TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

CHƯƠNG TRÌNH CHẤT LƯỢNG CAO

LÊ THÀNH NAM – LÊ HỒNG QUANG

KHAI THÁC LUẬT CÓ THỨ TỰ BÁN PHẦN TRONG CƠ SỞ DỮ LIỆU CHUỖI

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT

TP.HCM, NĂM 2022

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

CHƯƠNG TRÌNH CHẤT LƯỢNG CAO

LÊ THÀNH NAM – 18127158

LÊ HỒNG QUANG – 18127190

KHAI THÁC LUẬT CÓ THỨ TỰ BÁN PHẦN TRONG CƠ SỞ DỮ LIỆU CHUỖI

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT

GIÁO VIÊN HƯỚNG DẪN

GS.TS LÊ HOÀI BẮC

TP.HCM, NĂM 2022

Lời cảm ơn

Đầu tiên chúng tôi muốn dành lời cảm ơn sâu sắc đến giảng viên hướng dẫn của chúng tôi – Thầy Lê Hoài Bắc vì đã tận tình hướng dẫn và trợ giúp chúng tôi xuyên suốt quá trình thực hiện khóa luận. Khóa luận này gần như không thể hoàn thành nếu không có sự tham gia của thầy.

Tiếp theo, chúng tôi xin chân thành cảm ơn các giáo viên của Khoa Công nghệ thông tin, trường Đại học Khoa học Tự nhiên, Đại học Quốc gia TP.Hồ Chí Minh vì đã cho chúng tôi một môi trường làm việc và học tập chuyên nghiệp trong suốt bốn năm học.

Chúng tôi cũng xin cảm tạ sự hỗ trợ nhiệt tình của phía gia đình và bạn bè trong những lúc khó khăn cũng như trong suốt quá trình thực hiện khóa luận.

Trong khóa luận này, mặc dù đã cố gắng hoàn thiện những gì trong khả năng nhưng vẫn không tránh được khả năng mắc sai sót. Rất mong quý thầy cô đưa ra những nhận xét và đánh giá để chúng tôi tiến hành chỉnh sửa trong thời gian sớm nhất.

Một lần nữa, xin chân thành cảm ơn và mong nhận được sự giúp đỡ của quý thầy cô để khóa luận tốt nghiệp được hoàn thành trọn vẹn.

Thành phố Hồ Chí Minh, tháng 07 năm 2022

Nhóm sinh viên thực hiện

Lê Hồng Quang – Lê Thành Nam

Mục lục

Danh sách các hình

Danh sách các bảng

Chữ viết tắt

Partially-Ordered Episode Rule Miner – POERM

Frequent Episode Mining – FEM

Partially-Ordered Episode Rule – POER

Tóm tắt

Khai thác luật có thứ tự là một công đoạn quan trọng trong lĩnh vực khai thác dữ liệu phục vụ cho mục đích phân tích chuỗi các sự kiện hoặc ký tự. Công đoạn trên bao gồm xác định các chuỗi con xuất hiện thường xuyên trong một chuỗi lớn và kết hợp chúng lại với nhau tạo nên các tập luật có mối liên kết chặt chẽ giữa các sự kiện. Từ các luật được tạo nên, dữ liệu sẽ được thấu hiểu và khám phá một cách dễ dàng.

Vấn đề cốt lõi của khai thác luật có thứ tự nằm ở sự rập khuôn – sự lặp đi lặp lại của các luật theo các thứ tự khác nhau, dù về mặt ý nghĩa các thứ tự khác nhau này cùng biểu diễn cho một luật. Chính vì vậy, sẽ có các luật được xem là khác nhau nhưng trên thực tế lại miêu tả cùng một trường hợp.

Để có thể tìm ra các luật nhỏ mang tính tổng quát hơn và có khả năng thay thế phần lớn luật, khóa luận này nghiên cứu một cách tiếp cận mới để khai thác các luật, thuật toán Partially-Ordered Episode Rules Miner.

Kết quả đánh giá thực nghiệm trên một vài bộ dữ liệu chuẩn cho thấy sử dụng phương pháp này sẽ giảm nhiều gánh nặng về bộ nhớ cũng như tốc độ thực thi. Việc này sẽ giúp ích rất nhiều khi áp dụng việc khai thác luật cho các bộ dữ liệu có kích thước lớn. Kết quả thu được của phương pháp là các bộ luật hợp lệ tổng quát có thể biểu diễn cho hầu hết các thứ tự xuất hiện của bộ luật đó.

Chương 1: Giới thiệu đề tài

1.1 Tổng quan bài toán khai thác luật:

Hiện nay, sự bùng nổ thông tin cùng với sự phát triển của công nghệ lưu trữ và mạng Internet đã làm cho lượng dữ liệu phục vụ cho nhu cầu hằng ngày của các tổ chức trở nên phức tạp, đa dạng và phong phú hơn. Do đó, nhu cầu khám phá thông tin, tri thức cần thiết để định hướng chiến lược cho tổ chức đang là một thách thức lớn. Chính vì thế, lĩnh vực khai thác dữ liệu đã ra đời để giải mã các thông tin và tri thức tiềm ẩn sau dữ liệu.

Khai thác dữ liệu, hay còn gọi là khám phá tri thức, là quá trình khám phá những mẫu hay tri thức có ích từ nguồn dữ liệu, ví dụ như CSDL, văn bản, ảnh, dữ liệu Web, v.v…

Các mẫu khai thác được phải mang lại giá trị và lợi ích cho tổ chức và đặc biệt dễ hiểu.

Quá trình khám phá tri thức trong cơ sở dữ liệu được chia thành ba giải đoạn chính: Tiền xử lý dữ liệu, khai thác dữ liệu, và hậu xử lý. Quá trình này lặp đi lặp lại đến khi đạt được kết quả khả quan cuối cùng.

Hiện tại, khai thác dữ liệu thường tập trung vào các hướng chính và được đề cập nhiều nhất là phân lớp, phân cụm dữ liệu, khai thác luật kết hợp và khai thác luật tuần tự. Các mô hình liên quan được đề xuất gồm khai thác văn bản, khai thác dữ liệu Web, khai thác dữ liệu không gian và thời gian, v.v…

Một trong những mô hình mang tầm ảnh hưởng lớn và được tập trung nghiên cứu đó là mô hình cơ sở dữ liệu chuỗi. Trong đó, khai thác luật có thứ tự

là một trong những chủ đề nghiên cứu mang tính thiết thực và quan trọng bậc nhất trong lĩnh vực khai thác dữ liệu. Mục đích của việc này là tìm ra các luật biểu diễn các quy luật tiềm ẩn trong cơ sở dữ liệu chuỗi. Mỗi luật thể hiện mỗi quan hệ giữa các mẫu dữ liệu theo thứ tự thời gian.

Quá trình khai thác luật có thứ tự trong cơ sở dữ liệu chuỗi thường được chia thành hai giai đoạn chính: Khai thác mẫu và tiếp theo là giai đoạn sinh luật có thứ tự.

Do vậy, số lượng luật tuần tự khai thác được là nhân tố quyết định đến độ hiệu quả của việc sinh luật.

Hạn chế chính của các thuật toán khai thác mẫu tuần tự tổng quát là cố gắng tìm tất cả những luật tuần tự có thể có. Điều này làm gia tăng nhanh chóng mẫu khai thác được, dẫn đến sự dư thừa, trùng lặp của các mẫu. Đồng thời, một số thuật toán phải lưu giữ luật khai thác được ở bước trước nhằm mở rộng và kết hợp tạo thành các luật mới dẫn đến sự tiêu tốn bộ nhớ và thời gian lưu trữ lớn.

1.2 Động lực nghiên cứu:

Trong những năm gần đây, đã có rất nhiều thuật toán được thiết kế nhằm mục đích nhận dạng mẫu trong chuỗi dữ liệu rời rạc (một chuỗi các sự kiện hoặc ký hiệu) bởi tính phổ biến của loại dữ liệu này là cực kỳ cao. Ví dụ, một văn bản có thể được biểu diễn dưới dạng nhiều chuỗi các ký tự, một danh sách lịch sử mua hàng của một khách hàng có thể biểu diễn thành một chuỗi các giao dịch và việc xác định vị trí bằng vệ tinh có thể được biểu diễn bằng một chuỗi các địa điểm.

Trong khi một số phương pháp nhận dạng mẫu tìm thấy các mẫu tương đồng của các chuỗi dữ liệu [6, 7, 15, 16], có một số phương pháp khác lại tìm ra mẫu tương đồng chỉ trong một chuỗi dữ liệu có độ dài lớn. Một trong những công đoạn phổ biến của các phương pháp khác ở trên chính là khai thác luật phổ biến (Frequent Episode Mining - FEM) [9, 11, 12, 14]. Nó bao gồm việc tìm tất cả các luật phổ biến trong một chuỗi các sự kiện mà trong đó các chuỗi con có tần suất xuất hiện lớn hơn ngưỡng minsup được định nghĩa bởi người dùng.

Có hai kiểu chuỗi dữ liệu trong FEM: chuỗi đơn giản – chuỗi chỉ có một sự kiện xảy ra tại một mốc thời gian, tất cả các sự kiện đều có các mốc thời gian và hoàn toàn theo thứ tự, chuỗi phức tạp – chuỗi cho phép có nhiều sự kiện xảy ra trên một mốc thời gian.

Đã có nhiều thuật toán được ra đời phục vụ cho việc khám phá luật phổ biến như MINEPI hay WINEPI [12], EMMA và MINEPI+ [9] và TKE [8]. Trong khi một số thuật toán tìm ra các tập nối tiếp (tức là danh sách các sự kiện theo thứ tự), các thuật toán còn lại tìm các tập song song (tập các sự kiện đồng thời) hoặc tập kết hợp (sự kết hợp của các tập nối tiếp/song song). Mặc dù việc tìm các tập phổ biến là hữu ích thì các tập chỉ được khám phá dựa trên cơ sở hỗ trợ của chúng (tần suất xuất hiện) [1]. Do đó, một vài sự kiện chỉ có thể xuất hiện cùng nhau trong một tập một cách tình cờ. Hơn nữa, những tập phổ biến không cung cấp thông tin về khả năng một số sự kiện sẽ xảy ra sau một số sự kiện khác.

Để giải quyết những vấn đề này, một bước xử lý hậu kỳ có thể được áp dụng sau FEM, đó là kết hợp các cặp tập thường xuyên để tạo tập luật. Tập luật là một luật có dạng $E_1 \rightarrow E_2$, chỉ ra rằng nếu một tập E_1 xuất hiện, nó sẽ được theo sau bởi một tập E_2 với một độ tin cậy hoặc xác suất được cho trước [4, 12].

Khai thác tập luật là hữu ích bởi vì việc này có thể thể hiện mối quan hệ thời gian mạnh mẽ giữa các sự kiện trong dữ liệu từ nhiều miền [2-4, 12]. Ví dụ, một luật $R_1 : \langle\{a\},\{b\},\{c\}\rangle \rightarrow \langle\{d\}\rangle$ có thể được tìm thấy trong dữ liệu xem phim của khán giả, qua đó cho biết rằng nếu một người xem các bộ phim a, b và c theo thứ tự thì người đó sau đấy sẽ xem bộ phim d. Dựa trên những luật như vậy, các quyết định tiếp thị hoặc các khuyến nghị đến khách hàng có thể được đưa ra một cách hiệu quả.

Tuy nhiên, một nhược điểm lớn của những thuật toán khai thác luật truyền thống chính là các sự kiện trong từng luật phải theo thứ thự một cách chặt chẽ. Dẫn đến, các luật tương tự nhau lại được xử lý khác nhau. Ví dụ, luật E_1 được cho là khác với những luật:

$R_2 : \langle\{b\},\{a\},\{c\}\rangle \rightarrow \langle\{d\}\rangle$, $R_3 : \langle\{b\},\{c\},\{a\}\rangle \rightarrow \langle\{d\}\rangle$, $R_4 : \langle\{c\},\{a\},\{b\}\rangle \rightarrow \langle\{d\}\rangle$, $R_5 : \langle\{c\},\{b\},\{a\}\rangle \rightarrow \langle\{d\}\rangle$ và $R_6 : \langle\{a\},\{c\},\{b\}\rangle \rightarrow \langle\{d\}\rangle$. .

Tất cả những luật trên đều chứa cùng các sự kiện giống nhau. Đây là một vấn đề vì tất cả những luật trên là tương tự nhau và trong thực tế có thể đại diện cho cùng một tình huống, chẳng hạn một người đã xem ba bộ phim (ví dụ: Frozen, Sleeping Beauty và Lion King) thì sau đó sẽ xem bộ phim khác (ví dụ: Harry Potter). Bởi vì những luật này được xem là khác biệt, tần suất xuất hiện và độ tin cậy của chúng được tính toán tách biệt và có thể rất khác nhau. Hơn nữa, việc phân tích số lượng lớn các luật đại diện cho cùng một tình huống với những thay đổi nhỏ về thứ tự sẽ không thuận tiện cho người dùng. Do đó, sẽ có mong muốn trích xuất các loại luật mang tính tổng quát và linh hoạt hơn, cụ thể sự thay đổi về thứ tự giữa các sự kiện là chấp nhận được.

Khóa luận này giải quyết vấn đề trên bằng cách nghiên cứu một loại luật mới được gọi là các tập luật có thứ tự bán phần – Partially-Ordered Episode

Rules, với những sự kiện trong phần tiền tố của một luật và trong phần hậu tố của một luật là không có thứ tự. Một POER có dạng $E_1 \rightarrow E_2$ với E_1 và E_2 là tập các sự kiện. Một luật được hiểu như sau: nếu tất cả sự kiện trong E_1 xuất hiện theo thứ tự bất kỳ thì chúng sẽ được theo sau bởi tất cả sự kiện trong E_2 theo thứ tự bất kỳ.

Ví dụ, một POER $R_7 : \{a, b, c\} \rightarrow \{d\}$ chỉ ra rằng nếu một người xem các bộ phim a,b và c theo thứ tự bất kỳ thì người đó sẽ xem bộ phim d. Lợi ích của việc tìm ra các POER là một luật đơn lẻ có thể thay thế được nhiều tập luật. Ví dụ, R_7 có thể thay thế $R_1, R_2, \ldots, R_6$.

Tuy nhiên việc khám phá các POER là thử thách bởi vì chúng không bắt nguồn từ các tập. Do đó, một thuật toán mới phải được thiết kế để tìm ra các POER trong một dữ liệu chuỗi một cách hiệu quả.

Trong khóa luận này, vấn đề của việc khám phá các POER sẽ được định nghĩa và các thuộc tính của nó sẽ được nghiên cứu. Sau đấy, một thuật toán hiệu quả tên là Partially-Ordered Episode Rule Miner – Khai thác luật có thứ tự bán phần, sẽ được trình bày. Cuối cùng, một đánh giá thực nghiệm được thực hiện trên một số bộ dữ liệu chuẩn để đánh giá POERM. Các kết quả cho thấy rằng thuật toán này có hiệu suất tuyệt vời.

1.3 Mục tiêu nghiên cứu:

Mục tiêu nghiên cứu mà khóa luận muốn hướng đến là áp dụng thuật toán POERM được đề xuất bởi Philippe Fournier-Viger và cộng sự để khai thác hiệu quả các luật có thứ tự bán phần, thông qua các luật đó giúp người dùng hiểu rõ thêm về bộ dữ liệu và từ đó làm tiền đề để thực hiện các bước khai thác dữ liệu tiếp theo một cách hiệu quả.

Chương 2: Cơ sở lý thuyết

2.1 Vấn đề bài toán:

Kiểu dữ liệu sẽ xem xét và xử lý trong khai thác luật phổ biến là một kiểu chuỗi chứa các các dữ liệu đi cùng với các mốc thời gian cụ thể [9, 12].

Gọi chuỗi các sự kiện là tập hợp $E = \{e_1, e_2, e_3, \dots e_m\}$ biểu diễn cho các vật phẩm cũng như kí tự. Thêm vào đó, gọi các mốc thời gian là tập hợp $T = \{t_1, t_2, t_3, \dots, t_n\}$ với bất kỳ số nguyên $1 \leq i < j \leq n$ và luôn có mối quan hệ $t_i < t_j$.

Khoảng thời gian $[t_i, t_j]$ sẽ có thời lượng là $t_j - t_i$ đơn vị thời gian. Hơn nữa, hai khoảng thời gian $[t_{i1}, t_{j1}]$ và $[t_{i2}, t_{j2}]$ được gọi là không trùng lặp nếu $t_{j1} < t_{i2}$ hay $t_{j2} < t_{i1}$. Nói cách khác, hai khoảng thời gian này sẽ không chồng chéo lên nhau nếu chúng không trùng lặp.

Một tập hợp X ⊆ E được gọi là chuỗi sự kiện. Theo đó, X được gọi là chuỗi *k*-sự kiện nếu X có chứa *k* sự kiện.

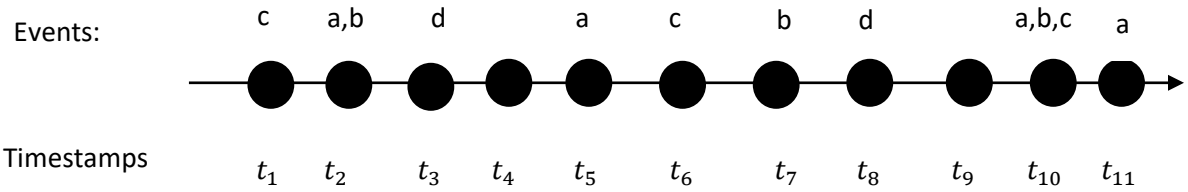
Một chuỗi sự kiện phức tạp là một danh sách các chuỗi sự kiện theo thứ tự đi cùng với các mốc thời gian, được biểu diễn dưới dạng $S = \langle (SE_{t_1}, t_1), (SE_{t_2}, t_2), \dots, (SE_{t_n}, t_n) \rangle$ với $SE_{t_i} \subseteq E, 1 \leq i \leq n$.

Một chuỗi sự kiện liên tục trong một chuỗi sự kiện phức tạp là một chuỗi sự kiện mà các sự kiện bên trong nó xảy ra cùng lúc hay cùng mốc thời gian.

Nếu một chuỗi sự kiện phức tạp chứa không quá một sự kiện trong mỗi mốc thời gian, nó là chuỗi sự kiện đơn giản.

Dữ liệu các loại khác nhau đều có thể biểu diễn dưới dạng chuỗi như chuỗi báo động, chuỗi dữ liệu về vị trí di chuyển, ... [8].

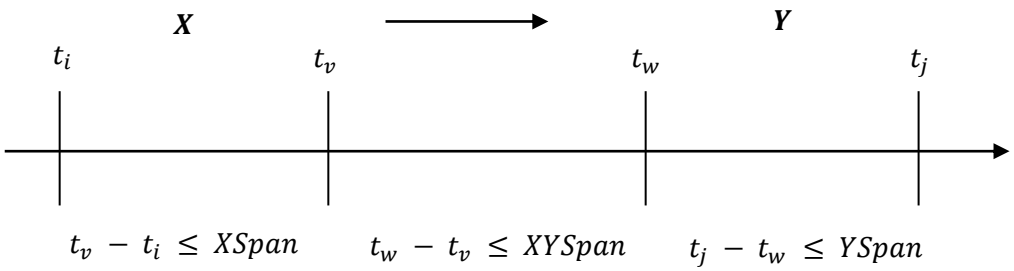
Ví dụ, một chuỗi phức tạp được biểu diễn trong Hình 1. Chuỗi này bao gồm mười một mốc thời gian $T = \{t_1, t_2, t_3, \dots, t_{11}\}$ và tập sự kiện $E = \{a,\ b,\ c,\ d\}$. Chuỗi trên chỉ ra rằng sự kiện c xuất hiện tại mốc thời gian t_1, theo sau đó là sự kiện $\{a, b\}$ là sự kiện liên tục tại mốc thời gian t_2, theo tiếp là sự kiện đơn giản *d* tại mốc thời gian t_3, sự kiện *a* tại mốc thời gian t_5 và sự kiện *c* tại mốc thời gian t_6, v.v…



Hình 1: Ví dụ minh họa một chuỗi sự kiện phức tạp

Khóa luận này nghiên cứu một loại luật mới được gọi là tập luật có thứ tự bán phần. Một POER có dạng $X \rightarrow Y$ với X ⊂ E và Y ⊂ E và X, Y ∉ ∅. Ý nghĩa của luật như vậy là nếu tất cả các sự kiện từ X xuất hiện theo bất kỳ thứ tự nào trong chuỗi, chúng sẽ được theo sau bởi tất cả sự kiện từ Y. Để tránh việc tìm thấy các luật chứa các sự kiện quá xa rời, ba ràng buộc được chỉ định: (1) các sự kiện từ X phải xuất hiện trong khoảng thời gian tối đa $XSpan \in \mathbb{Z}^+$, (2) các sự kiện từ Y phải xuất hiện trong khoảng thời gian tối đa $YSpan \in \mathbb{Z}^+$ và (3) thời gian giữa X và Y không được lớn hơn hằng số $XYSpan \in \mathbb{Z}^+$. Ba ràng buộc

XSpan, YSpan và XYSpan phải được xác định bởi người dùng và được minh họa trong hình 2 cho một luật $X \rightarrow Y$.



Hình 2: Ví dụ minh họa cho 3 tham số thời gian do người dùng định nghĩa

Hơn nữa, để lựa chọn các luật thú vị, hai độ đo được sử dụng gọi là độ hỗ trợ và độ tin cậy, được lấy cảm hứng từ các công việc trước đây trong khai thác luật. Chúng được định nghĩa dựa trên khái niệm về sự xuất hiện.

Nguyên nhân sử dụng độ hỗ trợ và độ tin cậy: độ hỗ trợ là một thước đo quan trọng bởi vì một luật có độ hỗ trợ rất thấp có thể xảy ra một cách tình cờ. Ngoài ra, từ góc độ kinh doanh, một luật với độ hỗ trợ thấp có thể không thú vị vì nó có thể không mang lại lợi nhuận khi quảng cáo các mặt hàng mà các khách hàng hiếm khi mua cùng nhau. Độ hỗ trợ được khai thác để phát hiện hiệu quả các quy tắc kết hợp. Mặt khác, độ tin cậy đo lường sự đáng tin của suy luận được đưa ra bởi một luật. Đối với luật $X \rightarrow Y$ cho trước, độ tin cậy càng cao thì càng

có nhiều khả năng Y có mặt trong các giao dịch chứa X. Độ tin cậy cũng cung cấp ước tính về xác suất có điều kiện của Y với X được cho trước.

Một thời điểm diễn ra của một tập sự kiện F ⊂ E trong một chuỗi sự kiện phức tạp S là một khoảng thời gian $[t_i,\ t_j]$ nơi mà tất cả sự kiện từ F xuất hiện, đó là F ⊆ $SE_i \cup SE_{i+1} \cdots \cup SE_j$.

Một thời điểm diễn ra của một luật $X \to Y$ trong một chuỗi sự kiện phức tạp S là một khoảng thời gian $[t_i,\ t_j]$ sao cho tồn tại một số mốc thời gian t_v, t_w với $t_i \le t_v < t_w \le t_j$, X có sự xuất hiện trong $[t_i, t_v]$, Y có sự kiện trong $[t_w, t_j]$, $t_v - t_i < XSpan,\ t_w - t_v < XYSpan$ và $t_j - t_w < YSpan$.

Việc phân tích sự xuất hiện của các tập sự kiện hoặc các luật, qua đó có thể tiết lộ các mối quan hệ thú vị giữa các sự kiện. Tuy nhiên, có một vấn đề là một vài sự xuất hiện có thể chồng lên nhau, do đó một sự kiện có thể được tính là một phần của nhiều lần xuất hiện. Để giải quyết vấn đề này, khóa luận này đề xuất chỉ xem xét một tập con của tất cả các sự xuất hiện được định nghĩa như sau. Một thời điểm diễn ra $[t_{i1}, t_{j1}]$ được cho là dư thừa trong một tập các sự xuất hiện nếu tồn tại một khoảng thời điểm diễn ra trùng lặp $[t_{i2}, t_{j2}]$, sao cho $t_{i1} \le t_{i2} \le t_{j1}$ hoặc $t_{i2} \le t_{i1} \le t_{j2}$. Cho $occ(F)$ biểu thị tập hợp các lần xuất hiện không dư thừa của một tập sự kiện F trong một chuỗi S. Hơn nữa, cho $occ(X \to Y)$ biểu thị tập các lần xuất hiện không trùng lặp của một luật $X \to Y$ trong một chuỗi S.

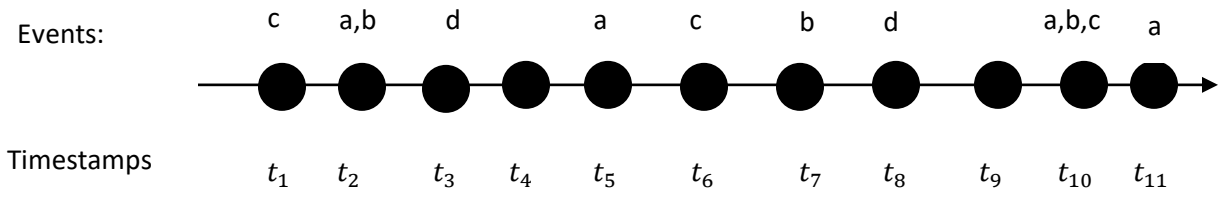
Độ hỗ trợ của một luật $X \to Y$ được định nghĩa là $sup(X \to Y) = |occ(X \to Y)|$. Độ hỗ trợ của một tập sự kiện F được định nghĩa là $sup(F) = |occ(F)|$. Độ tin cậy của một luật $X \to Y$ được định nghĩa là $conf(X \to Y) =$

$|occ(X \rightarrow Y)|/|occ(X)|$. Nó đại diện cho xác suất có điều kiện mà các sự kiện từ X được theo sau bởi các sự kiện từ Y.

Định nghĩa 1:

Cho X, Y là các tập sự kiện, S là một chuỗi sự kiện phức tạp và năm tham số được định nghĩa bởi người dùng: XSpan, YSpan, XYSpan, minsup và minconf. Vấn đề của việc khai thác các POER là tìm tất cả các POER hợp lệ. Một POER r được gọi là phổ biến nếu $sup(r) \geq minsup$, và một POER r được gọi là hợp lệ nếu POER đó thường xuyên và $conf(r) \geq minconf$.

Ví dụ:



Hình 3: Ví dụ minh họa một chuỗi sự kiện phức tạp

Xét *hình 3*, minsup = 3, minconf = 0.6, XSpan = 3, XYSpan = 1 và YSpan = 1. Sự xuất hiện của $\{a, b, c\}$ là $occ(\{a, b, c\} = \{[t_1, t_2], [t_5, t_7], [t_{10}, t_{10}]\}$. Sự xuất hiện của luật $R : \{a, b, c\} \rightarrow \{d\}$ là $occ(R) = \{[t_1, t_3], [t_5, t_8]\}$. Do đó, sup(R) = 3, conf (R) = 2/3 và R là một luật hợp lệ.

2.2 Thuật toán POERM:

Khó khăn của việc khai thác POER nằm ở số lượng luật được sinh ra tương ứng với số lượng sự kiện. Một tập dữ liệu gồm m sự kiện riêng biệt có thể

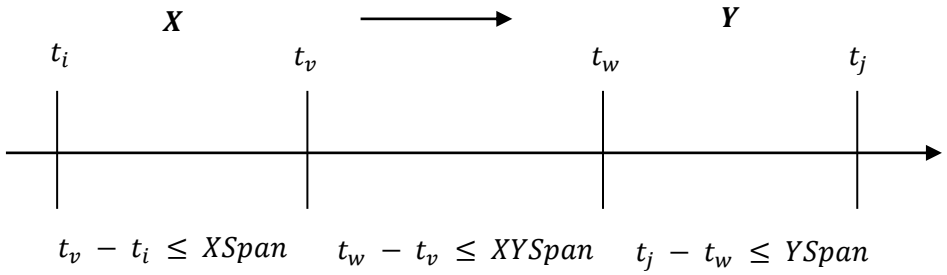
sản sinh ra số lượng luật lên đến con số $(2^m - 1) \times (2^m - 1)$. Hơn nữa, mỗi luật có thể có một số lượng lớn các thời điểm diễn ra trong một chuỗi. Nói cách khác, gánh nặng về bộ nhớ lưu trữ là quá lớn khi sử dụng bộ luật POER.

Đầu tiên, thuật toán POERM sẽ tìm tất cả các tập sự kiện có khả năng là tiền tố của một luật hợp lệ cũng như các thời điểm diễn ra của chúng trong chuỗi đầu vào và lưu lại chúng. Tiếp đến, thuật toán tìm kiếm các hậu tố có thể kết hợp với các tiền tố kể trên để xây dựng các POER. Các POER hợp lệ này sẽ được lưu giữ lại và biểu diễn cho người sử dụng thấy được. Để tránh việc suy xét tất cả các luật có thể, thuật toán POERM tối ưu hóa bằng việc cắt giảm không gian tìm kiếm qua Quy tắc 1.

Quy tắc 1:

Một tập sự kiện X không thể là tiền tố của một luật hợp lệ nếu $sup(X) < minsup$. Một tập sự kiện Y không thể là hậu tố của một luật hợp lệ nếu $sup(Y) < minsup \times minconf$.

Thuật toán POERM được biểu diễn qua hình 5, nhận dữ liệu đầu vào là một chuỗi sự kiện phức tạp S cùng với các tham số được định nghĩa bởi người dùng là XSpan, YSpan, XYSpan, minsup và minconf.



Hình 4: Ví dụ minh họa cho 3 tham số thời gian do người dùng định nghĩa

Ban đầu, thuật toán đọc chuỗi sự kiện S và tạo ra XFres – một bản copy của chuỗi S nhưng chỉ chứa các sự kiện có support không bé hơn minsup. Các sự kiện khác sẽ được loại bỏ bởi vì chúng không thể xuất hiện trong các luật hợp lệ dựa theo Quy tắc 1. Sau đó, thuật toán tìm kiếm các tiền tố của luật bằng cách gọi quy trình MiningXEventSet với các tham số đầu vào XFres, XSpan và minsup. Quy trình này trả về giá trị là một danh sách xSet chứa các chuỗi sự kiện có khả năng làm tiền tố của các POER hợp lệ, trong đó từng tập sự kiện có tối thiểu minsup số lần diễn ra không trùng lặp của một khoảng thời gian không lớn hơn XSpan.

Algorithm 1: *POERM*

Input: an event sequence S, the *XSpan*, *YSpan*, *XYSpan*, *minsup* and *minconf* parameters;

Output: the set POERs of valid partially-ordered episode rules

1 *XFres* = *loadFrequentSequence*(*S*, *minsup*);

2 *xSet* = *MiningXEventSet*(*XFres*, *XSpan*, *minsup*);

3 *YFres* = *loadFrequentSequence*(*XFres*, *minsup* × *minconf*);

4 POERs ← ∅;

5 **foreach** event set *x* in *xSet* do

6 **for** *i* = 1 to (*XY Span* +*Y Span*) **do**

7 *xOccurrenceList*←{*t*|*t* = *occur.end* + *i*, *occur* ∈ *x.OccurrenceList*}

8 **end**

9 Scan each timestamp of *YFres* in *xOccurrenceList* to obtain a map *conseMap* that records each event e and its occurrence list;

Scan *conseMap* and put the pair (*x*⟶*e*, *OccurrenceList*) in a queue *candidateRuleQueue* (note: infrequent rules are kept because event e may be extended to obtain some frequent rules);

Add each rule $x \rightarrow e$ such that $|occ(x \rightarrow e)| \geq minconf \times |occur(x)|$ into the set POERs;

while $candidateRuleQueue \neq \emptyset$ **do**

    Pop a rule X → Y from *candidateRuleQueue*;

    For each occurrence occur of X → Y , let;

    start ←*max*(*occur.X.end* + 1, *occur.Y.end* − *YSpan* + 1);

    end ←*min*(*occur.X.end* + *XYSpan* + *YSpan*, *occur.X.start* + *YSpan*);

    Scan each timestamp in *[start,end),* add each candidate rule in *candidateRuleQueue*, and add each valid POER in POERs;

end

end

return POERs;

Hình 5: Thuật toán POERM

Quy trình MiningXEventSet được biểu diễn qua hình 6. Trước hết, quy trình quét chuỗi XFres để tìm danh sách các thời điểm diễn ra của từng sự kiện. Dữ kiện này được lưu lại trong một map-fresMap bao gồm các cặp giá trị (key, value) với key biểu thị cho sự kiện và value là danh sách thời điểm diễn ra của sự kiện đó. Sau đó, quy trình sẽ quét tất cả các cặp giá trị trong fresMap và đưa cặp giá trị (key, value) mà có tối thiểu minsup số lần diễn ra không trùng lặp hay

có thể hiểu |value| (số lượng phần tử trong value) – số lượng thời điểm diễn ra không trùng lặp lớn hơn hoặc bằng minsup – vào xSet. Tại thời điểm này, xSet sẽ chứa tất cả các tập 1-sự kiện có khả năng làm tiền tố của luật hợp lệ theo Quy tắc 1. Sau đó, quy trình quét toàn bộ chuỗi XFres thêm một lần nữa để mở rộng các tập 1-sự kiện thành các tập 2-sự kiện và tiếp tục mở rộng các tập 2-sự kiện thành các tập 3-sự kiện và chỉ kết thúc cho tới khi không tăng số lượng sự kiện trong chuỗi sự kiện lên được nữa. Trong quá trình mang tính lặp lại trên, mỗi tập sự kiện được sản sinh ra mà có số lượng các thời điểm diễn ra không trùng lặp lớn hơn minsup sẽ được thêm vào danh sách xSet và được xem xét để mở rộng thành các tập chứa số lượng sự kiện lớn hơn. Cuối cùng, quy trình MiningXEventSet sẽ trả về xSet, danh sách chứa tất cả các tiền tố hợp lệ của các POER hợp lệ.

Một thử thách trong việc cài đặt quy trình MiningXEventSet hiệu quả là các chuỗi sự kiện – theo định nghĩa – không có quy luật về mặt thứ tự. Chính vì vậy, các thứ tự khác nhau của cùng một tập sự kiện sẽ đại diễn cho cùng một tập sự kiện. Ví dụ, ba chuỗi sự kiện {a, b, c}, {b, a, c}, {a, c, b} là cùng một chuỗi sự kiện. Nhằm loại bỏ việc sản sinh ra các tập sự kiện trùng lặp nhiều lần, các sự kiện trong một tập sự kiện sẽ được sắp xếp theo thứ tự của bảng chữ cái (ví dụ {a,b,c}) hoặc thứ tự số học. Cụ thể ở ví dụ trên, cả ba chuỗi sự kiện sẽ chỉ được biểu diễn bằng chuỗi sự kiện {a, b, c} và quy trình sẽ chỉ mở rộng một chuỗi sự kiện F bằng cách thêm một sự kiện e vào F với điều kiện sự kiện e phải lớn hơn sự kiện cuối cùng trong F (sự kiện mang giá trị lớn nhất về mặt chữ cái hoặc số học).

Algorithm 2: *MiningXEventSet*

Input: *XFres*: the sequence with only events having support ≥ *minsup*; XSpan: maximum window size; *minsup* threshold;

Output: a list of event sets that may be antecedents of valid POERs

1 Scan the sequence *XFres* to record the occurrence list of each event in a map *fresMap* (*key* = event set, *value* = occurrence list);

2 *xSet* ← all the pairs of *fresMap* such that |value| ≥ *minsup*;

3 *start* ← 0;

4 **while** *start* < |*xSet*| **do**

5 F ← *xSet*[*start*].*getKey*;

6 *OccurrenceList* ← *xSet*[*start*].*getValue*;

7 Clear *fresMap*;

8 *start* = *start* +1;

9 **foreach** occurrence pos in *OccurrenceList* **do**

10 *pStart* ← pos.end− *XSpan* +1; *pEnd* ←− pos.*start*+ *XSpan*;

11 Search the time intervals [*pStart*, *pos*.*start*), [*pos*.*end* + 1, *pEnd*), [*pos*.*start*, *pos*.*end*] to add each event set $F \cup \{e\}$ such that $e >$ *F.lastItem* and its occurrences of the forms [*i, pos.end], [pos.start,*

i] or *[pos.start, pos.end*], in the map *fresMap*;

end

Add each pair of *fresMap* such that |value| ≥ *minsup* into *xSet;*

end

return *xSet*

Hình 6: Quy trình MiningXEventSet

Một điều kiện cốt lõi được xem như là thử thách thứ hai trong việc cài đặt quy trình trên là tìm ra phương pháp để mở rộng một tập *l*-sự kiện F thành các tập (*l*+1)-sự kiện và tính toán các thời điểm diễn ra của nó bằng cách sử dụng chuỗi XFres. Để tránh việc quét hết toàn bộ chuỗi XFres trên, công việc tính toán sẽ được thực hiện bằng cách tìm kiếm xung quanh mọi thời điểm diễn ra pos (occurrence pos) của F trong danh sách OccurrenceList của F. Cho pos.start, pos.end là hai mốc thời gian bắt đầu và kết thúc của thời diểm diễn ra pos. Thuật toán tìm kiếm các sự kiện diễn ra trong ba khoảng thời gian [pos.end – XSpan + 1, pos.start), [pos.end +1, pos.start + Xspan) và [pos.start, pos.end]. Với mỗi sự kiện e lớn hơn sự kiện cuối cùng trong F, các thời điểm diễn ra của chuỗi sự kiện $F \cup \{e\}$ sẽ có dạng [i, pos.end], [pos.start, i] hay [pos.start, pos.end] với i là mốc thời gian nằm trong ba khoảng không gian tìm kiếm đã được đề cập trước đó. Và các thời điểm diễn ra của chuỗi sự kiện $F \cup \{e\}$ trên sẽ được thêm vào fresMap. Tiếp đến, quy trình quét toàn bộ fresMap để đếm số lượng các thời điểm diễn ra của từng tập (*l*+1)-node sự kiện và tất cả các tập sự kiện có các thời điểm diễn ra không trùng lặp không bé hơn minsup được thêm vào xSet.

Việc đếm được tối đa số lần các thời điểm diễn ra không trùng lặp của một tập *l*-sự kiện F được thực hiện bằng cách sử dụng OccurenceList của F (không được nêu trong hình 6). Quy trình đầu tiên áp dụng thuật toán sắp xếp quick sort để sắp xếp lại OccurenceList theo thứ tự mốc thời gian kết thúc (pos.end) tăng dần. Sau đó, một tập CoverSet được tạo ra để lưu trữ số lượng tối đa các thời điểm diễn ra không trùng lặp của F. Thuật toán thực hiện vòng lặp đi hết các phần tử trong OccurrenceList của F để kiểm tra từng thời điểm diễn ra từ lần đầu tiên đến lần cuối cùng. Nếu thời điểm diễn ra đang được xét ở hiện tại không trùng lặp với thời điểm cuối cùng vừa được thêm vào CoverSet thì nó sẽ được thêm vào CoverSet. Nếu không, nó sẽ bị bỏ qua. Khi vòng lặp kết thúc, tập CoverSet sẽ chứa số lượng tối đa các thời điểm diễn ra không trùng lặp của F.

Sau khi áp dụng quy trình MiningXEventSet để tìm kiếm tiền tố, thuật toán POERM tìm kiếm các tập sự kiện có thể là hậu tố kết hợp được với các tiền tố đã được thu thập trước đó nhằm mục đích hình thành các POER hợp lệ. Ở bước này, việc đầu tiên thuật toán thực hiện là loại bỏ tất cả các sự kiện có số lượng thời điểm diễn ra nhỏ hơn minsup × minconf từ chuỗi XFres để tạo nên YFres dựa vào Quy tắc 1. Tiếp đến, một vòng lặp đi qua mỗi tiền tố x trong xSet để tìm kiếm hậu tố. Trong vòng lặp trên, các khoảng thời gian mà hậu tố có khả năng xuất hiện sẽ được xác định trước và lưu vào biến xOccurrenceList. Thêm vào đó, một cấu trúc dữ liệu dạng map conseMap sẽ được tạo ra để lưu mỗi sự kiện e cùng với danh sách các thời điểm diễn ra của nó cho những mốc thời gian này trong YFres. Sau đó, map sẽ được quét toàn bộ để tạo ra hàng chờ candidateRuleQueue chứa các luật có dạng $x \rightarrow e$ cùng với danh sách các thời điểm diễn ra của luật trên. Nếu như luật $x \rightarrow e$ có $|occ(x \rightarrow e)| \geq$ minconf ×

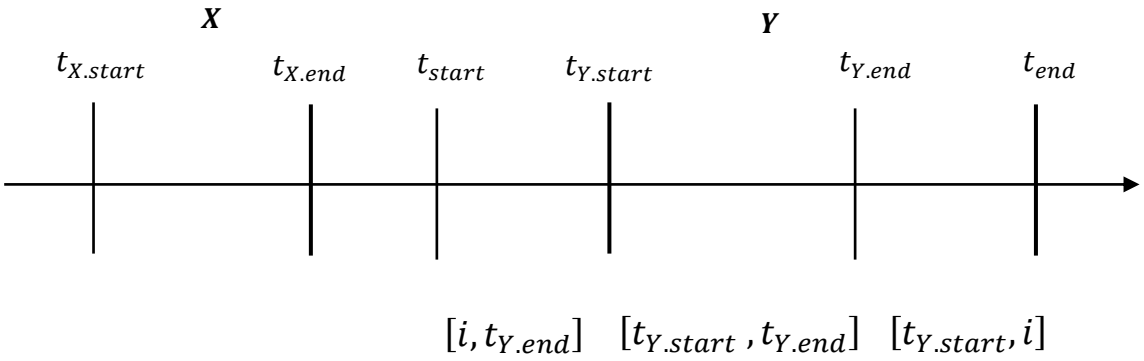
|occ(x)| thì sẽ được thêm vào tập chứa các POER hợp lệ. Lúc này, candidateRuleQueue sẽ chứa tất cả các luật ứng cử viên với hậu tố là *1*-sự kiện.

Ở giai đoạn này, thuật toán thực hiện thêm một vòng lặp để loại các luật $X \rightarrow Y$ khỏi hàng chờ để tiến hành mở rộng hậu tố bằng cách kết hợp với các sự kiện đơn. Giai đoạn này được tiến hành thông qua việc quét toàn bộ conseMap. Luật mở rộng thu được sẽ có dạng $X \rightarrow Y \cup \{e\}$ và danh sách các thời điểm diễn ra của luật trên được thêm vào candidateRuleQueue để tiếp tục phục vụ cho quá trình mở rộng tiếp theo. Hơn nữa, mỗi luật có |occ (x → e) | ≥ minconf × |occ(x)| sẽ được thêm vào tập chứa các POER hợp lệ. Vòng lặp sẽ tiếp tục cho đến khi candidateRuleQueue trở thành hàng chờ rỗng. Cuối cùng, thuật toán trả về tất cả các POER hợp lệ.

Có thể nhận thấy rằng một thời điểm diễn ra của luật $X \rightarrow Y$ có hậu tố là tập *l*-sự kiện có thể không được đếm vào tần suất xuất hiện của nó trong khi lại có thể được đếm vào tần suất của một luật có hậu tố (*l+1*)-sự kiện – luật được mở rộng dựa vào luật trước đó. Ví dụ, xem xét chuỗi trong hình 1, ta có XSpan = YSpan = 2 và XYSpan = 1. Thời điểm diễn ra của luật {a} → {b} trong khoàng thời gian [t_5, t_7] không được đếm vào tần suất xuất hiện của luật {a} → {b} bởi vì khoảng cách thời gian giữa a và b lớn hơn XYSpan. Nhưng luật {a} → {b} có thể được mở rộng hậu tố trở thành {a} → {b, c} và lúc này thời điểm diễn ra trong khoảng thời gian [t_5, t_7] được tính vào tần suất xuất hiện của luật. Để tìm ra các thời điểm diễn ra của luật dạng $X \rightarrow Y \cup \{e\}$ được mở rộng từ luật $X \rightarrow Y$, một cách tiếp cận sau được sử dụng. Với từng thời điểm diễn ra của luật $X \rightarrow Y$ một biến start được khởi tạo giá trị max(occur.X.end +1, occur.Y.end – Yspan +1), một biến end được khởi tạo giá trị (occur.X.end + XYSpan + YSpan, occur.Y.start +YSpan). Tiếp đến, ba khoảng thời gian được khoanh vùng là

[start, occur.Y.end), (occur.Y.end, end) và [occur.Y.start, occur.Y.end] để thêm luật $X \rightarrow Y \cup \{e\}$ sao cho e > Y.lastItem cùng các thời điểm diễn ra của nó sẽ xuất hiện dưới dạng [i, occur.Y.end], [occur.Y.start, i] hay [occur.Y.start, occur.Y.end] vào conseMap. Ba khoảng thời gian trên đều được lưu trữ trong hình 7 và cho phép thuật toán tìm ra chính xác các thời điểm diễn ra của các luật có hậu tố là (*l+1*)-node.

Trong hình 7 dưới, t_Y biểu diễn cho occur.Y và t_X biểu diễn cho occur.X:



Hình 7: Các mốc thời gian được tìm kiếm ứng với mỗi luật X → Y

Thuật toán POERM được đề xuất có thể tìm thấy tất cả bộ POER hợp lệ vì nó cắt giảm các sự kiện không thể là thành phần của luật hợp lệ theo Quy tắc 1. Chương dưới đây đưa ra các đánh giá tổng quan về POERM, nơi hiệu suất được so sánh với phiên bản đã cải tiến POERM-Optimize.

sChương 3: Kết quả nghiên cứu

3.1 Đánh giá thực nghiệm:

Thực hiện chạy thuật toán trên hai bộ dữ liệu chuẩn từ thư viện SPMF [5], có tên OnlineRetail và Fruithut.

OnlineRetail là một chuỗi gồm 541,909 giao dịch từ một cửa hàng bán lẻ trực tuyến có trụ sở tại Vương quốc Anh với 2,603 sự kiện riêng biệt cho biết việc mua các mặt hàng.

Fruithut là một chuỗi gồm 181,970 giao dịch của khách hàng từ một cửa hàng bán lẻ ở Mỹ, chủ yếu là bán trái cây với 1,265 loại sự kiện riêng biệt.

Thuật toán có năm tham số (minSupport, minConfidence, XSpan, YSpan, XYSpan), các tham số sẽ được tùy chỉnh sao cho phù hợp với từng bộ dữ liệu trên, lần lượt là:

OnlineRetail : (7000, 0.5, 5, 5, 5)

Fruithut: (5000, 0.5, 5, 5, 5)

Bảng 1: Kết quả thực thi trên hai bộ dữ liệu

Tên bộ dữ liệu	Thời gian (s)	Bộ nhớ (MB)
OnlineRetail	≈1848	≈2012
Fruithut	≈733	≈595

```
1084 1097 1126 2183 2375
1261 1394 2375
582 644 668 1082 1100
349 897 1142 1243 2316 2363
1098 1143 1816 2375 2402
121 219 363 1500 1943
964 1017 1126 2096 2183
1079 1189 2316 2356
766 1079 1720 1816 2356
209 298 593 1565
276 1709 1737
1236 1709 1737
508 752 881 1763
624 1182 1243 1456
114 209 358 593 1534 2209
320 358 1208 1534 2339
173 320 358 1534 2339
319 388 1131 2403
319 388 1324 2403
358 1097 1394 1809 2395
762 839 1502
1191 1196 1360 1943 2395
582 869 1611 1796 1816
582 869 1611 1796 2419
324 582 869 1611 1796
255 319 388 2403
169 229 574 1680
169 229 574 1816
169 229 574 1046
```

Hình 8: Input file OnlineRetail

```
225 1816 ==> 225 #SUP: 16114 #CONF: 0.5048405113565844
225 720 ==> 225 #SUP: 13311 #CONF: 0.5146871008939975
225 1834 ==> 225 #SUP: 13753 #CONF: 0.5161782883734458
225 1680 ==> 225 #SUP: 10518 #CONF: 0.5226278760220574
720 1336 ==> 225 #SUP: 8081 #CONF: 0.530627397599307
1215 1816 ==> 225 #SUP: 7830 #CONF: 0.5572158365261813
1215 2339 ==> 225 #SUP: 7169 #CONF: 0.5479146324452504
225 1816 1834 ==> 225 #SUP: 11193 #CONF: 0.5421245421245421
225 720 1336 ==> 225 #SUP: 7395 #CONF: 0.5626774847870183
225 1215 1816 ==> 225 #SUP: 7396 #CONF: 0.5751757706868578
```

Hình 9:Output OnlineRetail

@ITEM=9173=Apple Juice 350ml	1079 2032
@ITEM=9174=Apple & Blackcurrant Juice 350ml	2032
@ITEM=1206=Italian Artichoke Purple	1001 4019
@ITEM=1205=Thai Green Chilli	1001
@ITEM=1204=Amami Papadam	1001 1003 1123
@ITEM=1203=Beetroot bunch	1079 2032 4024
@ITEM=1202=Jerusalem Artichokes	1001 1004 1123 1203
@ITEM=1201=Mushroom Brown	1001 1007 1014
@ITEM=1200=beetroot Golden Delicious	1003 1004 1007 1203
@ITEM=1209= Artichoke 2 For 3	1037 2036
@ITEM=1208=Artichoke Purple	1021 1033 1037 2030
@ITEM=1207= Rape	2030 2045 4031
	1073 2022
	1002 1033 1076 1099 1109 2036
	1002 1032 1034

Hình 10:Input của dataset Fruithut

```
1079 ==> 2010 #SUP: 7609 #CONF: 0.6480483637797345
2032 ==> 2010 #SUP: 6260 #CONF: 0.6391373801916933
1037 ==> 2010 #SUP: 7475 #CONF: 0.6575250836120401
1021 ==> 2010 #SUP: 7036 #CONF: 0.6650085275724844
1033 ==> 2010 #SUP: 7782 #CONF: 0.6626831148804935
2045 ==> 2010 #SUP: 12758 #CONF: 0.585358206615457
1073 ==> 2010 #SUP: 5099 #CONF: 0.6822906452245538
1002 ==> 2010 #SUP: 7209 #CONF: 0.659592176446109
1076 ==> 2010 #SUP: 8701 #CONF: 0.6452131938857603
1099 ==> 2010 #SUP: 8586 #CONF: 0.6523410202655485
1109 ==> 2010 #SUP: 5412 #CONF: 0.6400591278640059
1032 ==> 2010 #SUP: 10511 #CONF: 0.6358101037008848
2003 ==> 2010 #SUP: 8106 #CONF: 0.6401431038736738
1008 ==> 2010 #SUP: 9814 #CONF: 0.642551457102099
2002 ==> 2010 #SUP: 5060 #CONF: 0.6756916996047431
2021 ==> 2010 #SUP: 5684 #CONF: 0.6217452498240675
1031 ==> 2010 #SUP: 8236 #CONF: 0.6582078678970374
1058 ==> 2010 #SUP: 6398 #CONF: 0.6752110034385745
1071 ==> 2010 #SUP: 6638 #CONF: 0.6708345887315457
2005 ==> 2010 #SUP: 5361 #CONF: 0.6606976310389853
1017 ==> 2010 #SUP: 11414 #CONF: 0.6209917644997371
1077 ==> 2010 #SUP: 5576 #CONF: 0.6698350071736011
```

Hình 11: Output đã được xử lý của Fruithut

```
Pumpkin Japanese/Kent ==> Banana Cavendish #SUP: 7609 #CONF: 0.6480483637797345
Orange navel ==> Banana Cavendish #SUP: 6260 #CONF: 0.6391373801916933
Garlic loose ==> Banana Cavendish #SUP: 7475 #CONF: 0.6575250836120401
Carrots  ==> Banana Cavendish #SUP: 7036 #CONF: 0.6650085275724844
Eggplant ==> Banana Cavendish #SUP: 7782 #CONF: 0.6626831148804935
Watermelon seedless ==> Banana Cavendish #SUP: 12758 #CONF: 0.585358206615457
Potato Desiree ==> Banana Cavendish #SUP: 5099 #CONF: 0.6822906452245538
Beans green ==> Banana Cavendish #SUP: 7209 #CONF: 0.659592176446109
Potato sweet Gold ==> Banana Cavendish #SUP: 8701 #CONF: 0.6452131938857603
Zucchini green ==> Banana Cavendish #SUP: 8586 #CONF: 0.6523410202655485
Tomatoes Roma ==> Banana Cavendish #SUP: 5412 #CONF: 0.6400591278640059
Cucumber Lebanese ==> Banana Cavendish #SUP: 10511 #CONF: 0.6358101037008848
Apples Pink Lady ==> Banana Cavendish #SUP: 8106 #CONF: 0.6401431038736738
Broccoli ==> Banana Cavendish #SUP: 9814 #CONF: 0.642551457102099
Apples Granny Smith ==> Banana Cavendish #SUP: 5060 #CONF: 0.6756916996047431
Grapes black ==> Banana Cavendish #SUP: 5684 #CONF: 0.6217452498240675
Cucumber continental ==> Banana Cavendish #SUP: 8236 #CONF: 0.6582078678970374
Mushroom Cup ==> Banana Cavendish #SUP: 6398 #CONF: 0.6752110034385745
Potato brushed ==> Banana Cavendish #SUP: 6638 #CONF: 0.6708345887315457
Apples Royal Gala ==> Banana Cavendish #SUP: 5361 #CONF: 0.6606976310389853
Capsicum red ==> Banana Cavendish #SUP: 11414 #CONF: 0.6209917644997371
Potato washed ==> Banana Cavendish #SUP: 5576 #CONF: 0.6698350071736011
Onion brown ==> Banana Cavendish #SUP: 6999 #CONF: 0.6649521360194314
Nectarine White ==> Banana Cavendish #SUP: 5687 #CONF: 0.5941621241427818
Capsicum green ==> Banana Cavendish #SUP: 5494 #CONF: 0.6698216235893703
Ginger ==> Banana Cavendish #SUP: 6265 #CONF: 0.6614525139664804
```

Hình 12: Output đã chuyển đổi về trạng thái chưa xử lý

Ở tập dữ liệu Fruithut, do dữ liệu đầu vào bao gồm tên các sự kiện và các con số là ID của từng loại sự kiện khác nhau nên khi thực thi thuật toán, chúng ta sẽ chỉ làm việc với các con số (ID). Sau khi có kết quả như hình 11 chúng ta sẽ thực thi hàm chuyển đổi để biến đổi các ID trong dữ liệu đầu ra thành các tên gọi sự kiện ứng với từng ID trong dữ liệu đầu vào như hình 12.

Thử nghiệm trên ba bộ dữ liệu khác:

Bảng 2: Thông số của ba bộ dữ liệu khác

Tên bộ dữ liệu	Số lượng giao dịch	Loại sự kiện	Tham số
BMS WebView 1	59602	497	(200,0.5,5,5,5)
retail	9284	2626	(4000,0.5,3,3,3)
foodmart	4141	1559	(5,0.5,5,5,5)

Bảng 3: Kết quả thực thi của ba bộ dữ liệu

Tên bộ dữ liệu	Thời gian (s) (xấp xỉ)	Bộ nhớ (MB) (xấp xỉ)
BMS WebView 1	471	410
retail	383	461
foodmart	14	33

```
10307 10311 12487
12559
12695 12703 18715
10311 12387 12515 12691 12695 12699 12703 12823 12831 12847 18595 18679 18751
10291 12523 12531 12535 12883
12523 12539 12803 12819
12819
12471 12491 12531 12535 12567 12663 12667 12823 18447 18507 18691
12487
12547 12815 12895
12527 12799 12807 12819 12879
12807
12875
12407 12507 12519 12547
12479 12483 12487 12527 12535 12539 12555 12715 12723 18447 18619 18691 18815 18819 18831
12403
12875
12371 12419
12823
18547 18587 18735
12523
12459 12523 12531 12559 12663 12667 12687 12847 12851
12621 12647 12667 12807 18707 18715 18747 18755 18763 18767
12647
12395 12403 12431 12447 12451 12515 12679 12715 12727 18431
12511 12627 18507 18755
12559
12663
12647
```

Hình 13: Input BMS WebView 1

```
10307 12663 33469 ==> 33469 #SUP: 202 #CONF: 0.5
10307 33449 33469 ==> 33469 #SUP: 572 #CONF: 0.513986013986014
12691 33449 33469 ==> 33449 #SUP: 270 #CONF: 0.5037037037037037
12831 33449 33469 ==> 33469 #SUP: 280 #CONF: 0.5
12819 33449 33469 ==> 33469 #SUP: 251 #CONF: 0.5139442231075697
12663 33449 33469 ==> 33449 #SUP: 391 #CONF: 0.5089514066496164
12663 33449 33469 ==> 33469 #SUP: 391 #CONF: 0.5063938618925832
12815 12895 33469 ==> 33449 #SUP: 241 #CONF: 0.5020746887966805
12815 12895 33469 ==> 33469 #SUP: 241 #CONF: 0.5269709543568465
12895 33449 33469 ==> 33469 #SUP: 773 #CONF: 0.5148771021992238
12483 33449 33469 ==> 33469 #SUP: 403 #CONF: 0.5012406947890818
12715 33449 33469 ==> 33469 #SUP: 252 #CONF: 0.5
12621 33449 33469 ==> 33469 #SUP: 264 #CONF: 0.5037878787878788
12751 33449 33469 ==> 33469 #SUP: 234 #CONF: 0.5341880341880342
12463 33449 33469 ==> 33449 #SUP: 200 #CONF: 0.52
12463 33449 33469 ==> 33469 #SUP: 200 #CONF: 0.525
10295 33449 33469 ==> 33469 #SUP: 454 #CONF: 0.5066079295154186
10331 33449 33469 ==> 33449 #SUP: 220 #CONF: 0.5
10331 33449 33469 ==> 33469 #SUP: 220 #CONF: 0.5409090909090909
33433 33449 33469 ==> 33449 #SUP: 512 #CONF: 0.501953125
10307 10311 33449 33469 ==> 33469 #SUP: 218 #CONF: 0.5045871559633027
10307 12895 33449 33469 ==> 33469 #SUP: 276 #CONF: 0.5688405797101449
10307 10315 33449 33469 ==> 33469 #SUP: 237 #CONF: 0.510548523206751
10307 33433 33449 33469 ==> 33469 #SUP: 202 #CONF: 0.5148514851485149
12895 33433 33449 33469 ==> 33469 #SUP: 247 #CONF: 0.5222672064777328
12827 12895 33449 33469 ==> 33449 #SUP: 209 #CONF: 0.507177033492823
12827 12895 33449 33469 ==> 33469 #SUP: 209 #CONF: 0.5023923444976076
10295 10307 33449 33469 ==> 33469 #SUP: 295 #CONF: 0.5220338983050847
10295 12895 33449 33469 ==> 33469 #SUP: 220 #CONF: 0.5136363636363637
```

Hình 14: Output BMS WebView 1

```
31 32 33
34 35 36
37 38 39 40 41 42 43 44 45 46 47
39 40 48 49
39 40 49 50 51 52 53 54 55 56 57 58 59
33 42 60 61 62 63
4 40 49
64 65 66 67 68 69
33 70
49 71 72 73
40 74 75 76 77 78 79 80
37 39 40 42 49 80 81 82
83 84 85
42 86 87 88 89
40 49 90 91 92 93 94 95 96 97 98 99 100 101 102
37 39 40 49 90
40 42 103 104 105 106 107 108 109
39 40 42 110 111
40 112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129 130 131 132 133 134
49 135 136 137
40 49 138 139 140 141 142 143 144 145 146 147 148 149 150
40 151 152 153
```

Hình 15: Input retail

```
33 ==> 40 #SUP: 15167 #CONF: 0.725456583371794
33 ==> 40 49 #SUP: 15167 #CONF: 0.6561614030460869
33 ==> 49 #SUP: 15167 #CONF: 0.6701391178215863
39 ==> 40 #SUP: 15596 #CONF: 0.722428827904591
39 ==> 40 49 #SUP: 15596 #CONF: 0.6470248781738908
39 ==> 49 #SUP: 15596 #CONF: 0.6602333931777379
42 ==> 40 #SUP: 14945 #CONF: 0.6410170625627301
42 ==> 40 49 #SUP: 14945 #CONF: 0.5628638340582135
42 ==> 49 #SUP: 14945 #CONF: 0.5917029106724657
49 ==> 40 #SUP: 42135 #CONF: 0.5352082591669634
66 ==> 49 #SUP: 4472 #CONF: 0.7853309481216458
66 ==> 40 #SUP: 4472 #CONF: 0.846824686940966
66 ==> 40 49 #SUP: 4472 #CONF: 0.7983005366726297
33 39 ==> 40 #SUP: 7275 #CONF: 0.8248797250859107
33 39 ==> 40 49 #SUP: 7275 #CONF: 0.7718213058419244
33 39 ==> 49 #SUP: 7275 #CONF: 0.7726460481099656
33 40 ==> 40 #SUP: 14121 #CONF: 0.7347921535302032
33 40 ==> 40 49 #SUP: 14121 #CONF: 0.6767226117130515
33 40 ==> 49 #SUP: 14121 #CONF: 0.7078818780539622
33 42 ==> 40 #SUP: 6490 #CONF: 0.7755007704160246
33 42 ==> 40 49 #SUP: 6490 #CONF: 0.7157164869029276
33 42 ==> 40 42 #SUP: 6490 #CONF: 0.5602465331278891
33 42 ==> 40 42 49 #SUP: 6490 #CONF: 0.5326656394453004
33 42 ==> 49 #SUP: 6490 #CONF: 0.7318952234206472
33 42 ==> 42 #SUP: 6490 #CONF: 0.5244992295839753
33 42 ==> 42 49 #SUP: 6490 #CONF: 0.5326656394453004
33 49 ==> 40 #SUP: 13537 #CONF: 0.7588091896284258
33 49 ==> 40 49 #SUP: 13537 #CONF: 0.6769594444854842
33 49 ==> 49 #SUP: 13537 #CONF: 0.6824259437098323
```

Hình 16: Output retail

```
214 763 260
778 195 385 961
12 934 871 328
715 359 282
586 16 898 1368 785 975 1551
151 1299 557
540 1276 1453 1336 128 224 667
1346 803
1147 258
1440 1350 475 1130 308
497 701
1222 354 1502 760 1430 248 774
1234 1505
27 110 1419 610 1065 843 366 1381
638 1347 1191
16 925 1314 1291 216 326 409
646 157 480 657 540 890 1399 284
404 128 544 854
1236 747 1070
1255 1498 1182 573
623 279 895 606
393 942 853 1535
1358 749
892 1003 1086
445 35 851 1520 1483
668 512 1394 1450 867 1471 75 318
1358 388 1324
1108 1551 222 266
130 933 910 1393 1089 1172
1156 973 1522 1432 557 68 390
```

Hình 17: Input foodmart

```
552 ==> 1350 #SUP: 6 #CONF: 0.5
453 ==> 498 #SUP: 6 #CONF: 0.5
1176 ==> 1378 #SUP: 7 #CONF: 0.5714285714285714
901 ==> 1200 #SUP: 6 #CONF: 0.5
901 ==> 1110 #SUP: 6 #CONF: 0.5
782 ==> 1372 #SUP: 8 #CONF: 0.625
116 ==> 1384 #SUP: 5 #CONF: 0.6
686 ==> 1224 #SUP: 6 #CONF: 0.5
799 ==> 1389 #SUP: 9 #CONF: 0.5555555555555556
632 ==> 1110 #SUP: 7 #CONF: 0.5714285714285714
335 ==> 274 #SUP: 6 #CONF: 0.5
105 930 ==> 333 #SUP: 5 #CONF: 0.6
405 1497 ==> 1146 #SUP: 5 #CONF: 0.6
```

Hình 18: Output foodmart

3.2 Nhận xét:

Với tập dữ liệu có kích thước lớn thì thuật toán POERM sẽ tiêu tốn nhiều thời gian thực thi cũng như bộ nhớ hơn so với tập dữ liệu có kích thước nhỏ. Do đó, cần cải thiện thời gian thực thi hoặc lượng bộ nhớ tiêu tốn của thuật toán.

3.3 Cải thiện thuật toán:

3.3.1 Ý tưởng:

Thay đổi cấu trúc dữ liệu được sử dụng trong thuật toán. Cụ thể, thay vì kết hợp cả hai kiểu dữ liệu Lists và Numpy Arrays để lưu trữ và xử lý, chúng ta sẽ chỉ dùng kiểu dữ liệu Lists.

3.3.2 Nguyên nhân:

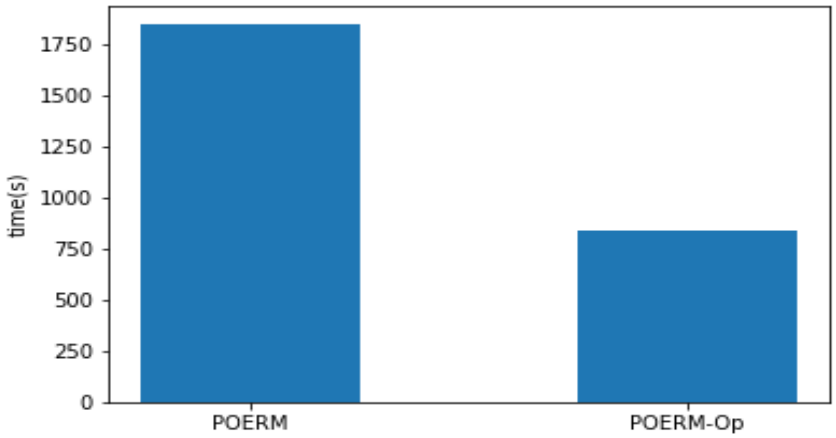
Về mặt lý thuyết, Numpy Arrays có ba điểm ưu việt hơn Lists. Numpy Arrays sẽ chiếm ít không gian bộ nhớ hơn, có thời gian thực thi nhanh hơn khi xử lý trên tập dữ liệu có kích thước lớn và thư viện có sẵn các hàm tối ưu hơn Lists.

Nhưng trong một vài trường hợp Numpy Arrays sẽ tỏ ra kém hiệu quả hơn Lists. Chẳng hạn các thao tác thêm mới phần tử vào Lists/Numpy Arrays. Cụ thể, trong thuật toán POERM, các thao tác tính toán – điều giúp Numpy Arrays hiệu quả hơn Lists chiếm rất ít, đa số các thao tác so sánh, thêm mới phần tử và thực hiện các vòng lặp.

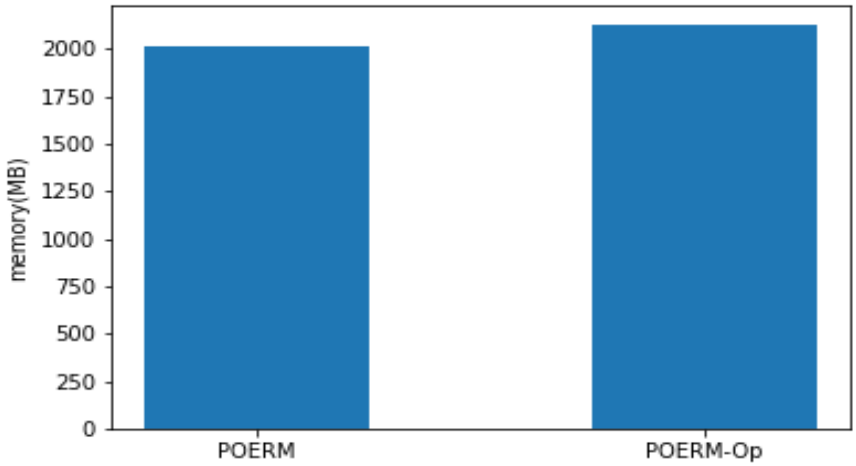
3.4 Kết quả so sánh giữa hai phiên bản POERM:

3.4.1 Tập OnlineRetail:

Các tham số: minSupport = 7000, minConfidence = 0.5, XSpan = YSpan = XYSpan = 5.



Hình 19: Kết quả so sánh về thời gian trên tập OnlineRetail



Hình 20: Kết quả so sánh về bộ nhớ trên tập OnlineRetail

Bảng 4: So sánh 2 phiên bản POERM và POERM-Op trên tập OnlineRetail

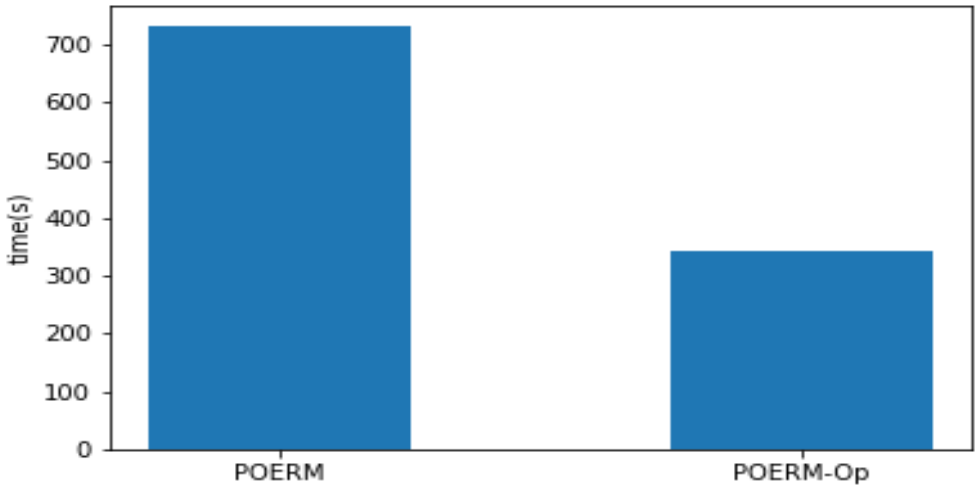
Thuật toán	Thời gian (s)	Bộ nhớ (MB)
POERM	≈1848	≈2012
POERM-Op	≈836	≈2127

Nhận xét:

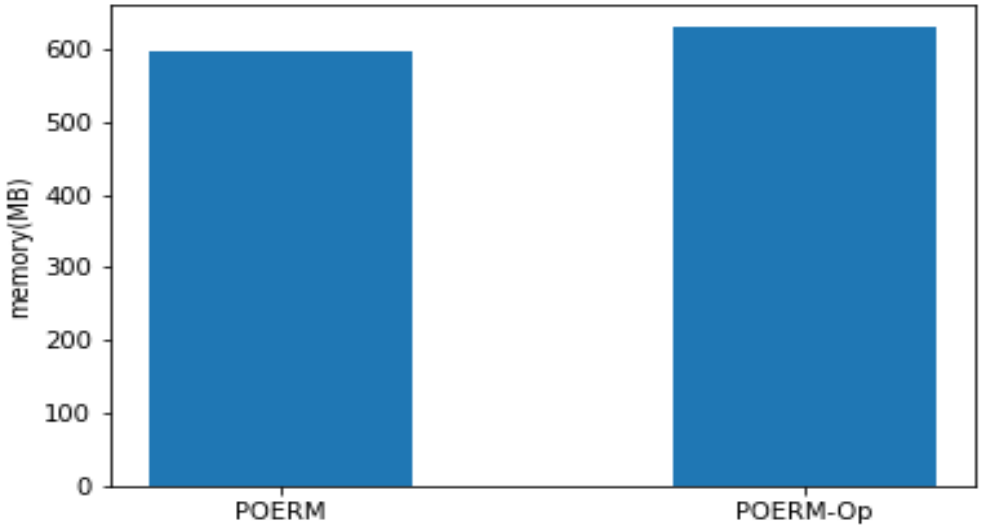
Thời gian thực thi giảm đáng kể, cụ thể giảm đi hơn 2 lần, tuy nhiên bộ nhớ tăng không đáng kể, cụ thể tăng khoảng 1.05 lần.

3.4.2 Tập Fruithut:

Các tham số: minSupport = 5000, minConfidence = 0.5, XSpan = YSpan = XYSpan = 5.



Hình 21: Kết quả so sánh về thời gian trên tập Fruithuit



Hình 22: Kết quả so sánh về bộ nhớ trên tập Fruithuit

Bảng 5: So sánh 2 phiên bản POERM và POERM-Op trên tập Fruithut

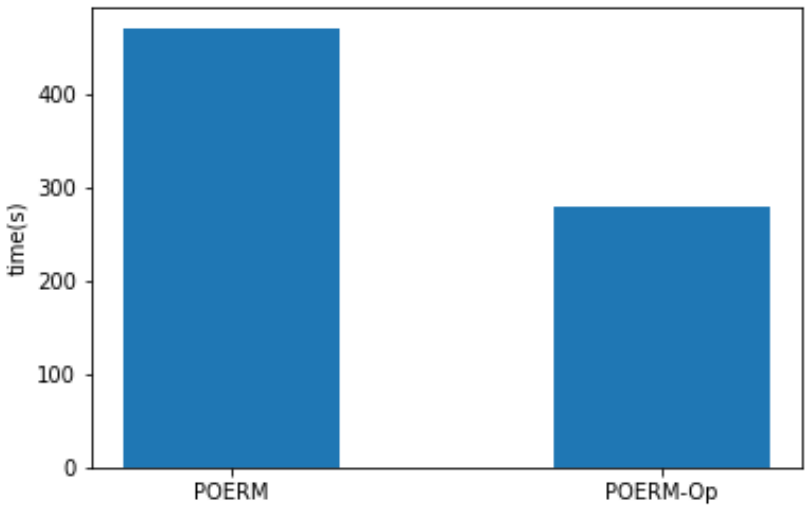
Thuật toán	Thời gian (s)	Bộ nhớ (MB)
POERM	≈733	≈595
POERM-Op	≈342	≈617

Nhận xét:

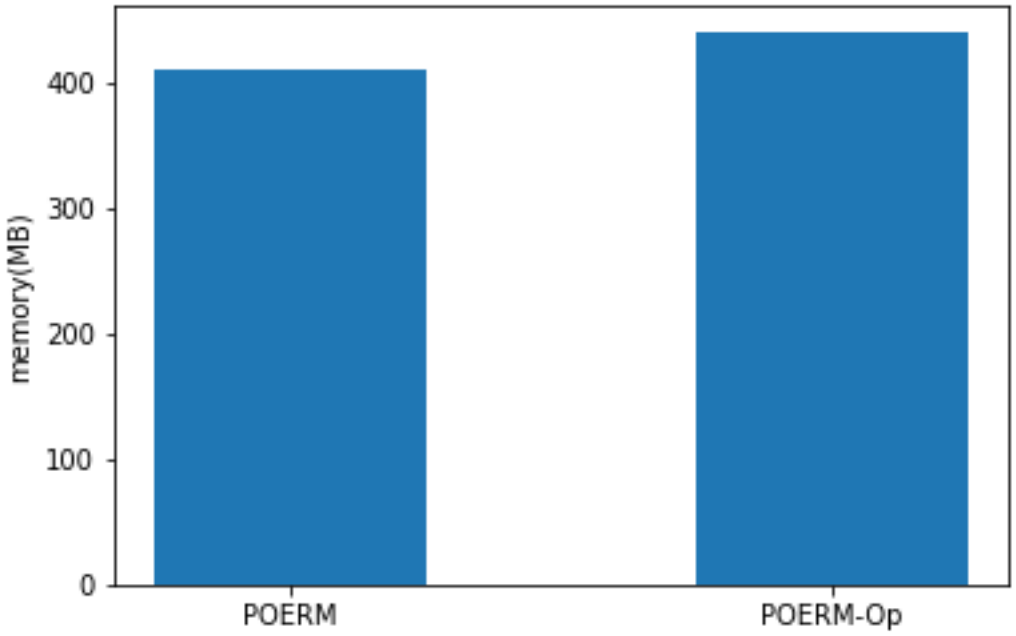
Thời gian thực thi giảm đi khoảng 1.68 lần, tuy nhiên bộ nhớ tăng không đáng kể, cụ thể tăng khoảng 22MB.

3.4.3 Tập BMS WebView 1:

Các tham số: minSupport = 200, minConfidence = 0.5, XSpan = YSpan = XYSpan = 5.



Hình 23: Kết quả so sánh về thời gian trên tập BMS WebView 1



Hình 24: Kết quả so sánh về bộ nhớ trên tập BMS WebView 1

Bảng 6: So sánh 2 phiên bản POERM và POERM-Op trên tập BMS Web View 1

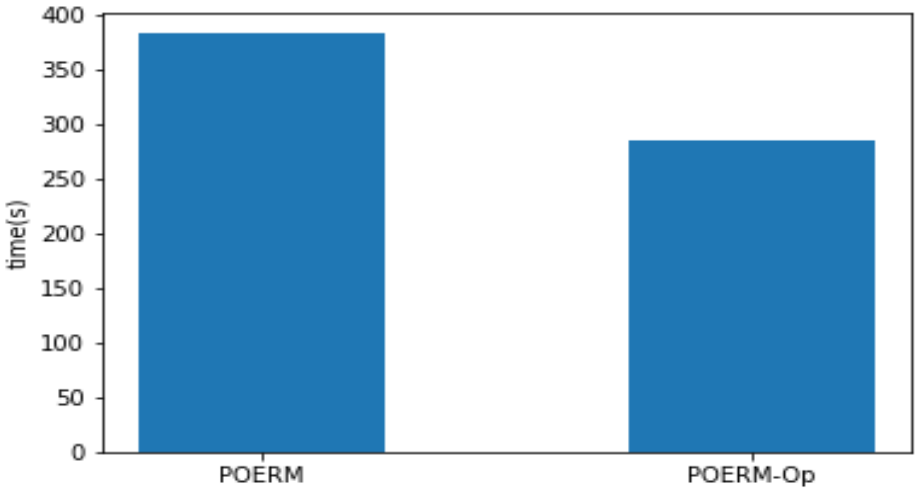
Thuật toán	Thời gian (s)	Bộ nhớ (MB)
POERM	≈471	≈410
POERM-Op	≈280	≈440

Nhận xét:

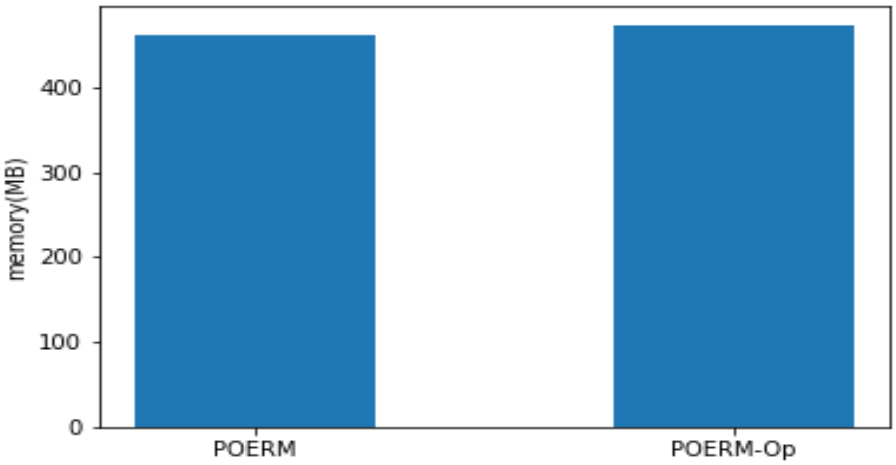
Thời gian thực thi giảm đáng kể, cụ thể giảm đi khoảng 2.14 lần, tuy nhiên bộ nhớ tăng không đáng kể, cụ thể tăng khoảng 30MB.

3.4.4 Tập Retail:

Các tham số: minSupport = 200, minConfidence = 0.5, XSpan = YSpan = XYSpan = 3.



Hình 25: Kết quả so sánh về thời gian trên tập Retail



Hình 26: Kết quả so sánh về bộ nhớ trên tập Retail

Bảng 7: So sánh 2 phiên bản POERM và POERM-Op trên tập Retail

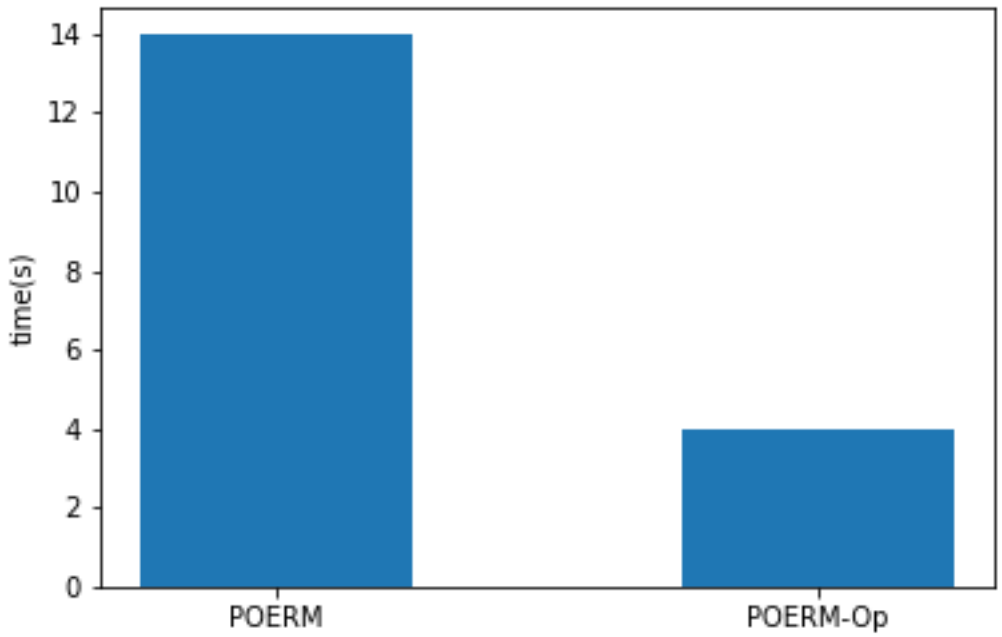
Thuật toán	Thời gian (s)	Bộ nhớ (MB)
POERM	≈383	≈461
POERM-Op	≈285	≈473

Nhận xét:

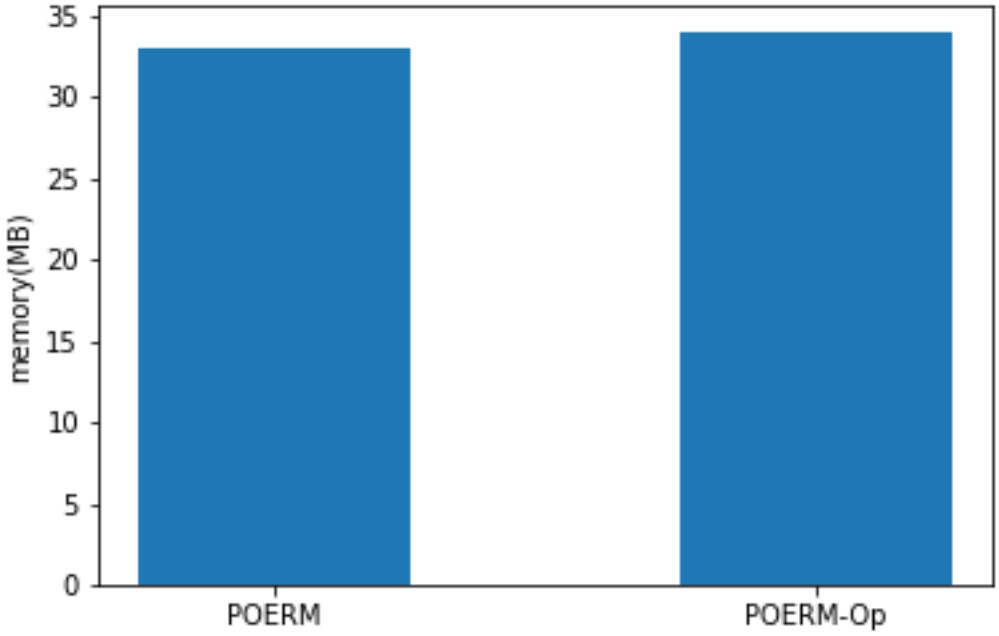
Thời gian thực thi giảm đi khoảng 1.34 lần, tuy nhiên bộ nhớ tăng không đáng kể, cụ thể tăng khoảng 12MB.

3.4.5 Tập Foodmart:

Các tham số: minSupport = 5, minConfidence = 0.5, XSpan = YSpan = XYSpan = 5.



Hình 27: Kết quả so sánh về thời gian trên tập Foodmart



Hình 28: Kết quả so sánh về bộ nhớ trên tập Foodmart

Bảng 8: So sánh 2 phiên bản POERM và POERM-Op trên tập Foodmart

Thuật toán	Thời gian (s)	Bộ nhớ (MB)
POERM	≈14	≈33
POERM-Op	≈4	≈34

Nhận xét:

Thời gian thực thi giảm đáng kể, cụ thể giảm đi khoảng 3.5 lần, tuy nhiên bộ nhớ tăng không đáng kể, cụ thể tăng khoảng 1MB.

3.5 Khảo sát ảnh hưởng của các tham số:

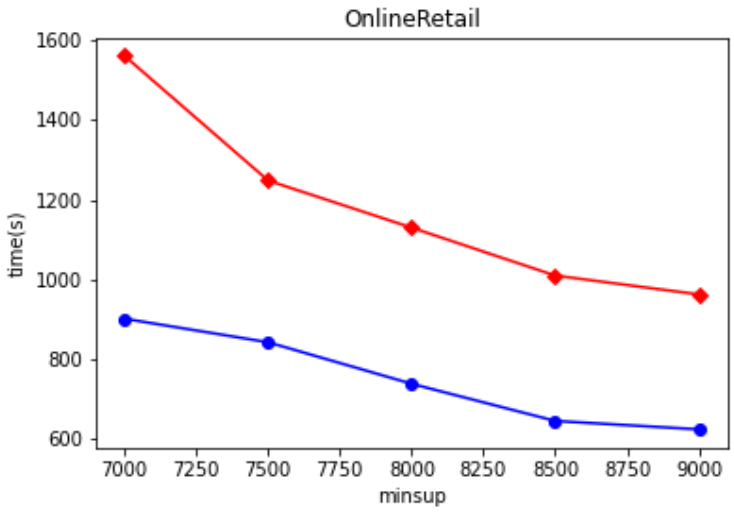
Trong mỗi thử nghiệm, một tham số sẽ thay đổi trong khi các tham số khác là cố định. Bởi vì thuật toán có năm tham số: minSupport, minConfidence , XSpan, YSpan và XYSpan và không gian không cho phép đánh giá từng tham số riêng biệt nên ba tham số ràng buộc thời gian XSpan, YSpan và XYSpan được đặt thành cùng một giá trị được gọi là Span và sẽ chỉ khảo sát trên hai bộ dữ liệu chuẩn là Online Retail và Fruithut. Các thử nghiệm được khảo sát trên hai tập dữ liệu chuẩn OnlineRetail và Fruithut.

Quy ước:

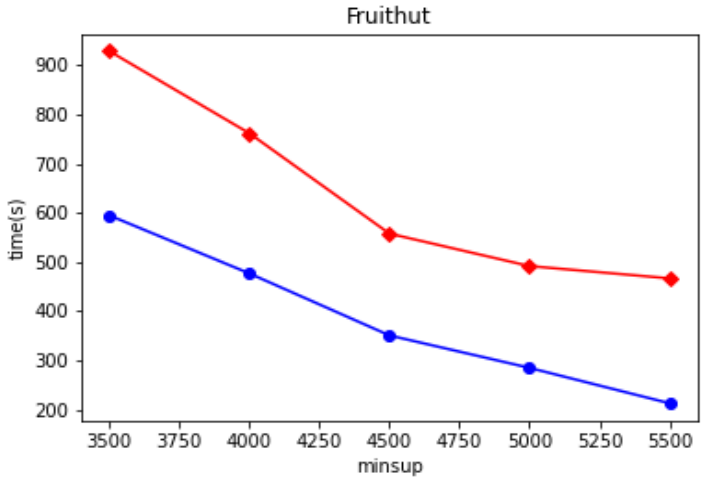
Hình 29: Quy ước về biểu đồ

3.5.1 Khảo sát sự ảnh hưởng về thời gian:

3.5.1.1 Ảnh hưởng của minSupport:



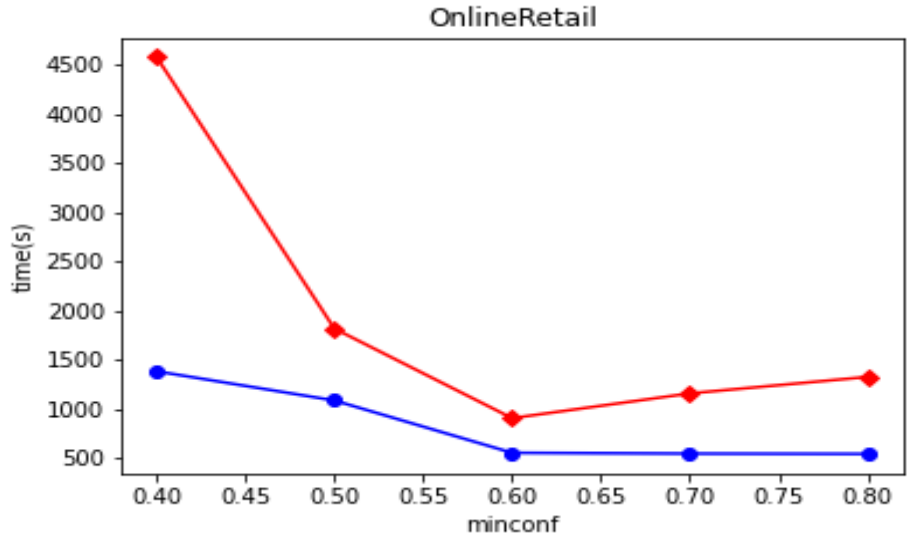
Hình 30: Ảnh hưởng của minsup đến thời gian trên tập OnlineRetail



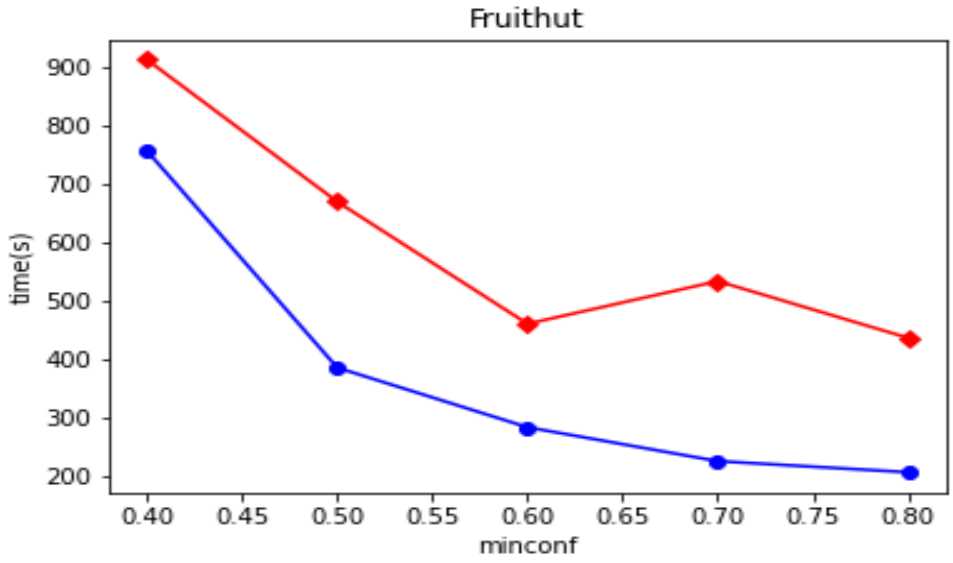
Hình 31: Ảnh hưởng của minsup đến thời gian trên tập Fruithut

Hai hình trên cho thấy khi minsup càng tăng thì thời gian thực thi càng giảm. Lý do là khi minsup càng nhỏ thì thuật toán sẽ phải xem xét nhiều tập tiền tố ($sup(X) \geq minsup$) và hậu tố ($sup(Y) \geq minsup \times minconf$) hơn.

3.5.1.2 Ảnh hưởng của minConfidence:



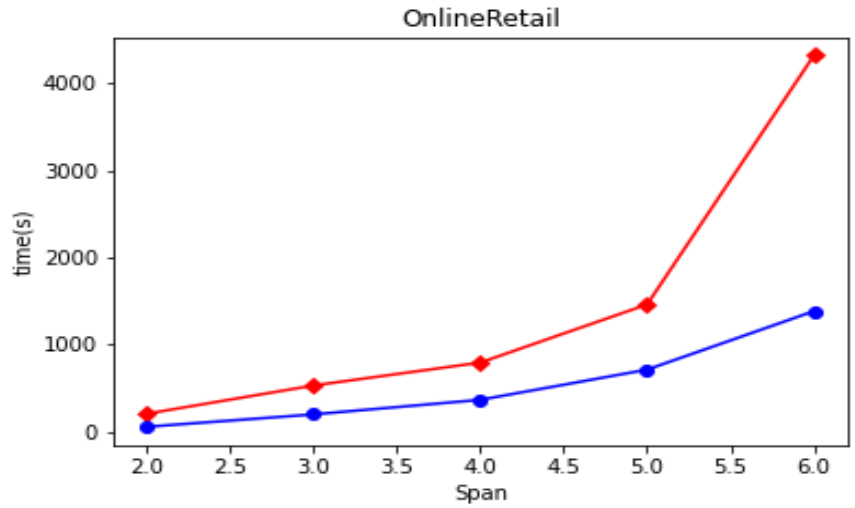
Hình 32: Ảnh hưởng của minconf đến thời gian trên tập OnlineRetail



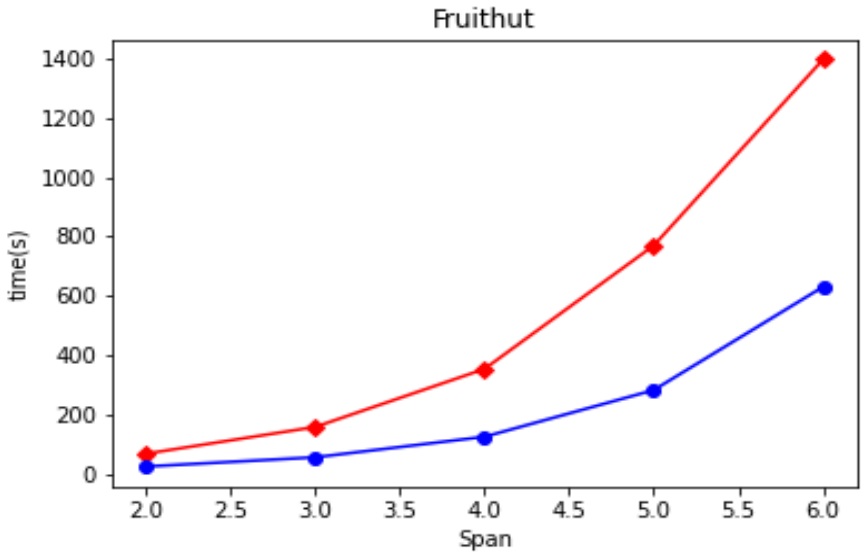
Hình 33: Ảnh hưởng của minconf đến thời gian trên tập Fruithut

Hai hình trên cho thấy khi minconf càng giảm thì thời gian thực thi càng tăng.

3.5.1.3 Ảnh hưởng của Span:



Hình 34: Ảnh hưởng của Span đến thời gian trên tập OnlineRetail

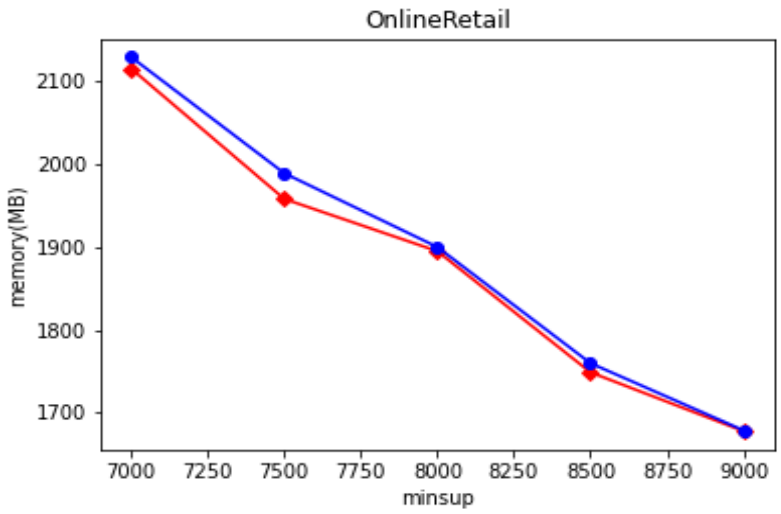


Hình 35: Ảnh hưởng của Span đến thời gian trên tập Fruithut

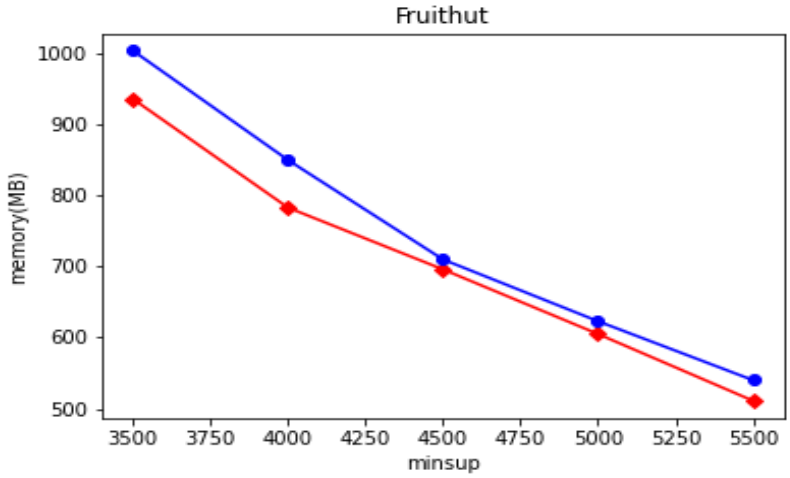
Hai hình trên cho thấy khi tham số Span sẽ tỉ lện thuận với thời gian thực thi do khi Span tăng thì thuật toán phải tìm nhiều tiền tố hơn so với trước.

3.5.2 Khảo sát ảnh hưởng của bộ nhớ:

3.5.2.1 minSupport:



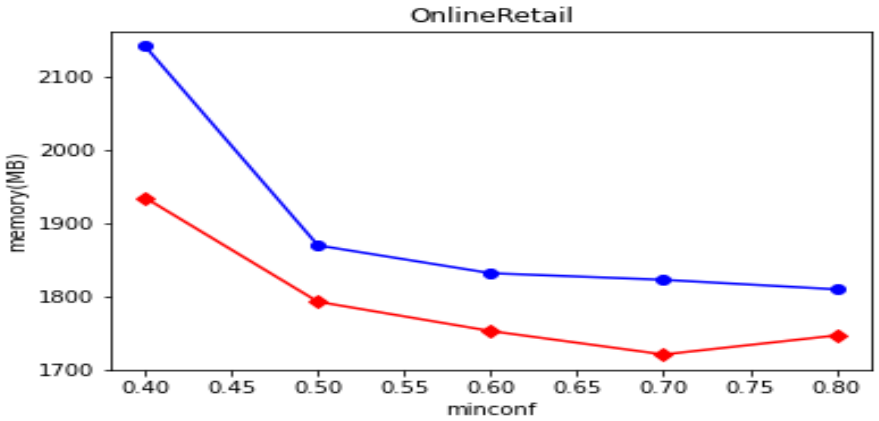
Hình 36: Ảnh hưởng của minsup đến bộ nhớ trên tập OnlineRetail



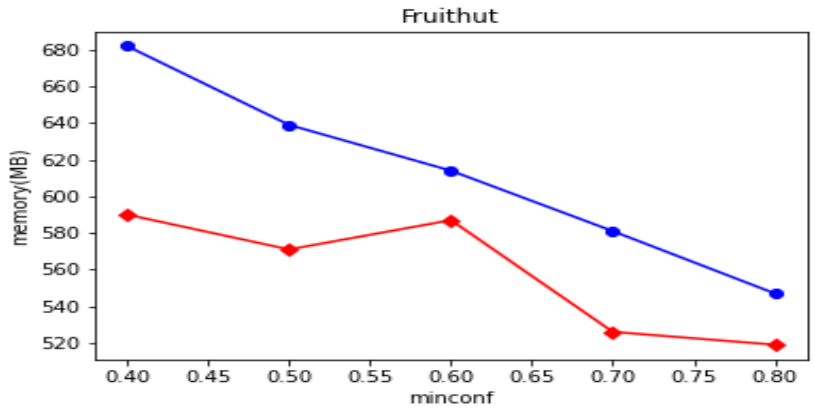
Hình 37: Ảnh hưởng của minsup đến bộ nhớ trên tập Fruithut

Khi minsup tăng lượng bộ nhớ sử dụng sẽ giảm do các sự kiện thuật toán phải xem xét sẽ ít hơn.Minsup ảnh hưởng ít đến bộ nhớ do chênh lệch về bộ nhớ giữa hai thuật toán không quá đáng kể.

3.5.2.2 minConfidence:



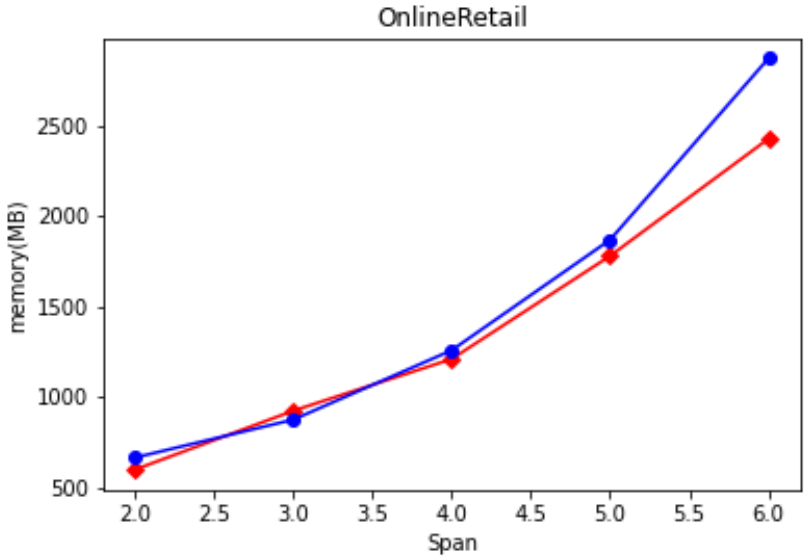
Hình 38: Ảnh hưởng của minconf đến thời gian trên tập OnlineRetail



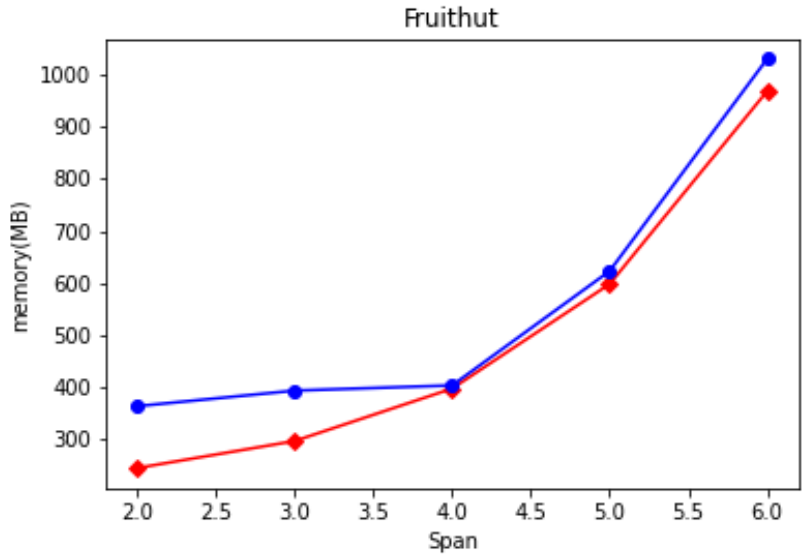
Hình 39: Ảnh hưởng của minconf đến bộ nhớ trên tập Fruithut

Nhìn chung khi minconf càng tăng thì lượng bộ nhớ sử dụng càng giảm.

3.5.2.3 Span:



Hình 40: Ảnh hưởng của Span đến bộ nhớ trên tập OnlineRetail



Hình 41: Ảnh hưởng của Span đến bộ nhớ trên tập Fruithut

Tương tự thời gian thực thi, khi Span tăng thì mức sử dụng bộ nhớ tăng do không gian tìm kiếm tăng lên.

3.5.3 Các luật được khám phá:

Bảng 9: Ví dụ một số luật được khám phá

LUẬT	SUPPORT	CONFIDENCE (xấp xỉ)
Zucchini green Field Tomatoes ==> Banana Cavendish	5127	0.7138
Lettuce Iceberg Field Tomatoes ==> Banana Cavendish	5133	0.7162
Cucumber Lebanese Field Tomatoes ==> Banana Cavendish	6152	0.7064
Cucumber continental Field Tomatoes ==> Banana Cavendish	5023	0.7203
Broccoli Field Tomatoes ==> Banana Cavendish	5859	0.715
Lettuce Iceberg Field Tomatoes ==> Banana Cavendish	5133	0.7161

Field Tomatoes Watermelon seedless ==> Banana Cavendish	6131	0.7017
Capsicum red Field Tomatoes Banana Cavendish ==> Banana Cavendish	6352	0.7067
Cucumber Lebanese Field Tomatoes Banana Cavendish ==> Banana Cavendish	5023	0.7203

Bảng 9 là ví dụ cho thấy một số quy tắc được khám phá từ tập dữ liệu Fruithut có độ tin cậy khá cao khoảng 70% nên từ đó có thể cung cấp cái nhìn tổng quát về tập dữ liệu, giúp các cửa hàng bán lẻ hoặc công ty am hiểu về các lựa chọn của khách hàng và có thể đưa ra các phương án kinh doanh phù hợp.

Chương 4: Kết luận và hướng phát triển

4.1 Kết luận:

Trong khóa luận này, từ kết quả nghiên cứu của Philippe Fournier-Viger và cộng sự, nhóm đã nghiên cứu các loại luật mới được đề xuất gọi là các luật có thứ tự bán phần, trong đó các sự kiện trong luật được sắp xếp bán phần, dùng để tìm các tập luật tổng quát hơn.

Để tìm một cách hiệu quả tất cả các luật này trong một chuỗi, một thuật toán hiệu quả có tên là POERM – Khai thác tập luật có thứ tự bán phần đã được nghiên cứu.

Nhóm đã đề xuất cải thiện về mặt cấu trúc dữ liệu trong thuật toán để cho hiệu suất tốt hơn.

Một số đánh giá thực nghiệm trên một số tập dữ liệu cho thấy thuật toán POERM có hiệu suất tuyệt vời, đồng thời khảo sát ảnh hưởng của các tham số đến thời gian thực thi và lượng bộ nhớ đã sử dụng khi thực thi thuật toán.

4.2 Hướng phát triển:

Một số khả năng phát triển trong tương lai: Mở rộng POERM để xử lý dữ liệu streaming và chạy thuật toán trên dữ liệu lớn hoặc môi trường đa luồng để hưởng lợi từ tính song song; Xem xét các dữ liệu phức tạp hơn, chẳng hạn như các sự kiện được tổ chức theo một phân loại [3] hoặc một luồng [17]; Phát triển mô hình dự đoán chuỗi dựa trên các POER; Các chức năng lựa chọn mẫu khác cũng được xem xét như tiện ích [13,18] và độ hiếm [10].

Tài liệu tham khảo

[1] Ao, X., Luo, P., Li, C., Zhuang, F., He, Q.: Online frequent episode mining. In: *2015 IEEE 31st International Conference on Data Engineering*, pp. 891–902. IEEE,2015.

[2] Ao, X., Luo, P., Wang, J., Zhuang, F., He, Q.: Mining precise-positioning episode rules from event sequences. IEEE *Transactions on Knowledge and Data Engineering 30(3),* pp.530–543, 2017.

[3] Ao, X., Shi, H., Wang, J., Zuo, L., Li, H., He, Q.: Large-scale frequent episode mining from complex event sequences with hierarchies. *ACM Transactions on Intelligent Systems and Technology (TIST) 10(4)*, pp.1–26, 2019.

[4] Fahed, L., Brun, A., Boyer, A.: Deer: Distant and essential episode rules for early prediction. *Expert Systems with Applications 93*, pp.283–298, 2018.

[5] Fournier-Viger, P., Lin, J.C.W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z.,Lam, H.T.: The spmf open-source data mining library version 2. In: *Joint European conference on machine learning and knowledge discovery in databases*, pp. 36–40, Springer,2016.

[6] Fournier-Viger, P., Lin, J.C.W., Kiran, U.R., Koh, Y.S.: A survey of sequential pattern mining. *Data Science and Pattern Recognition 1(1)*, pp. 54–77, 2017.

[7] Fournier-Viger, P., Wu, C.W., Tseng, V.S., Cao, L., Nkambou, R.: Mining partially ordered sequential rules common to multiple sequences. *IEEE*

Transactions on Knowledge and Data Engineering 27(8), pp. 2203–2216,2015.

[8] Fournier-Viger, P., Yang, Y., Yang, P., Lin, J.C.W., Yun, U.: Tke: Mining top k-frequent episodes. In: *Proc. 33rd Intern. Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems.* Springer, 2020.

[9] Huang, K., Chang, C.: Efficient mining of frequent episodes from complex sequences. *Inf. Syst. 33(1),* pp.96–114, 2008.

[10] Koh, Y.S., Ravana, S.D.: Unsupervised rare pattern mining: a survey. *ACM Transactions on Knowledge Discovery from Data (TKDD) 10(4),* pp.1–29,2016, Mannila, H., Toivonen, H., Verkamo, A.I.: Discovering frequent episodes in sequences. In: Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining, 1995.

[11] Song, W., Huang, C.: Mining high average-utility itemsets based on particle swarm optimization. *Data Science and Pattern Recognition 4(2),* pp. 19–32, 2020.

[12] Su, M.Y.: Applying episode mining and pruning to identify malicious online attacks. *Computers & Electrical Engineering 59,* pp.180–188, 2017.

[13] Truong, T., Duong, H., Le, B., Fournier-Viger, P.: Fmaxclohusm: An efficient algorithm for mining frequent closed and maximal high utility sequences. *Engineering Applications of Artificial Intelligence 85*, pp.1–20, 2019.

[14] Wenzhe, L., Qian, W., Luqun, Y., Jiadong, R., Davis, D.N., Changzhen, H.: Mining frequent intra-sequence and inter-sequence patterns using

bitmap with a maximal span. In: *Proc. 14th Web Inf. Syst. and Applications Conf*. pp. 56–61. IEEE, 2017.

[15] You, T., Li, Y., Sun, B., Du, C.: Multi-source data stream online frequent episode mining. *IEEE Access 8*, pp.107465–107478, 2020.

[16] Yun, U., Nam, H., Lee, G., Yoon, E.: Efficient approach for incremental high utility pattern mining with indexed list structure. *Future Generation Computer Systems 95*, 221–239, 2019.

[17] Ginni:https://www.tutorialspoint.com/why-use-support-and-confidence-in-data-mining#, 2022.

[18] Văn Thị Thiên Trang, Khai thác luật tuần từ trên cơ sở dữ liệu chuỗi, Luận văn thạc sĩ Ngành Hệ thống thông tin, Thành phố Hồ Chí Minh, 2010.