập dữ liệu ban đầu, mô tả dữ liệu, khám phá dữ liệu và xác nhận chất lượng dữ liệu.
- Tiền xử lý (Data Preparation): bao gồm việc chọn lựa, làm sạch, xây dựng theo định dạng mong muốn.
- Mô hình hoá dữ liệu (Modeling): trực quan hoá bằng hình ảnh (vẽ biểu đồ biểu diễn dữ liệu và xác định mối quan hệ) và phân tích cụm (để xác định các biến có phù hợp với nhau).
- Đánh giá kết quả (Evaluation): kết quả của mô hình được đánh giá theo mục tiêu đã xác định trong giai đoạn đầu.
- Triển khai (Deployment): có thể sử dụng để xác nhận các giả thuyết đã từng có hoặc khám phá tri thức mới.

Một số ứng dụng của khai phá dữ liệu:

- Kinh doanh: dự đoán nguồn khách hàng thân thiết, tư vấn sản phẩm có liên quan...
- Tài chính: dự đoán khả năng trả nợ của khách hàng trước khi cho vay/tín dụng, phân nhóm, khoanh vùng thị trường, phát hiện gian lận...
- Chăm sóc sức khoẻ: tìm phương pháp điều trị hiệu quả...

2.1.2 Khái niệm khai phá luật kết hợp

Khai phá luật kết hợp (Association rule mining) là mối quan hệ kết hợp giữa các thuộc tính trong cơ sở dữ liệu.

Luật kết hợp còn được biểu diễn dưới dạng $X \Rightarrow Y$, trong đó:

- $X, Y \subset I$
- $X \cap Y = \emptyset$
- X là tiền đề và Y là hệ quả của luật.

Luật kết hợp thường được đánh giá dựa trên hai tiêu chí chính là độ hỗ trợ (support) và độ tin cậy (confidence). Trong đó:

- Độ hỗ trợ của $X \Rightarrow Y$: độ phổ biến của luật (trong D)

$$\text{sup}(X \Rightarrow Y) = P(X \cap Y) = \frac{freq(X \cap Y)}{|D|}$$

- Độ tin cậy của $X \Rightarrow Y$ là độ chính xác của luật.

$$\text{conf}(X \Rightarrow Y) = P(Y|X) = \frac{freq(X \cap Y)}{freq(X)}$$

Luật kết hợp được tìm ra cần có tính tin cậy, vì thế nó cần độ tin cậy vượt qua một ngưỡng tin cậy tối thiểu do người dùng đưa vào. Luật đáp ứng được ràng buộc của cả độ hỗ trợ và độ tin cậy được gọi là luật kết hợp "mạnh" (strong association rule):

$$\text{sup}(X \Rightarrow Y) \geq \text{minSup}$$

$$\text{conf}(X \Rightarrow Y) \geq \text{minConf}$$

Phần lớn các thuật toán khai thác luật kết hợp sẽ thực hiện hai bước như sau:

- Bước 1: Tìm tất cả tập phổ biến (frequent itemset) trong D, điều kiện: $freq(X) \geq minSup$ và $L_k =$ frequent k-itemsets
- Bước 2: Tạo ra các luật liên kết "mạnh" từ tập phổ biến vừa tìm được ở trên.

Các khái niệm có liên quan:

- Items: $I = \{I_1, I_2, I_3, \ldots, I_n\}$
- itemsets: $X \subseteq I$
- k - itemset: $X_k = \{I_1, I_2, I_3, \ldots, I_n\}$ với $k \leq n$
- Giao dịch (transaction): $\emptyset \neq T \subseteq I$ (items trong T được sắp xếp tăng dần)
- Database (cơ sở dữ liệu): $D = \{T_i\}$

2.1.3 Khái niệm về thuật toán có liên quan

Khái niệm về thuật toán Apriori

Thuật toán Apriori [1] được sử dụng để khai thác tập phổ biến và luật kết hợp. Thuật toán sử dụng phương pháp tìm kiếm theo chiều rộng (BFS), trong đó $k - itemsets$ được sử dụng để tìm $(k+1) - itemsets$.

Hai giai đoạn khai phá luật kết hợp:

- Bước 1: Khai phá những tập phổ biến.
  - Cách Apriori tìm tất cả X phổ biến bằng level-wise search:
    - Tạo $L_1 = \{\text{frequent 1-itemset}\}$
    - Mở rộng L_1 để tìm $L_2 = \{\text{frequent 2-itemset}\}$
    - Mở rộng L_2 để tìm $L_3 = \{\text{frequent 3-itemset}\}$
    - ...
    - Tiếp tục cho đến khi không tìm được k-itemset nào.
  - Phương pháp tạo L_k, mở rộng từ L_{k-1} ($k \geq 2$):
    - Công đoạn kết nối (join step) tạo C_k chứa các ứng viên.
      - Phép kết: $C_k = L_{k-1} \bowtie L_{k-1}$
      - Điều kiện kết hai tập ứng viên (dãy 2 mảng) $x, y \in L_{k-1}$:

$$x[i] = y[i] \quad \forall i < (k-2)$$

$$x[k-1] < y[k-1]$$

      - Kết quả $\times$ y = {x[1],..., x[k-1], y[k-1]}
      - Công đoạn cắt tỉa (prune step):

$$L_k = \{C \in C_k \mid \text{freq}(C) \geq \text{minSup}\}$$

- Bước 2: Phát sinh những luật kết hợp (mạnh) từ các tập phổ biến.

  Tạo tất cả các tập con X_i khác $\emptyset$ của X

  Tạo luật kết hợp: $X_i \Rightarrow (X - X_i)$ nếu $\frac{\text{freq}(X)}{\text{freq}(X_i)} \geq \text{minConf}$

Khái niệm về thuật toán FP-Growth

FP-Growth [2] biểu diễn dữ liệu của các giao dịch bằng một cấu trúc dữ liệu cây phổ biến gọi FP-tree. Cấu trúc này cô đọng các thông tin chính nhưng vẫn đủ để khai thác các tập phổ biến, giúp giảm không gian và thời gian xử lý so với Apriori.

Xây dựng cây mẫu phổ biến (FP-Tree): xây dựng từ CSDL

- Bước 1. Xây dựng F-list: duyệt CSDL (lần 1) để tạo L_1, sau đó sắp xếp L_1 giảm dần theo frequency.
- Bước 2. Xây dựng FP-tree và Item Header Table: duyệt CSDL (lần 2) với items trong mỗi transaction được sắp xếp theo thứ tự F-list.
- Bước 3. Duyệt "ngược" các item I_k trong E-F-list để tạo các tập phổ biến
  - Xử lý F-list theo thứ tự freq tăng dần
    - Tạo cơ sở mẫu điều kiện (Conditional Pattern Base - CPB) từ prefix của các paths đến item I_k.
    - Dùng CPB như CSDL để tạo FP-tree điều kiện
    - Phát sinh tập phổ biến từ mỗi path trong FP-tree điều kiện.

2.1.4 Khái niệm về thuật toán ECLAT

Thuật toán ECLAT (Equivalence Class Clustering and bottom-up Lattice Traversal) [6] là một thuật toán dựa trên tìm kiếm theo chiều sâu (Depth First search) được sử dụng để khai thác tập phổ biến và luật kết hợp. Nó sử dụng cấu trúc cơ sở dữ liệu dọc (vertical database layout), nghĩa là thay vì liệt kê tất cả giao dịch một cách rõ ràng thì mỗi mục được lưu trữ cùng với tập chủ của nó (titlist) và sử dụng phương pháp dựa trên giao điểm để tính toán độ hỗ trợ của itemset.

ECLAT đặc biệt phù hợp với các tập dữ liệu nhỏ, bởi vì nó yêu cầu ít thời gian và không gian hơn để tạo ra mô hình so với Apriori. ECLAT thực hiện tìm kiếm theo chiều sâu trên itemset, bắt đầu từ đỉnh. Đây là một sự khác biệt chính so với thuật toán Apriori, thực hiện tìm kiếm theo chiều rộng (breadth-first search).

Các khái niệm trong thuật toán ECLAT:

- TID (transaction ID): đại diện cho một giao dịch duy nhất $T_1, T_2, T_3, \ldots, T_n$ trong cơ sở dữ liệu giao dịch DB.
- Items: $I = \{I_1, I_2, I_3, \ldots, I_n\}$
- Itemsets: $X \subseteq I$
- k-itemset: $X_k = \{I_1, I_2, I_3, \ldots, I_k\}$ với $k \leq n$

Bên cạnh đó vì thuật toán ECLAT dựa trên tìm kiếm theo chiều sâu nên ở đây chúng ta cũng sẽ tìm hiểu khái quát về thuật toán tìm kiếm này. Tìm kiếm theo chiều sâu (Depth - First Search hay DFS) là một thuật toán duyệt hoặc tìm kiếm trên một cây hoặc một đồ thị. Thuật toán khởi đầu tại gốc (hoặc chọn một đỉnh nào đó coi như gốc) và phát triển xa nhất có thể theo mỗi nhánh.

Các bước thực hiện theo:

- Bước 1. Tìm một đỉnh chưa viếng thăm nằm liền kề với đỉnh hiện tại, đánh dấu nó là đã viếng thăm và thêm vào ngăn xếp.

- Bước 2. Nếu không thể tìm thấy một đỉnh liền kề chưa được viếng thăm với đỉnh hiện tại, loại bỏ một đỉnh khỏi ngăn xếp và bắt đầu lại.

- Bước 3. Nếu ở bước 2, ngăn xếp rỗng, thuật toán dừng.

2.2 Nguồn gốc thuật toán

Trong lĩnh vực Khai thác dữ liệu (Data Mining), việc tìm hiểu và phát triển các thuật toán khai thác mẫu kết hợp (Frequent Pattern Mining) là rất quan trọng. ECLAT (Equivalence Class Transformation for Lattice Traversing) là một thuật toán hiệu quả được thiết kế để khai thác các mẫu kết hợp từ dữ liệu có cấu trúc đa tầng.

Thuật toán ECLAT được giới thiệu bởi Pasquier, Bastide và Taouil trong bài báo khoa học "Efficient Mining of Association Rules using Closed Itemset Lattices"(1999). Ông Pasquier và đồng nghiệp đã đề xuất ECLAT nhằm cải thiện hiệu suất của thuật toán Apriori, một trong những thuật toán khai thác mẫu kết hợp đầu tiên và phổ biến.

ECLAT đã đóng góp vào việc phát triển lĩnh vực khai thác mẫu kết hợp bằng cách cải thiện thời gian chạy so với các thuật toán trước đó. Nó được ứng dụng rộng rãi trong nhiều lĩnh vực như bán lẻ, y tế, và nghiên cứu khoa học.

Nguồn gốc của thuật toán ECLAT có liên quan chặt chẽ đến nhu cầu cải thiện hiệu suất của các thuật toán khai thác mẫu kết hợp trong quá trình phân tích dữ liệu. Sự kết hợp giữa equivalence class và lattice structure đã giúp ECLAT trở thành một công cụ mạnh mẽ và linh hoạt, đặc biệt là khi áp dụng cho các tập dữ liệu có cấu trúc phức tạp.

2.3 Mô tả thuật toán Eclat

Thuật toán Eclat bắt đầu bằng việc xây dựng danh sách các itemsets tiềm năng và sử dụng lược đồ phân chia lớp tương đương (equivalent class) để giảm thiểu số lượng itemsets cần xét. Ngoài ra, thuật toán còn hoạt động theo tính đệ quy để kiểm tra tất cả các tập hợp con có thể có, đảm bảo tìm ra các tập phổ biến với độ dài bất kỳ. Qua từng bước đệ quy, các itemsets có độ hỗ trợ thấp sẽ bị loại bỏ giúp tối ưu hóa hiệu quả tìm kiếm.

2.3.1 Mã giả

Mã giả cho thuật toán ECLAT được trình bày như sau [5]:

Algorithm 1 ECLAT

Require: $F = \{I_1, I_2, \ldots, I_n\}$ frequent k Itemsets
Terminology:
(i) F_k is defined as database having $F_k = \{I_1, I_2, \ldots, I_n\}$
(ii) $\emptyset$ denotes the itemsets where itemsets means collection of items in database F_k
(iii) I_i and I_j both should be from same equivalence class
Ensure: F_R Frequent Item Sets
Bottom-Up (F_k):
for all $I_i \in F_k$ **do**
  $F_{k+1} = \emptyset$
  for all $I_j \in F_k$ where $I_i \neq I_j$ **do**
    $N = I_i \cap I_j$; //Both should be from same equivalence class
    if N.sup $\geq$ minsup **then**
      $F_{k+1} = F_{k+1} \cup \{N\}$; $F_R = F_R \cup \{N\}$
    end if
  end for
  if $F_{k+1} = \emptyset$ **then**
    return Bottom-Up (F_{k+1})
  end if
end for=0

2.3.2 Giai đoạn thực hiện

Input: Danh sách các mẫu frequent k-item

Ouput: Danh sách tất cả các tập phổ biến được tìm thấy

Giai đoạn 1: Khởi tạo - xây dựng danh sách các itemset có thể kết hợp

- Bước 1: Duyệt qua từng giao dịch trong tập dữ liệu, sau đó biến đổi CSDL từ định dạng theo chiều ngang thành định dang theo chiều dọc và đếm số lần xuất hiện của mỗi item. Các item có tần suất thấp hơn mức hỗ trợ tối thiểu (minSup) sẽ bị loại bỏ.
- Bước 2: Khởi tạo một danh sách rỗng để chứa các itemsets tiềm năng trước khi tìm kiếm các mẫu frequent mới

Giai đoạn 2: Đệ quy tìm kiếm các tập phổ biến:

- Bước 3: Thuật toán sẽ lấy item đầu tiên từ danh sách itemsets tiềm năng và sử dụng nó để tạo ra các itemsets con.
- Bước 4: Tạo danh sách itemsets con: Duyệt qua từng lớp tương đương, lọc các item từ danh sách itemsets tiềm năng sao cho mỗi item xuất hiện trong ít nhất một giao dịch chung với item vừa chọn. Tạo các itemsets con mới bằng cách kết hợp item vừa chọn với từng item còn lại trong danh sách lọc.
- Bước 5: Tính độ hỗ trợ cho các itemsets con: Kiểm tra từng giao dịch trong tập dữ liệu để xem item có xuất hiện và tạo danh sách các giao dịch chứa item đó. Tính độ hỗ trợ (số giao dịch chứa item trên tổng số giao dịch) của itemset mới bao gồm item vừa chọn

- Bước 6: Kiểm tra độ hỗ trợ cho các itemsets con: Nếu độ hỗ trợ của một itemsets con lớn hơn hoặc bằng minSup, thì itemset đó được coi là một tập phổ biến và thêm vào danh sách kết quả.

- Bước 7: Chạy đệ quy cho các itemsets con: Gọi đệ quy cho từng itemsets con với danh sách itemsets con mới và danh sách giao dịch chung với item gốc. Quá trình này lặp lại cho đến khi không còn itemsets nào có thể kết hợp.

2.3.3 Ví dụ

Ta có tập Cơ sở dữ liệu như sau:

Transaction_ID	Items
T1	Lassi, Cheese
T2	Ghee, Lassi, Cheese, Coffee
T3	Lassi, Coffee
T4	Cheese, Lassi, Coffee
T5	Coffee, Lassi

Đầu tiên thực hiện chuyển CSDL từ định dạng ngang theo định dạng dọc:

Item	Transaction_ID_list
Lassi	T1,T2,T3,T4,T5
Coffee	T2,T3,T4
Cheese	T1,T2,T4
Ghee	T2

Giả sử minSup = 2, ta loại bỏ các item có tần suất xuất hiện bé hơn 2 (bé hơn minSup)

Item	Transaction_ID_list
Lassi	T1, T2, T3, T4, T5
Coffee	T2, T3, T4
Cheese	T1, T2, T4

Chọn 1 item để thực hiện nhóm với các itemset của lớp tương đương riêng lẻ thứ 1. Lọc các item xuất hiện trong ít nhất một giao dịch chung với item vừa chọn. Kiểm tra có thoã mức độ hỗ trợ tối thiểu (minSup = 2) không. Nếu thoã sẽ thêm vào lớp tương đương thứ 2

Item	Transaction_ID_list
Lassi, Coffee	T2, T3, T4
Lassi, Cheese	T1, T2, T4

Tiếp tục thực hiện đệ quy trong lớp tương đương này cho đến hết item

Item	Transaction_ID_list
Coffee, Cheese	T2, T4

Sau khi hết lớp tương đương thứ 2, ta sẽ tiếp tục tương tự đối với lớp tương đương thứ 3

Item	Transaction_ID_list
Lassi, Coffee, Cheese	T2, T4

Kết quả: {Lassi}, {Coffee}, {Cheese}, {Lassi, Coffee}, {Lassi, Cheese}, {Lassi, Coffee, Cheese}

2.4 Đặc điểm của thuật toán

Thuật toán ECLAT khai thác các tập phổ biến với cơ sở dữ liệu có định dạng dọc. Trong số các thuật toán khác dùng để khai thác các tập phổ biến là Apriori và FP-Growth, 2 thuật toán này khai thác dựa tập dữ liệu dịnh dạng ngang.

Thể hiện 2 dạng dữ liệu nói trên như sau:

Dữ liệu dạng ngang	
transaction_id	items
0	A, B, D
1	B, D, F
2	A, B, F
3	B, D
4	E, F

Dữ liệu dạng dọc	
itemset	transaction_id_list
A	0, 2
B	0, 1, 2, 3
E	4
F	1, 2, 4

Thuật toán ECLAT sử dụng cấu trúc dữ liệu Lattice và Equivalence Class (EC) khai thác mẫu kết hợp một cách hiệu quả:

Cấu trúc dữ liệu Lattice (Cấu trúc phân cấp):

Lattice là một biểu đồ hình cây có các nút đại diện cho một tập hợp con của mẫu kết hợp. Nút gốc của cấy biểu thị tập hợp rỗng, và mỗi nút lá đại diện cho một mẫu kết hợp đơn.

Lattice được sử dụng để biểu diễn tất cả các mẫu kết hợp và mối quan hệ giữa chúng. Cấu trúc Lattice giúp thuật toán tự động khám phá mẫu kết hợp mà không cần phải duyệt qua toàn bộ không gian tìm kiếm, điều này giúp giảm độ phức tạp của thuật toán so với các phương pháp truyền thống.

Equivalence Class (Lớp tương đương):

Mỗi nút trong Lattice thường được liên kết với Equivalence Class, đại diện cho một tập hợp các đối tượng có giá trị tương đương đối với các thuộc tính quan trọng. Equivalence Class giúp giảm số lượng giao dịch cần xem xét, tăng tốc quá trình khai thác kết hợp và giảm độ phức tạp của thuật toán. Equivalence Class đại diện cho tập hợp các đối tượng có cùng giá trị của các thuộc tính quan

trọng. Equivalence Class được sử dụng để xác định mẫu kết hợp đóng (closed itemsets), là những mẫu không thể mở rộng thêm mà không làm thay đổi hỗ trợ.

Closed Itemsets (mẫu kết hợp đóng): là các tập mẫu kết hợp không thể mở rộng thêm 1 cách có ý nghĩa mà vẫn giữ nguyên giá trị hỗ trợ (có thể mở rộng khi giá trị hỗ trợ không đổi). Thuật toán sử dụng Lattice và Equivalence Class trên cấu trúc dữ liệu dọc để nhanh chóng xác định các mẫu kết hợp đóng (closed itemsets) một cách hiệu quả.

ECLAT được áp dụng rộng rãi trong nhiều lĩnh vực như bán lẻ, y tế, và nghiên cứu khoa học. Sự linh hoạt và ứng dụng rộng rãi của ECLAT làm cho nó trở thành một công cụ mạnh mẽ trong lĩnh vực Data Mining.

2.5 Ưu điểm và nhược điểm

2.5.1 Ưu điểm

Thuật toán Eclat có thể chạy nhanh hơn nhiều so với các thuật toán khác. Điều này là do thuật toán Eclat chỉ cần duyệt qua các giao dịch một lần, trong khi các thuật toán duyệt toàn bộ cần duyệt qua các giao dịch nhiều lần. Ngoài ra, thuật toán còn có thể được triển khai trên nhiều bộ xử lý bằng cách chia các tập dữ liệu thành các phần nhỏ hơn và chạy thuật toán trên mỗi phần. Điều này giúp thuật toán Eclat có thể xử lý các tập dữ liệu lớn mà không cần thêm bộ nhớ.

2.5.2 Nhược điểm

- Có thể bỏ sót các tập phổ biến: Thuật toán ECLAT có thể bỏ sót các tập phổ biến nếu ngưỡng hỗ trợ cho trước quá cao.
- Phức tạp tính toán đối với tập có độ dài lớn: Khi xử lý các tập có độ dài lớn, Eclat có thể trở nên phức tạp tính toán do số lượng kết hợp cực kỳ lớn và sự gia tăng nhanh chóng về khả năng kết hợp.

CHƯƠNG 3: THỰC NGHIỆM VÀ ĐÁNH GIÁ KẾT QUẢ

3.1 Bài toán thực nghiệm

Bài toán phân tích dữ liệu để xác định các các sản phẩm phổ biến trong các giao dịch giúp chúng ta xác định tập hợp các sản phẩm mà người mua thường xuyên mua cùng nhau.

3.2 Bộ dữ liệu sử dụng

Bộ dữ liệu Marketing Basket Analysis 2 do giảng viên cung cấp trong đó có 1 cột Items là Các cột này có thể chứa tên các sản phẩm hoặc mục được mua trong mỗi giao dịch. Ngoài ra, nhóm cũng sử dụng bộ dữ liệu "Courses"nhằm so sánh hiệu suất của thuật toán ECLAT trên nhiều bộ dữ liệu với cấu trúc khác nhau.

3.3 Cài đặt thuật toán

Bước 1: Import file vào hệ thống

```
!pip install -q mlxtend
!gdown 1a47mER2sDPvDzZsgQhiFzhtv-uU5o_x2
!gdown 1AScqLzxGHsaMBExWHV3idPZWZTCL1nq5
```

Bước 2: Thêm các thư viện cần thiết cho việc chạy thuật toán

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Bước 3: Đọc dữ liệu từ file Maketing Basket Analysis 2 do giảng viên cung cấp

```
def read_data(filename):
    with open(filename, 'r') as file:
        transactions = [line.strip().split(',')[:-1] for line in file]
    return transactions
transactions = read_data('Market Basket Analysis 2.csv')
datasets.append(transactions)
```

Bước 4: Viết các function cho thuật toán

Bước 4.1: Xử lý dữ liệu cho thuật toán, tạo lớp tương đương

```
def processing_data_eclat(transactions, min_support):
    # Đếm tổng số giao dịch
    transactions_count = len(transactions)
    item_transactions = {}

    # Lưu lại index của các giao dịch mà item xuất hiện
    for index, transaction in enumerate(transactions):
        for item in transaction:
            if item in item_transactions:
                item_transactions[item].add(index)
            else:
                item_transactions[item] = {index}

    # Lọc các items có support >= minsup
    items = [(item, indices) for item, indices in item_transactions.items()
             if len(indices) / transactions_count >= min_support]

    return items, transactions_count
```

Bước 4.2: Viết thuật toán ECLAT dùng đệ quy

```
def eclat(prefix, items, frequent_itemsets, min_support,
↪  transactions_count):
    # Duyệt qua các items còn lại
    while items:
        # Trích xuất item và các transactions mà item có xuất hiện
        item, item_transactions = items.pop()
        itemset = prefix | {item}
        support = len(item_transactions) / transactions_count

        if support >= min_support:
            frequent_itemsets.append((itemset, support))

            suffix_items = []
```

```
            for next_item, next_transactions in items:
                intersection = item_transactions & next_transactions
                if intersection:
                    suffix_items.append((next_item, intersection))

            # Chạy đệ quy
            eclat(itemset, suffix_items, frequent_itemsets, min_support,
            ↪  transactions_count)
```

Bước 4.3: Lấy ra tập phổ biến

```
def run_eclat(items, transactions_count):
    # Khai phá các tập phổ biến
    frequent_itemsets = []
    eclat(set(), items, frequent_itemsets, min_support, transactions_count)
    return frequent_itemsets
```

Bước 4.4: Chạy thuật toán ECLAT

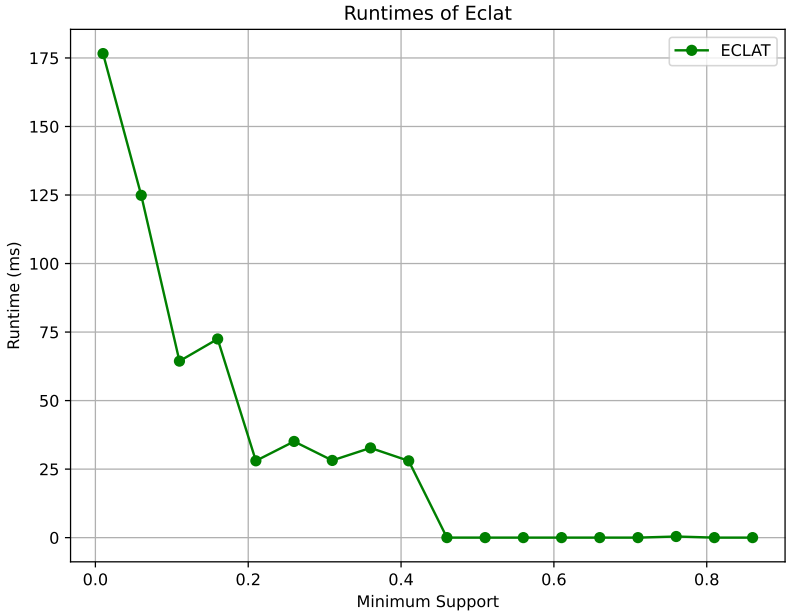
```
eclat_items, transactions_count = processing_data_eclat(transactions,
↪  min_support)
eclat_frequent_itemsets = run_eclat(eclat_items, transactions_count)
```

Để kiểm tra Min Support ta chạy lần lượt min_sup từ 0.01 đến 0.9 mỗi bước 0.05 rồi vẽ lên biểu đồ xem nên dùng min_support nào tốt

```
min_support_values = np.arange(0.01, 0.9, 0.05)
for min_support in min_support_values:
    eclat_items, transactions_count = processing_data_eclat(transactions,
    ↪  min_support)
    start_time = time.time()
    eclat_frequent_itemsets = run_eclat(eclat_items, transactions_count)
    eclat_time = time.time() - start_time
    eclat_runtimes.append(eclat_time * 1000)

# Plotting the runtimes
plt.figure(figsize=(8, 6))
```

```
plt.plot(min_support_values, eclat_runtimes, marker='o', label='ECLAT',
    color='g')

plt.title('Runtimes of Eclat')
plt.xlabel('Minimum Support')
plt.ylabel('Runtime (ms)')
plt.legend()
plt.grid(True)
plt.show()
```

Kết quả:



Hình 1: Biểu đồ thể hiện thời gian chạy của thuật toán ECLAT trên các minsup khác nhau

Nhận xét: Biểu đồ 1 bắt đầu với thời gian chạy cao ở ngưỡng minsup thấp nhất ($minsup = 0.01$). Khi minsup tăng lên, thời gian chạy giảm một cách nhanh chóng, cho thấy rằng thuật toán xử lý ít các itemsets hơn và hoàn thành nhanh hơn khi có ít itemsets đáp ứng tiêu chí về tần suất xuất hiện.

3.4 So sánh hiệu suất của thuật toán ECLAT so với Apriori và FP-Growth

Để so sánh hiệu suất của thuật toán ECLAT với các thuật toán Apriori và FP-Growth ta dùng thêm thư viện mlxtend để chạy thuật toán Apriori và FP-Growth

3.4.1 So sánh về thời gian chạy

Cố định 1 seed

Bước 1: Chạy các thuật toán để so sánh

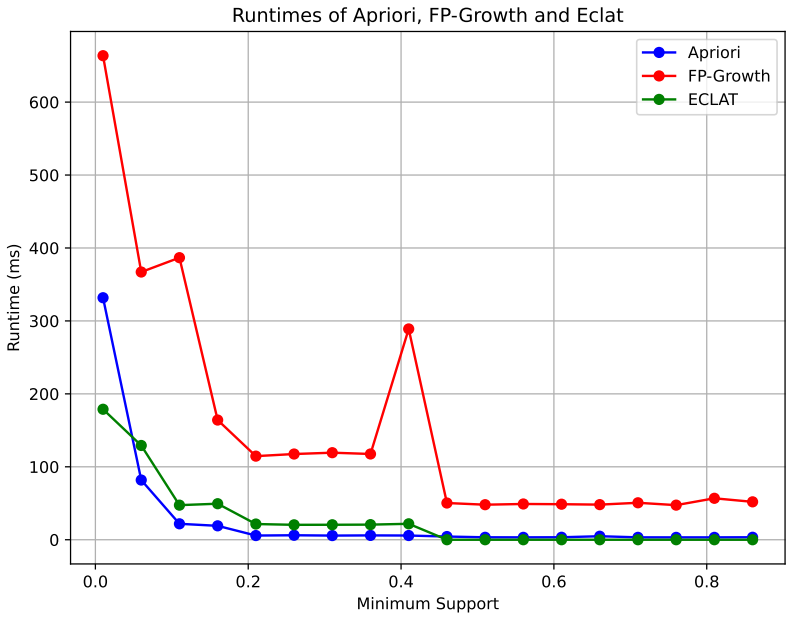
```
for min_support in min_support_values:
    eclat_items, transactions_count = processing_data_eclat(transactions,
    ↪ min_support)

    # Measure runtime for Apriori
    start_time = time.time()
    apriori_frequent_itemsets = apriori(df, min_support=min_support,
    ↪ use_colnames=True)
    apriori_time = time.time() - start_time
    apriori_runtimes.append(apriori_time * 1000)
    # print(f'apriori: {len(apriori_frequent_itemsets)}')

    # Measure runtime for FP-Growth
    start_time = time.time()
    fpgrowth_frequent_itemsets = fpgrowth(df, min_support=min_support,
    ↪ use_colnames=True)
    fpgrowth_time = time.time() - start_time
    fpgrowth_runtimes.append(fpgrowth_time * 1000)
    # print(f'fpgrowth: {len(fpgrowth_frequent_itemsets)}')

    start_time = time.time()
    eclat_frequent_itemsets = optimized_eclat(eclat_items,
    ↪ transactions_count)
    eclat_time = time.time() - start_time
    eclat_runtimes.append(eclat_time * 1000)
    # print(f'eclat: {len(eclat_frequent_itemsets)}')
```

Bước 2: Vẽ biểu đồ

```
# Plotting the runtimes
plt.figure(figsize=(8, 6))
```

```
plt.plot(min_support_values, apriori_runtimes, marker='o', label='Apriori',
    color='b')
plt.plot(min_support_values, fpgrowth_runtimes, marker='o',
    label='FP-Growth', color='r')
plt.plot(min_support_values, eclat_runtimes, marker='o', label='ECLAT',
    color='g')
plt.title('Runtimes of Apriori, FP-Growth and Eclat')
plt.xlabel('Minimum Support')
plt.ylabel('Runtime (ms)')
plt.legend()
plt.grid(True)
plt.show()
```

Kết quả



Hình 2: Biểu đồ so sánh thuật toán ECLAT với 2 thuật toán FP-Growth và Apriori

Nhận xét: Ở biểu đồ 2, trục y thể hiện thời gian chạy (đo bằng miliseconds - ms), và trục x thể hiện ngưỡng hỗ trợ tối thiểu (minimum support). Khi ngưỡng hỗ trợ tối thiểu tăng lên, thời gian chạy của cả ba thuật toán đều giảm vì số lượng các itemsets phổ biến cần xét giảm đi. Mỗi thuật toán có đặc điểm riêng khi xử lý dữ liệu: Apriori cần duyệt nhiều candidate set, FP-Growth xây dựng và duyệt qua cây FP, ECLAT xử lý trực tiếp trên tập dữ liệu dạng đứng và sự chênh lệch về thời gian chạy giữa các thuật toán phản ánh cách tiếp cận cơ bản của chúng đối với việc tìm kiếm và xử lý itemsets phổ biến.

- **Apriori:** Thuật toán Apriori thường có thời gian chạy cao khi ngưỡng hỗ trợ tối thiểu thấp vì nó cần xét qua tất cả các kết hợp có thể của các item. Khi ngưỡng hỗ trợ tăng lên, số lượng itemsets cần xét giảm đi, dẫn đến việc giảm thời gian chạy. Tuy nhiên, do cách thức duyệt tất cả các kết hợp có thể, Apriori không hiệu quả như FP-Growth hay ECLAT, đặc biệt ở những bước đầu của quá trình tìm kiếm.
- **FP-Growth:** FP-Growth thường hiệu quả hơn Apriori vì nó sử dụng cấu trúc cây để lưu trữ thông tin và tìm itemsets phổ biến mà không cần tạo ra các kết hợp candidate. Điều này giúp giảm đáng kể thời gian chạy, nhất là khi ngưỡng hỗ trợ tối thiểu cao. Tuy nhiên, có thể thấy rằng ở một số điểm, FP-Growth có sự tăng đột biến về thời gian chạy, có thể do đặc thù của bộ dữ liệu, sự khác biệt về mặt tối ưu thuật toán trong thư viện hoặc việc xây dựng cây FP có thể tốn kém hơn trong một số trường hợp cụ thể.
- **ECLAT:** ECLAT là thuật toán hiệu quả với việc sử dụng cấu trúc dữ liệu dạng đứng để nhanh chóng tìm các tập phổ biến thông qua giao của các tập danh sách tid. Điều này thường nhanh hơn vì không cần duyệt qua cây như FP-Growth hay tạo ra các kết hợp candidate như Apriori. Trong biểu đồ, ECLAT thể hiện thời gian chạy thấp và ổn định hơn so với hai thuật toán kia, phản ánh hiệu quả của việc sử dụng cấu trúc dữ liệu dạng đứng trong việc giảm bớt các phép toán không cần thiết.

Ta tiếp tục so sánh các thuật toán khi chạy các seed khác nhau:

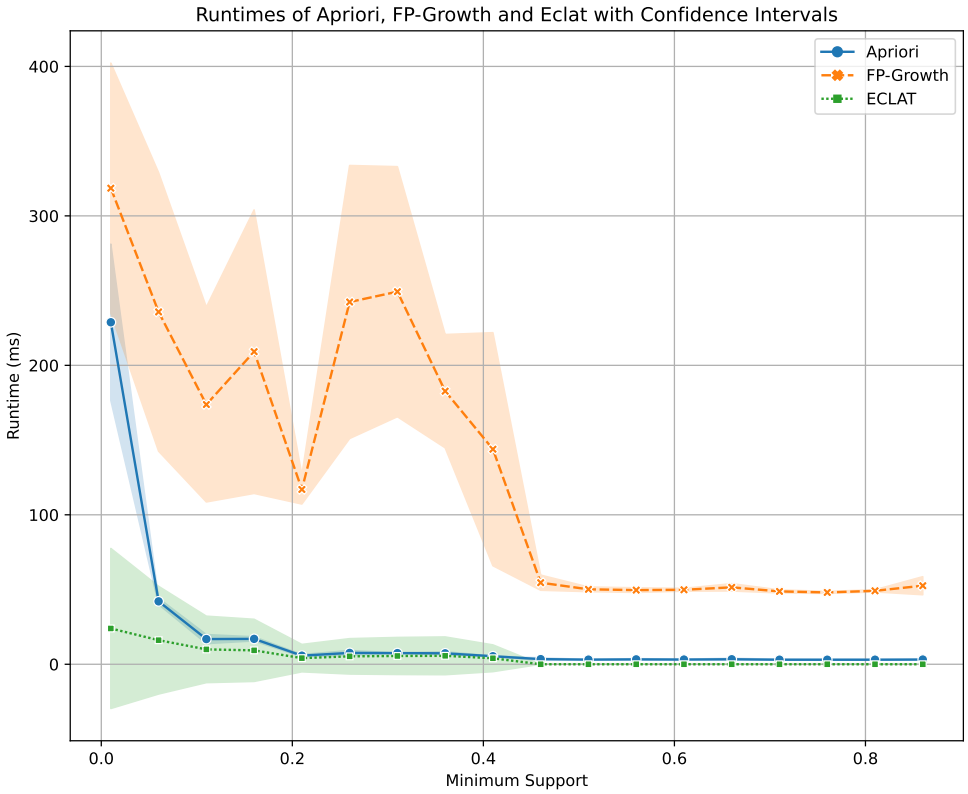
```
    for seed in range(num_seeds):
        np.random.seed(seed)

        # Measure runtime for Apriori
        start_time = time.time()
        apriori_frequent_itemsets = apriori(df, min_support=min_support,
        ↪  use_colnames=True)
        apriori_time = time.time() - start_time
        runtimes['algorithm'].append('Apriori')
        runtimes['min_support'].append(min_support)
        runtimes['runtime'].append(apriori_time * 1000)
        runtimes['seed'].append(seed)

        # Measure runtime for FP-Growth
        start_time = time.time()
        fpgrowth_frequent_itemsets = fpgrowth(df, min_support=min_support,
        ↪  use_colnames=True)
```

```
        fpgrowth_time = time.time() - start_time
        runtimes['algorithm'].append('FP-Growth')
        runtimes['min_support'].append(min_support)
        runtimes['runtime'].append(fpgrowth_time * 1000)
        runtimes['seed'].append(seed)

        # Measure runtime for ECLAT
        start_time = time.time()
        eclat_frequent_itemsets = optimized_eclat(eclat_items,
        ↪  transactions_count)
        eclat_time = time.time() - start_time
        runtimes['algorithm'].append('ECLAT')
        runtimes['min_support'].append(min_support)
        runtimes['runtime'].append(eclat_time * 1000)
        runtimes['seed'].append(seed)
```

Vẽ biểu đồ

```
# Convert the runtimes to a DataFrame
runtimes_df = pd.DataFrame(runtimes)

# Create a line plot with confidence intervals using seaborn
plt.figure(figsize=(10, 8))
sns.lineplot(data=runtimes_df, x='min_support', y='runtime',
↪  hue='algorithm', style='algorithm', markers=True, errorbar='sd',
err_style='band')

plt.title('Runtimes of Apriori, FP-Growth and Eclat with Confidence
↪  Intervals')
plt.xlabel('Minimum Support')
plt.ylabel('Runtime (ms)')
plt.legend()
plt.grid(True)
plt.show()
```

Nhận xét: Biểu đồ 3 cho thấy sự biến động lớn về thời gian chạy cho cả ba thuật toán khi thay đổi ngưỡng hỗ trợ tối thiểu. Từ biểu đồ này ta rút ra được các ý chính như sau:



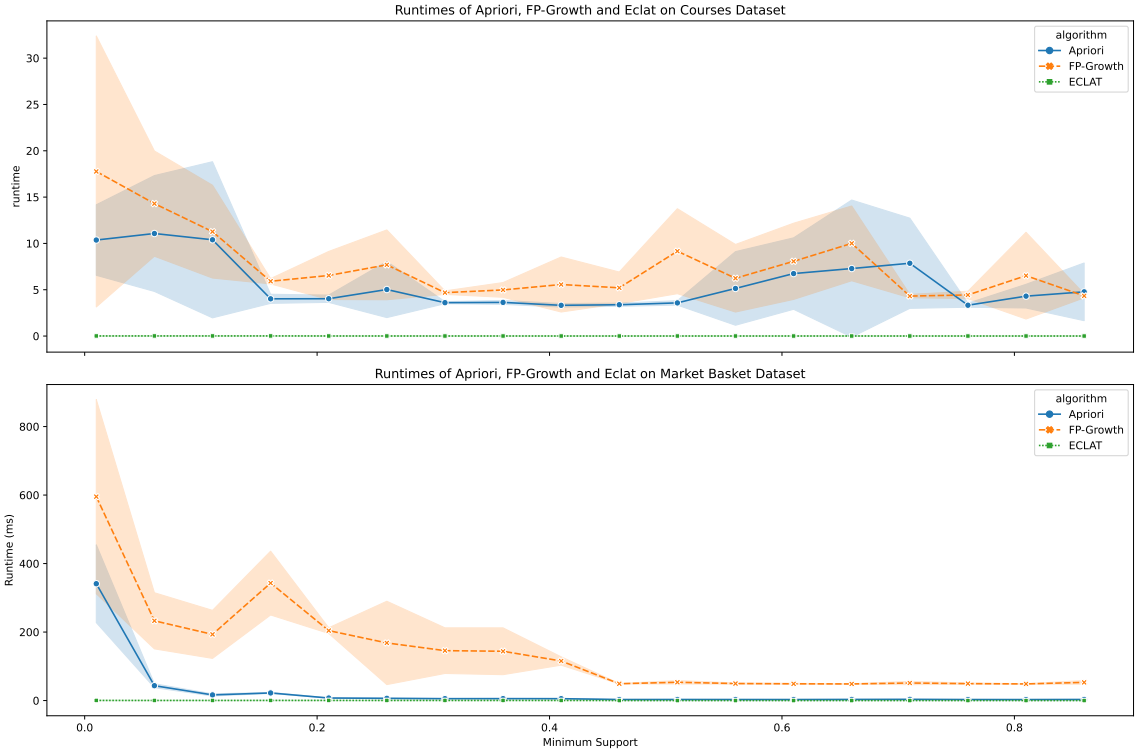
Hình 3: Biểu đồ thể hiện thời gian chạy của 3 thuật toán Apriori, FP-Growth và ECLAT

- **Apriori:** Đây là thuật toán dựa trên cơ sở tạo ra và kiểm tra các candidate itemsets thông qua quá trình lọc lặp đi lặp lại, nên thời gian chạy giảm nhanh chóng khi các itemsets ít phổ biến hơn được loại bỏ sớm.
- **ECLAT:** Biểu đồ cho thấy ECLAT có thời gian chạy thấp và ít biến động hơn, phản ánh khả năng tối ưu hóa tính toán của nó.
- Các dải màu xung quanh các đường biểu diễn cho thấy độ biến động trong thời gian chạy giữa các lần chạy khác nhau (các seeds khác nhau), cho thấy sự không ổn định có thể xảy ra do tính ngẫu nhiên trong bộ dữ liệu hoặc do các thay đổi nhỏ trong cách triển khai thuật toán.

Cuối cùng, ta thực hiện so sánh kết quả chạy của ECLAT so với 2 thuật toán còn lại là Apriori và FP-Growth trên 2 bộ dữ liệu khác nhau là "Market Basket Analysis 2"và "Courses". Mỗi thuật toán được chạy nhiều lần với các giá trị seed khác nhau để tính toán thời gian chạy, và ngưỡng hỗ trợ tối thiểu được thay đổi từ 0.01 đến 0.9 (hình 4).

Giải thích

Biểu đồ 1: Courses Dataset



Hình 4: Biểu đồ thể hiện thời gian chạy của 3 thuật toán Apriori, FP-Growth và ECLAT trên 2 bộ dữ liệu khác nhau

- **Biến động thời gian chạy:** Thời gian chạy cho mỗi thuật toán biến động rõ rệt, với khoảng sai số rộng, cho thấy sự không ổn định trong thời gian chạy giữa các lần chạy khác nhau. Điều này có thể do đặc tính của bộ dữ liệu hoặc sự biến động trong cấu trúc dữ liệu khi áp dụng các giá trị seed khác nhau.
- **So sánh giữa các thuật toán:** ECLAT có vẻ như là thuật toán ổn định nhất với thời gian chạy thấp, trong khi Apriori và FP-Growth có thời gian chạy cao hơn và biến động hơn.

Biểu Đồ 2: Market Basket Dataset

- **Thời gian chạy đối với bộ dữ liệu lớn hơn:** Tất cả các thuật toán đều có thời gian chạy cao hơn đáng kể so với bộ dữ liệu "Courses", điều này có thể chỉ ra rằng bộ dữ liệu "Market Basket"lớn hơn hoặc phức tạp hơn.
- **Apriori:** Thuật toán này có thời gian chạy giảm mạnh khi ngưỡng hỗ trợ tăng lên, điều này phản ánh đặc điểm của Apriori là phải duyệt qua nhiều candidate set hơn, đặc biệt khi ngưỡng hỗ trợ tối thiểu thấp.
- **FP-Growth và ECLAT:** Cả hai thuật toán này đều cho thấy thời gian chạy thấp hơn so với Apriori và có sự biến động thời gian chạy khi thay đổi ngưỡng hỗ trợ. Sự biến động này có thể phản ánh sự khác biệt trong cách mà các thuật toán xử lý dữ liệu và tối ưu hóa các bước tìm kiếm các itemsets phổ biến.

Kết luận, 2 biểu đồ cho thấy sự ảnh hưởng của ngưỡng hỗ trợ (minsup) đối với thời gian chạy của mỗi thuật toán và sự khác biệt về hiệu suất khi áp dụng chúng trên các bộ dữ liệu khác nhau. Điều này cung cấp thông tin quan trọng cho việc lựa chọn thuật toán phù hợp dựa trên đặc tính cụ thể của bộ dữ liệu và mục tiêu phân tích.

3.4.2 So sánh về bộ nhớ sử dụng

Để so sánh về bộ nhớ sử dụng ta có thể cài đặt thêm thư viện memory_profiler

```
!pip install -q memory_profiler
```

Bước 1: Tạo function để đo bộ nhớ sử dụng cho từng thuật toán

```
# Define a function to measure memory usage for each algorithm
def measure_memory(df, min_support, algorithm, eclat_items,
↪  transactions_count):
    if algorithm == 'Apriori':
        mem_usage = memory_usage((apriori, (df,), {'min_support':
        ↪  min_support, 'use_colnames': True}), max_usage=True)
    elif algorithm == 'FP-Growth':
        mem_usage = memory_usage((fpgrowth, (df,), {'min_support':
        ↪  min_support, 'use_colnames': True}), max_usage=True)
    elif algorithm == 'ECLAT':
        mem_usage = memory_usage((optimized_eclat, (eclat_items,
        ↪  transactions_count)), max_usage=True)
    else:
        mem_usage = [0]
    return max([mem_usage])
```

Bước 2: Tạo function để đo bộ nhớ sử dụng cho từng bộ dataset

```
# Simulate the memory usage measurement for each dataset
def simulate_memory_usage(df, dataset_name, transactions):
    for min_support in min_support_values:
        eclat_items, transactions_count =
        ↪  processing_data_eclat(transactions, min_support)
        for algorithm in ['Apriori', 'FP-Growth', 'ECLAT']:
            # Since we can't run the actual memory usage measurement here,
```

```
            # let's assume we run the measure_memory function and get the
            ↪ result.
            mem_usage = measure_memory(df, min_support, algorithm,
            ↪ eclat_items, transactions_count)
            memory_usages['algorithm'].append(algorithm)
            memory_usages['min_support'].append(min_support)
            memory_usages['memory_usage'].append(mem_usage)
            memory_usages['dataset'].append(dataset_name)
```

Bước 3: Chạy function để đo sử dụng bộ nhớ cho từng dataset

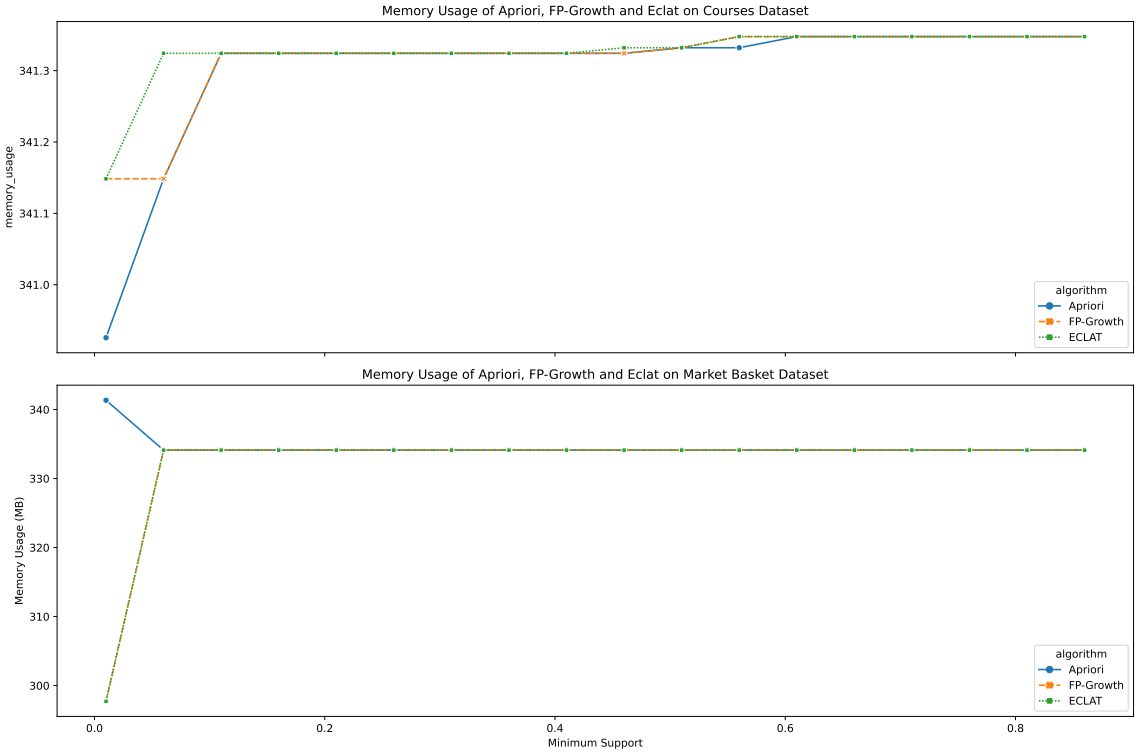
```
# Here you would call the simulate_memory_usage function for both datasets
simulate_memory_usage(df1, 'Courses Dataset', datasets[0])
simulate_memory_usage(df2, 'Market Basket Dataset', datasets[1])
```

Bước 4: Vẽ biểu đồ

```
# Convert the memory usages to a DataFrame
memory_usages_df = pd.DataFrame(memory_usages)
# Plotting the memory usages with error bands using seaborn
fig, axs = plt.subplots(2, 1, figsize=(15, 10), sharex=True)

# Courses Dataset
sns.lineplot(ax=axs[0], data=memory_usages_df[memory_usages_df['dataset'] ==
↪ 'Courses Dataset'],
             x='min_support', y='memory_usage', hue='algorithm',
             ↪ style='algorithm',
             markers=True, errorbar='sd', err_style='band')
axs[0].set_title('Memory Usage of Apriori, FP-Growth and Eclat on Courses
↪ Dataset')

# Market Basket Dataset
sns.lineplot(ax=axs[1], data=memory_usages_df[memory_usages_df['dataset'] ==
↪ 'Market Basket Dataset'],
             x='min_support', y='memory_usage', hue='algorithm',
             ↪ style='algorithm',
             markers=True, errorbar='sd', err_style='band')
```

```
axs[1].set_title('Memory Usage of Apriori, FP-Growth and Eclat on Market
↪  Basket Dataset')

plt.xlabel('Minimum Support')
plt.ylabel('Memory Usage (MB)')
plt.tight_layout()
plt.show()
```

Kết quả



Nhận xét: Từ biểu đồ ta thấy khi khởi tạo thì phần sử dụng bộ nhớ có khác nhau nhưng khi chạy với các min_sup tiếp theo thì bộ nhớ sử dụng tương đối bằng nhau không có sự khác biệt lớn.

CHƯƠNG 4: KẾT LUẬN VÀ KHUYẾN NGHỊ

4.1 Kết luận

Như vậy, qua thực nghiệm và so sánh trên tập dữ liệu Marketing Basket Analysis, có thể thấy thuật toán ECLAT là một lựa chọn hiệu quả để khai thác các luật kết hợp mạnh từ dữ liệu. Thuật toán đã chứng minh được ưu điểm về tốc độ xử lý và khả năng mở rộng so với các thuật toán truyền thống như Apriori và FP-Growth, đặc biệt ở các giá trị có minSup thấp.

Về mặt lý thuyết, cơ chế hoạt động dựa trên Lattice Structure và Equivalence Class giúp ECLAT vượt trội hơn hẳn. Thêm vào đó, định dạng dữ liệu dọc tiết kiệm được bộ nhớ đáng kể so với dạng ngang của các thuật toán khác. Tính đệ quy cũng góp phần giảm thiểu số itemset tạm thời phải lưu trữ.

4.2 Khuyến nghị

Dựa trên ưu và nhược điểm của thuật toán ECLAT, có một số hướng có thể phát triển và cải thiện.

Đầu tiên là tăng tốc độ xử lý và khả năng mở rộng. ECLAT chỉ cần duyệt qua các giao dịch một lần nhưng lại rất tốn thời gian đối với tập có độ dài lớn. Để khắc phục tình trạng này, ta có thể sử dụng kỹ thuật phân vùng và định dạng dọc bằng phương pháp PartEclat, được trình bày bởi Shashi Raj và Dharavath Ramesh trong bài báo cáo khoa học “PartEclat: an improved Eclat-based frequent itemset mining algorithm on spark clusters using partition technique” (2022)[4].

Tiếp theo là tránh bỏ sót các tập phổ biến, nếu độ hỗ trợ quá cao ECLAT có thể bỏ sót các tập phổ biến. Để khắc phục điều này, chúng ta có thể áp dụng phương pháp thống kê để ước lượng độ hỗ trợ một cách tối ưu nhất.

Tài liệu

[1] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499. Santiago, Chile, 1994.

[2] Jiawei Han. Mining frequent pattern without candidate generation. In *SIGMOD, 2000*, 2000.

[3] David L Olson, Dursun Delen, David L Olson, and Dursun Delen. Data mining process. *Advanced Data Mining Techniques*, pages 9–35, 2008.

[4] Shashi Raj and Dharavath Ramesh. Parteclat: an improved eclat-based frequent itemset mining algorithm on spark clusters using partition technique. *Cluster Computing*, 25(6):4463–4480, 2022.

[5] M Sinthuja, P Aruna, and N Puviarasan. Experimental evaluation of apriori and equivalence class clustering and bottom up lattice traversal (eclat) algorithms. *Pakistan journal of biotechnology*, 13(special issue II):77–82, 2016.

[6] Mohammed Javeed Zaki. Scalable algorithms for association mining. *IEEE transactions on knowledge and data engineering*, 12(3):372–390, 2000.