ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

NGUYỄN DUY CHINH

KHAI THÁC TOP-K SỰ KIỆN ĐỒNG XUẤT HIỆN VỚI BITTABLE

ĐỒ ÁN THẠC SĨ: CÔNG NGHỆ THÔNG TIN

TP. Hồ Chí Minh – 2017

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

NGUYỄN DUY CHINH

KHAI THÁC TOP-K SỰ KIỆN ĐỒNG XUẤT HIỆN VỚI BITTABLE

Chuyên ngành: Hệ Thống Thông Tin
Mã số chuyên ngành: 60480104

ĐỒ ÁN THẠC SĨ: CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC
TS. NGUYỄN NGỌC THẢO

TP. Hồ Chí Minh – 2017

LỜI CẢM ƠN

Tôi xin gửi lời cảm ơn chân thành, lòng biết ơn sâu sắc đến tất cả các quý thầy cô khoa Công Nghệ Thông Tin trường đại học Khoa Học Tự Nhiên TP.HCM đã giảng dạy bằng tất cả sự nhiệt tình trong quá trình tôi học tại đây.

Đặc biệt tôi xin gởi lời cảm ơn chân thành nhất và lòng biết ơn sâu sắc nhất đến giảng viên hướng dẫn, TS. Nguyễn Ngọc Thảo vì đã tận tình hướng dẫn và giúp đỡ tôi trong quá trình làm đồ án.

Xin cảm ơn Cha Mẹ đã sinh ra con, nuôi nấng và dạy dỗ con, động viên con trong học tập cũng như công việc.

Cảm ơn vợ tôi đã luôn động viên và là động lực cho tôi hoàn thành đồ án này.

Cuối cùng, tôi xin gửi lời cảm ơn đến các bạn của tôi ở lớp cao học khóa 23 đại học Khoa Học Tự Nhiên TP.HCM, đã giúp đỡ và động viên tôi rất nhiều để tôi có thể hoàn thành được đồ án tốt nghiệp này.

Nguyễn Duy Chinh

MỤC LỤC

DANH MỤC CÁC BẢNG

DANH MỤC CÁC HÌNH VẼ, ĐỒ THỊ

DANH MỤC CÁC TỪ VIẾT TẮT

STT	Từ viết tắt	Ý nghĩa
01	FIM	Frequent Itemset Mining
02	NT	Naive Hunting algorithm
03	NTI	Naive Hunting algorithm withInverted list index
04	NT-TA	NT with TA pruning
05	NTI-TA	NTI with TA pruning
06	PT	Pi-Tree-based algorithm
07	PT-TA	Pi-Tree-based algorithm with TA pruning
08	BT	BitTable based algorithm
09	BTI	BitTable base algorithm with Inverted list index
10	BTIV	BitTable based algorithm with Inverted list index in Vertical
11	CSDL	Cơ sở dữ liệu

MỞ ĐẦU

Ngày nay dữ liệu được tạo ra từ mọi thiết bị, hơn 90% lượng dữ liệu trên thế giới được tạo ra trong hai năm gần đây. Có rất nhiều thiết bị và hệ thống tạo ra dữ liệu đặc biệt là các máy vi tính và thiết bị di động, các mạng xã hội. Dữ liệu này phản ánh mọi hoạt động của con người. Do đó trong những năm gần đây khai thác dữ liệu nổi lên như một ngành quan trọng của khoa học máy tính nhằm tìm kiếm các tri thức có giá trị trong khối dữ liệu khổng lồ đó, nhất là trong những lĩnh vực như xử lý số liệu, phân lớp dữ liệu, dự báo,…

Sự đa dạng và phong phú của dữ liệu hình thành nên nhiều mô hình dữ liệu khác nhau: mô hình dữ liệu giao tác (transaction), mô hình dữ liệu chuỗi (sequence), mô hình dữ liệu thời gian thực (real-time), …

Khai phá dữ liệu đang dần trở thành một ngành khoa học quan trọng trong đời sống của con người, giúp chúng ta có thể hiểu rỏ hơn quá khứ cũng như dự đoán tương lai. Đặc biệt trong kinh doanh, hiểu được hành vi của khách hàng là một lợi thế vô cùng lớn quyết định thành công của doanh nghiệp. Từ khi mới ra đời, bài toán phân tích các giao dịch của khách hàng để tìm ra các quy luật hành vi mua sắm của khách hàng đã trở thành bài toán tiêu biểu trong khai phá dữ liệu và đã có rất nhiều nghiên cứu hằng năm liên quan đến đề tài này. Tuy nhiên, không phải lúc nào chúng ta cũng quan tâm đến tất cả các quy luật liên kết trong toàn bộ cơ sở dữ liệu mà chỉ quan tâm đến những những hành vi liên quan đến một sự kiện nhất định. Từ đó, xuất hiện bài toán "*Tìm top-k sự kiện đồng xuất hiện*" được đề xuất vào năm 2015 bởi Zhi-Hong Deng. Ví dụ, khi người dùng chọn mua ba món hàng A,B và C, chúng ta sẽ cần quan tâm đến một số mặt hàng mà thường được mua cùng với A,B và C nhất, từ đó ta có thể đưa ra những gợi ý hữu ích cho khách hàng nhằm bán được nhiều mặt hàng hơn. Từ khi bài toán được đề xuất đến nay chưa có thêm công trình nghiên cứu mới nào được đề xuất để giảm thời gian xử lý bài toán. Do đó, đề tài "*Khai thác top-k sự kiện đồng xuất hiện với BitTable*" được đề xuất nhằm cải tiến thời gian xử lý bài toán.

Ngoài phần mở đầu, danh mục tài liệu tham khảo và phụ lục, bố cục của luận văn gồm có 05 chương:

Chương 1 giới thiệu sơ lược về đề tài thực hiện, lý do đề xuất đề tài và các hoàn cảnh thực tế có thể áp dụng đề tài đồng thời mô tả các công trình nghiên cứu đã được thực hiện trước đây, có liên quan đến đề tài thực hiện sẽ được thảo luận nhằm tìm kiếm giải pháp cho đề tài, các công trình này là nền tảng cho việc nghiên cứu nhằm đề xuất các cấu trúc dữ liệu và giải thuật mới. Chương này cũng nêu hướng tiếp cận nghiên cứu và đóng góp của đề tài.

Chương 2 trình bày cơ sở lý thuyết của bài toán khai thác top-k sự kiện đồng xuất hiện và các giai đoạn giải quyết bài toán. Khai thác top-k sự kiện đồng xuất hiện là một bài toán mới, được đề xuất từ năm 2015 bởi Zhi-Hong Deng. Tác giả đã đưa ra bốn thuật toán cơ bản và hai thuật toán dựa trên một cấu trúc dữ liệu đặc biệt có tên là Pi-Tree. Trong chương này chúng ta sẽ nhắc lại những định nghĩa, các thuật toán cơ bản (NT, NTI) và khái quát bài toán khai thác top-k sự kiện đồng xuất hiện.

Chương 3 trình bày chi tiết các thuật toán mà đề tài đề xuất, hai thuật toán sử dụng cấu trúc BitTable theo chiều ngang và thuật toán thứ ba dùng cấu trúc BitTable theo chiều dọc. Ba thuật toán lần lược là BT(BitTable based algorithm), BTI(BitTable base algorithm with Inverted list index) và $BTIV$(BitTable based algorithm with Inverted list index in Vertical). BT khai thác bằng cách chuyển cơ sở dữ liệu sang BitTable theo chiều ngang, sau đó quét tất cả các bit trong BitTable để tìm ra các item đồng xuất hiện. BTI khai thác tương tự như BT, nhưng trước khi chuyển cơ sở dữ liệu sang BitTable, BTI rút tỉa các giao tác (transaction) không mong đợi bằng kỹ thuật Tid-set. $BTIV$ khai thác bằng cách rút tỉa các giao tác không mong đợi sau đó chuyển cơ sở dữ liệu đã được xử lý thành BitTable dạng dọc, sau đó $BTIV$ sẽ đếm số bit được set bằng 1 để tìm ra các item đồng xuất hiện.

Chương 4 trình bày kết quả thực nghiệm khai thác trên các cơ sở dữ liệu giao dịch phổ biến, thường được dùng trong các nghiên cứu khai thác dữ liệu thực tế là Connect, Accidents và hai tập dữ liệu tổng hợp là Syn_data1 và Syn_data2. Từ kết

quả thực nghiệm chúng ta sẽ có cái nhìn chính xác hơn về hiệu quả và lượng bộ nhớ tiêu tốn của từng thuật toán.

Chương 5 trình bày kết luận và định hướng phát triển của đề tài. Cuối cùng, phần tham khảo trình bày các bài báo, sách được tham khảo, trích dẫn trong luận văn.

Như vậy đồ án sẽ được trình bày như sau:

- Chương 1: Tổng quan
- Chương 2: Cơ sở lý thuyết
- Chương 3: Khai thác top-k sự kiện đồng xuất hiện với BitTable
- Chương 4: Kết quả thực nghiệm
- Chương 5: Kết luận và hướng phát triển

CHƯƠNG 1. TỔNG QUAN

1.1. Giới thiệu

Khai phá dữ liệu (data mining) là quá trình tính toán để tìm ra các mẫu trong các bộ dữ liệu lớn liên quan đến các phương pháp tại giao điểm của máy học, thống kê và các hệ thống cơ sở dữ liệu. Đây là một lĩnh vực liên ngành của khoa học máy tính… Mục tiêu tổng thể của quá trình khai thác dữ liệu là trích xuất thông tin từ một bộ dữ liệu và chuyển nó thành một cấu trúc dễ hiểu để sử dụng tiếp. Ngoài bước phân tích thô, nó còn liên quan tới cơ sở dữ liệu và các khía cạnh quản lý dữ liệu, xử lý dữ liệu trước, suy xét mô hình và suy luận thống kê, các thước đo thú vị, các cân nhắc phức tạp, xuất kết quả về các cấu trúc được phát hiện, hiện hình hóa và cập nhật trực tuyến. Khai thác dữ liệu là bước phân tích của quá trình "khám phá kiến thức trong cơ sở dữ liệu".

1.2. Khai thác tập sự kiện phổ biến (frequent itemset mining)

Khai thác tập sự kiện phổ biến [1] (FIM) là một nhánh quan trọng trong lĩnh vực khai phá dữ liệu, khám phá những mẫu hữu ích hoặc thú vị trong cơ sở dữ liệu giao tác. Bài toán được giới thiệu đầu tiên vào năm 1993 bởi Agrawal và Srikant với tên gọi là khai phá tập sự kiện lớn (*large itemset mining*). Công việc của FIM có thể được định nghĩa như sau: cho một cơ sở dữ liệu giao dịch của khách hàng, FIM khám phá những nhóm mặt hàng mà khách hàng mua thường xuyên dựa vào một ngưỡng minsup cho trước. Ví dụ, người ta có thể phân tích một cơ sở dữ liệu giao dịch của khách hàng và khám phá ra rằng khách hàng thường mua mặt hàng taco shell với ớt. Khám phá sự liên hệ giữa những mặt hàng rất hữu ích để xác định hành vi của khách hàng.

Mặt dù FIM ban đầu được đề xuất là để phân tích dữ liệu khách hàng, nhưng bây giờ nó đã được xem như là công việc khai phá dữ liệu tiêu biểu mà được ứng dụng trong nhiều lĩnh vực khác nhau. Trên thực tế, cơ sở dữ liệu giao dịch khách hàng có thể được xem như là một cơ sở dữ liệu của các thể hiện các đối tượng (giao dịch), trong đó mỗi đối tượng được mô tả bằng cách sử dụng các giá trị thuộc tính danh

nghĩa (các mặt hàng). Do đó, FIM có thể được định nghĩa tương đương như là nhiệm vụ tìm các giá trị thuộc tính thường xảy ra đồng thời trong cơ sở dữ liệu. Vì nhiều loại dữ liệu có thể được biểu diễn dưới dạng cơ sở dữ liệu giao dịch, FIM có nhiều ứng dụng trong nhiều lĩnh vực như sinh học, phân loại hình ảnh, phân tích lưu lượng mạng, phân tích đánh giá của khách hàng, theo dõi hoạt động, phát hiện phần mềm độc hại và e-learning… **Bảng 1** bên dưới là ví dụ mô tả một cơ sở dữ liệu giao tác. Cơ sở dữ liệu này sẽ là đầu vào cho những ví dụ cũng như mô phỏng thuật toán của toàn bộ đề tài.

TID	Items
1	a, b, c
2	a, b, c, d, f
3	e, f, g
4	a, c, e, f
5	b, c, d, f

Bảng 1. Ví dụ cơ sở dữ liệu giao tác

Lĩnh vực khai phá tập sự kiện là một lĩnh vực nghiên cứu rất năng động và có hàng trăm thuật toán được đề xuất mỗi năm. Những thuật toán tiêu biểu cho bài toán FIM là những thuật toán như Apriori, Eclat, FP-Growth… Song song đó cũng có một số bài toán tương tự liên quan như khai thác top-k tập phổ biến hoặc khai thác top-k sự kiện đồng xuất hiện.

1.3. Khai thác top-k sự kiện đồng xuất hiện

Khai thác top-k sự kiện đồng xuất hiện [2] giống với khai thác tập sự kiện phổ biến ở chổ là nó tìm mối liên hệ giữa những sự kiện này với những sự kiện khác. Nhưng khác với khai thác tập sự kiện phổ biến FIM là, FIM sẽ tìm ra tất cả các luật liên kết giữa các sự kiện trong toàn bộ cơ sở dữ liệu, còn khai thác top-k sự kiện đồng xuất hiện chỉ tập trung khai thác các sự kiện liên quan đến một hay một tập sự kiện cụ thể nào đó. Do đó dữ liệu đầu vào của hai bài toán này cũng khác nhau. Trong một số trường hợp người ta không quan tâm đến việc tìm tất cả các mối liên hệ giữa các

sự kiện trong toàn bộ cơ sở dữ liệu mà họ chỉ quan tâm đến một sự kiện hoặc một nhóm sự kiện cụ thể nào đó. Ví dụ, khi phân tích các mặt hàng trong cơ sở dữ liệu giao dịch của khách hàng, người ta muốn biết là top 5 sản phẩm khách hàng thường mua chung với sữa và đường là gì, hoặc trong các mạng xã hội, người ta quan tâm top 2 sự kiện thường xảy ra với một sự kiện nào đó… Do đó đầu vào của 2 bài toán này sẽ khác nhau. FIM chỉ cần mội ngưỡng minsup cho trước, còn top-k sự kiện đồng xuất hiện thì cần một tập sự kiện (itemset) cần quan tâm và một số k là số sự kiện đồng xuất hiện cần tìm.

Bài toán khai thác top-k sự kiện đồng xuất hiện được đưa ra lần đầu tiên bởi Zhi-Hong Deng, trong một bài báo khoa học có tên là "*Mining Top-K Co-Occurrence Items*" [2] năm 2015. Trong nghiên cứu của mình, ông đã trình bày bốn thuật toán cơ bản và giới thiệu một cấu trúc dữ liệu đặt biệt có tên là Pi-Tree. Dựa trên Pi-Tree, tác giả đã đề xuất hai thuật toán tên là PT(Pi-Tree-based algorithm) và $PT-TA$(Pi-Tree-based algorithm with TA pruning). Bốn thuật toán cơ bản lần lượt có tên là NT(Naive Hunting algorithm), NTI (Naive Hunting algorithm withInverted list index), $NT\text{-}TA$ (NT with TA pruning), và $NTI\text{-}TA$(NTI with TA pruning)

1.3.1. Thuật toán NT và NTI

NT là thuật toán cơ bản nhất. NT sẽ quét tất cả các sự kiện trong cơ sở dữ liệu để tìm số lần đồng xuất hiện của từng sự kiện. Rỏ ràng việc làm này sẽ không hiệu quả khi mà NT phải quét quá nhiều giao tác không mong đợi (không chứa tập sự kiện đầu vào cần quan tâm). Để trách không phải xử lý những ứng viên không mong đợi như vậy, một bước cải tiến đó là thuật toán NTI. NTI sẽ loại bỏ tất cả các giao tác không mong đợi dựa vào một kỹ thuật gọi là Tid-set, chuyển toàn bộ cơ sở dữ liệu sang dạng dọc rồi giao tất cả các Tid-set lại để cho ra kết quả là một cơ sở dữ liệu chỉ bao gồm các giao tác chứa tập sự kiện cần khai thác. Do NTI xử lý trên cơ sở dữ liệu đã được rút tỉa nhỏ hơn nên tốc độ xử lý sẽ nhanh hơn.

1.3.2. Thật toán $NT\text{-}TA$ và $NTI\text{-}TA$

$NT\text{-}TA$ và $NTI\text{-}TA$ là hai thuật toán mở rộng tương ứng của NT và NTI bằng cách áp dụng nguyên lý TA(the threshold algorithm) của Fagin để có thể ngừng thuật

toán ngay khi tìm ra được kết quả mà không cần phải quét hết toàn bộ cơ sở dữ liệu. Khi đó, thuật toán sẽ có hai danh sách, một danh sách chứa những item kết quả (L_k) và một danh sách chứa những item ứng ứng viên (I_c). Giá trị đồng xuất hiện lớn nhất của một item bằng giá trị đồng xuất hiện tại thời điểm đang xét cộng với số giao tác còn lại (chưa được duyệt) trong cơ sở dữ liệu. Do đó, nếu giá trị đồng xuất hiện nhỏ nhất trong L_k lớn hơn giá trị đồng xuất hiện lớn nhất trong I_c cộng số giao tác còn lại chưa duyệt trong cơ sở dữ liệu thì có thể ngừng thuật toán mà không cần phải duyệt tiếp nữa.

1.3.3. Thuật toán *PT* và *PT-TA*

Lần đầu tiên một cấu trúc dữ liệu đặc biệt được giới thiệu đó là Pi-Tree, sử dụng một cấu trúc gọn nhẹ để chứa thông tin của một cơ sở dữ liệu. Một Pi-Tree là một cấu trúc cây tiền tố được định nghĩa như sau:

1. Cây được tạo thành từ một gốc có nhãn là "*Root*", một tập của những cây con của root, và một bảng header.
2. Mỗi node của cây chứa 4 trường dữ liệu: *label*, *count*, *children-link*, và *parent-link*. Trường *label* đăng ký item mà node thể hiện. Trường *count* đăng ký số lượng giao tác được trình bày bởi một phần đường dẫn đi qua node. Trường *children-link* đăng ký tất cả các node con của node. Trường *parent-link* đăng ký node cha của node.
3. Mỗi thành phần trong bảng header bao gồm hai trường: *label* và *node-link*. Trường *label* đăng ký cho một item. Trường *node-link* đăng ký tất cả các node thể hiện item đó. Trong bảng header, các thành phần được xếp thứ tự giảm dần bởi độ hổ trợ (support count). Bảng header được tạo ra để tạo thuận lợi cho việc duyệt cây Pi-Tree.

Theo định nghĩa trên Pi-Tree trông giống như FP-Tree. Tuy nhiên, có hai điểm khác biệt quan trọng giữa chúng. Đầu tiên, Pi-Tree không yêu cầu những item được đăng ký trong các node phải là những item phổ biến trong khi FP-Tree chỉ tập trung vào những item phổ biến. Thứ hai, những node của Pi-Tree có liên kết đến node cha

(*parent-link*) còn FP-Tree thì không. Cách xây dựng cây Pi-Tree và tìm top-k sự kiện đồng xuất hiện trên cây sẽ được trình bày cụ thể trong phần cơ sở lý thuyết.

1.4. Mục tiêu nghiên cứu của đồ án

Từ khi bài toán được giới thiệu năm 2015 đến nay, chưa có thêm một nghiên cứu nào đề xuất các phương pháp mới để giải quyết bài toán. Do đó, đồ án muốn đóng góp một phương pháp mới đó là dùng BitTable [3] để giải quyết bài toán cho ra kết quả xử lý nhanh hơn.

BitTable hay còn gọi là bảng bit, chứa các giá trị 0 hoặc 1. Mỗi bit 1 biểu thị cho sự xuất hiện của một sự kiện trong cơ sở dữ liệu. Trong vấn đề lưu trữ, BitTable giúp nén cơ sở dữ liệu xuống gấp nhiều lần giúp tiết kiệm không gian lưu trữ và khi xử lý trên mảng bit sử dụng được các phép toán trên bit khiến cho tốc độ xử lý trên máy tính nhanh hơn. Với ý tưởng đó mà tác giả đồ án quyết định thực hiện đề tài này "*Khai thác top-k sự kiện đồng xuất hiện với BitTable*".

Trong quá trình nghiên cứu, đồ án đề xuất ba thuật toán: BT(Bittable-based), BTI(Bittable-based with Inverted list index) và $BTIV$(Bittable-based with Inverted list index in Vertical). Hai thuật toán đầu BT và BTI có cách thức xử lý tương tự NT và NTI, sử dụng BitTable theo chiều ngang và dùng phép AND bit để xác định mẫu truy vấn P có chứa trong giao tác T hay không. Thuật toán cuối cùng $BTIV$. Khác với BT và BTI, $BTIV$ loại bỏ các giao tác không mong đợi trước, sau đó chuyển cơ sở dữ liệu đã được rút tỉa sang dạng BitTable theo chiều dọc để xử lý. Sau quá trình tiền xử lý, $BTIV$ chỉ cần đếm tổng số bit được set là 1 trên 1 dòng của bảng BitTable là ra kết quả tương ứng của một item.

Mục tiêu nghiên cứu của đồ án là đề xuất phương pháp mới để xử lý bài toán tìm top-k sự kiện đồng xuất hiện, đồng thời rút ra một số ưu điểm và khuyết điểm của các phương pháp trên.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1. Giới thiệu

Khai thác tập sự kiện phổ biến được đề xuất đầu tiên bởi Agrawal, nhằm phân tích các giao dịch của khách hàng để rút ra tất cả các luật liên kết giữa các mặt hàng. Kể từ khi việc khai thác dữ liệu này được đề xuất và những thuật toán khai thác hữu ích liên quan tới nó, đã có hàng nghìn công trình nghiên cứu theo sau trên nhiều lĩnh vực và ứng dụng. Khai thác tập phổ biến đã xuất hiện như một chủ đề quan trọng trong lĩnh vực khai thái dữ liệu. Nó đã được chứng minh đóng một vai trò thiết yếu trong nhiều công việc khai thác dữ liệu.

Trong khai thác tập phổ biến, một tập được gọi là phổ biến nếu tuần suất xuất hiện của nó trong cơ sở dữ liệu không nhỏ hơn một ngưỡng cho trước (minsup) . Có nghĩa là, tập sự kiện phổ biến là một khái niệm bao trùm trên phạm vi tất cả cơ sở dữ liệu và chúng không có bất kỳ liên quan gì đến những tập sự kiện khác. Tuy nhiên, trong một số ứng dụng như các hệ thống gợi ý hoặc mạng xã hội, người ta có lẻ sẽ quan tâm đến những sự kiện có liên quan với nhau nhiều hơn. Ví dụ trong hệ thống đưa ra gợi ý của một cửa hàng trực tuyến, bất cứ khi nào một người dùng thanh toán một vài sản phẩm, hệ thống phải đưa ra những gợi ý về những sản phẩm khác mà thường mua cùng với những sản phẩm đó nhằm mục đích tăng doanh số. Những việc đó có thể được minh họa bằng ví dụ dưới đây.

Ví dụ: hãy xét một cơ sở dữ liệu DB chứa những giao dịch mua hàng được thể hiện ở **Bảng 1** [Chương 1]. TID và Items là những thuộc tính thuộc miền sau $\{1,2,3,4,5\}$ và $\{a, b, c, d, e, f, g\}$. Mỗi giao tác T trong DB có một định danh duy nhất (TID) là một tập con của $\{a, b, c, d, e, f, g\}$. Để đơn giãn, chúng ta thể hiện một giao tác với $tid = j$ bởi T_j. Ví dụ T_2 là ký hiệu cho giao tác thứ hai trong Bảng 1. Top-2 những sự kiện đồng xuất hiện với tập sự kiện $\{ac\}$ là sự kiện f và b vì có hai giao tác mua hàng chứa $\{acf\}$ và hai giao tác mua hàng chứa $\{acb\}$ trong khi những sự kiện khác đồng xuất hiện với $\{ac\}$ hầu hết có một giao tác. Tương tự, top-2 sự kiện đồng xuất hiện với tập sự kiện $\{c\}$ là sự kiện a và f.

Mặc dù nhiều thuật toán đã được đề xuất để khai thác các tập sự kiện phổ biến và top-k tập sự kiện phổ biến, nhưng không khả thi để dễ dàng áp dụng những thuật toán đó để khai thác top-k sự kiện đồng xuất hiện. Thứ nhất, khai thác top-k sự kiện đồng xuất hiện không cần thiết lập một ngưỡng hỗ trợ nhỏ nhất như khai thác tập sự kiện phổ biến hay top-k tập sự kiện phổ biến, nơi mà một tập sự kiện phổ biến được định nghĩa như một tập có độ hỗ trợ không nhỏ hơn ngưỡng hỗ trợ nhỏ nhất cho trước. Vì thế, không thể áp dụng trực tiếp các thuật toán cho khai thác tập sự kiện phổ biến hay top-k tập sự kiện phổ biến vào khai thác top-k sự kiện đồng xuất hiện khi mà không khả thi để thiết lập một ngưỡng phù hợp để tìm top-k sự kiện đồng xuất hiện của mỗi tập sự kiện. Chú ý rằng, độ hỗ trợ tối thiểu của top-k sự kiện đồng xuất hiện của những tập sự kiện khác nhau thì vô cùng, chúng ta không thể nào biết trước được. Thứ hai, top-k sự kiện đồng xuất hiện không có tính chất "downward closure property", mọi tập con khác rỗng của tập phổ biến cũng là tập phổ biến và mọi tập chứa tập không phổ biến đều là tập không phổ biến, giống khai thác tập phổ biến.

Cho một itemset cần truy vấn, một hướng tiếp cận đơn giản để tìm top-k những sự kiện đồng xuất hiện là xem xét tất cả những sự kiện không liên quan đến nó trong cơ sở dữ liệu. Chắc chắn cách tiếp cận này sẽ gặp phải vấn đề không gian tiềm kiếm lớn, đặc biệt khi cơ sở dữ liệu rất lớn và một con số lớn của những tập sự kiện truy vấn cần được tìm kiếm top-k sự kiện đồng xuất hiện của chúng. Vì thế, để giảm không gian tiềm kiếm và tìm top-k những sự kiện đồng xuất hiện một cách hiệu quả là một thách thức mới.

2.2. Cơ sở lý thuyết

Cho $I = \{i_1, i_2, \dots, i_m\}$ là tập tất cả các sự kiện (item) và $DB = \{T_1, T_2, \dots, T_n\}$ là một cơ sở dữ liệu giao tác (transaction), với mỗi $T_k (1 \leq k \leq n)$ là một giao tác, là tập các sự kiện mà $T_k \subseteq I$. Chúng ta cũng có thể gọi P là một itemset nếu P là tập các sự kiện $P \subseteq I$.

2.2.1. Định nghĩa 1

Cho P là một itemset. Một giao tác T được gọi là chứa P nếu và chỉ nếu $\subseteq T$.

2.2.2. Định nghĩa 2

Độ hổ trợ của một itemset, được ký hiệu là $SC(P)$, là số giao tác chứa P trong DB.

2.2.3. Định nghĩa 3

Cho itemset, một sự kiện $i(i \in I \wedge i \notin P)$ là một sự kiện đồng xuất hiện của P nếu và chỉ nếu một hoặc nhiều giao tác chứa $P \cup \{i\}$. Tập tất cả những sự kiện đồng xuất hiện của P được ký hiệu là P_{coi}.

2.2.4. Định nghĩa 4

Cho itemset P và i $(\in P_{coi})$, số lần đồng xuất hiện của P và i , được ký hiệu là $CO(P, i)$, được định nghĩa là độ hổ trợ của $P \cup \{i\}$.

2.2.5. Định nghĩa 5 (Top-k sự kiện đồng xuất hiện)

Cho itemset P, một sự kiện i $(\in P_{coi})$ được gọi là Top-k sự kiện đồng xuất hiện của P nếu có ít hơn k sự kiện mà số lần đồng xuất hiện của chúng lớn hơn $CO(P, i)$.

2.3. Phát biểu bài toán

Cho một cơ sở dữ liệu giao tác DB, một itemset P, và số k mong muốn, bài toán tìm top-k sự kiện đồng xuất hiện của P là tìm k sự kiện mà xảy ra phổ biến nhất với P trong DB.

Giả định chúng ta muốn biết những sự kiện nào xảy ra phổ biến nhất với $\{a, c\}$ trong Bảng 1. Từ định nghĩa 4, ta biết rằng $CO(\{a, c\}, b) = 2$ bởi vì T_1 và T_2 chứa $\{abc\}$. Tương tự, ta có $(\{a, c\}, d) = 1$, $CO(\{a, c\}, e) = 1$, và $CO(\{a, c\}, f) = 2$. Nếu ngưỡng k được thiết lập là 1, thì top-1 sự kiện đồng xuất hiện của $\{a, c\}$ là b hoặc f. Nếu $= 2$, thì top-2 những sự kiện đồng xuất hiện của $\{a, c\}$ là b và f.

2.4. Thuật toán *NT*

2.4.1. Nội dung thuật toán

Cho một itemset P và cơ sở dữ liệu giao tác, NT quét tất cả các giao tác trong DB. Nếu một giao tác chứa P, NT tăng số lượng đồng xuất hiện của mỗi sự kiện i , không chứa trong P trong giao tác, lên 1. Bằng cách duyệt tất cả các giao tác, NT có được số lượng đồng xuất hiện của tất cả các sự kiện đồng xuất hiện của P. Bằng việc

sắp xếp giảm dần tất cả các sự kiện đồng xuất hiện theo số lượng đồng xuất hiện, NT tìm thấy tất cả top-k sự kiện đồng xuất hiện của P.

Thuật toán *NT*
Input: cơ sở dữ liệu giao tác DB, itemset P, và ngưỡng k.
Output: R_{TK}, Top-k sự kiện đồng xuất hiện của P
1: **foreach** $T \in DB$ **do**
2: **if** $P \subseteq T$ **then**
3: **foreach** $i \in T - P$ **do**
4: **if** i chưa có trong danh sách **then**
5: $CO(i) \leftarrow 1$;
6: **else**
7: $CO(i) \leftarrow CO(i) + 1$;
8: $R_{TK} \leftarrow \{x \mid CO(x)$ là một trong top-k phần tử lớn nhất của $\{CO(i)\}\}$;
9: **return** R_{TK};

Ví dụ chạy thuật toán trên cơ sở dữ liệu Bảng 1 với itemset $P = \{a, c\}$ và $k = 2$. NT duyệt tất cả các giao tác từ T_1 đến T_5. Những giao tác chứa a, c là T_1, T_2, T_4.

- Duyệt $T_1 = \{a, b, c\}$ ta có $CO(b) = 1$
- Duyệt $T_2 = \{a, b, c, d, f\}$ ta có $CO(b) = 2, CO(d) = 1, CO(f) = 1$
- Duyệt $T_4 = \{a, c, e, f\}$ ta có $CO(e) = 1, CO(f) = 2$

Kết quả cuối cùng ta có $CO(b) = 2, CO(f) = 2, CO(e) = 1, CO(d) = 1$

Lấy top-2 ta được kết quả của thuật toán là b và f.

2.4.2. Đánh giá

Ưu điểm của thuật toán này là đơn giản và dễ thực hiện. Nhưng nhược điểm của thuật toán là phải duyệt tất cả cơ sở dữ liệu nên thời gian xử lý chậm.

2.5. Thuật toán NTI

2.5.1. Nội dung thuật toán

Rỏ ràng, NT không hiệu quả khi nó quét tất cả giao tác trong cơ sở dữ liệu mà trong đó có nhiều giao tác không chứa itemset P dẫn đến nó phải quét những giao tác không mong đợi. Để loại bỏ việc duyệt những giao tác không mong đợi, đầu tiên NTI sẽ tìm tất cả các giao tác chứa itemset P bằng cách sử dụng Tid-set [4], một cấu trúc dữ liệu được tổ chức từ danh sách được đảo ngược.Tid-set của một itemset là tập TIDs của tất cả các giao tác có chứa itemset đó. Như đã phát biểu trong [4], Tid-set có thể được tính toán bằng cách giao tất cả các Tid-set của những sự kiện mà được chứa bởi itemset đó. Khi đó, NTI tìm ra top-k sự kiện đồng xuất hiện trong một cơ sở dữ liệu được hoạch định, là tập tất cả các giao tác chứa P, như NT đã làm. Bởi vì NTI chỉ tìm top-k sự kiện đồng xuất hiện trong một cơ sở dữ liệu nhỏ hơn, nên nó sẽ hiệu quả hơn NT.

Giải thuật của thuật toán NTI

Input: cơ sở dữ liệu giao tác DB, itemset P, và ngưỡng k.

Output: R_{TK}, top-k sự kiện đồng xuất hiện của P.

1: **foreach** $T \in DB$ **do**
2: **foreach** $i \in T$ **do**
3: **if** i is first visited **then**
4: TID_list(i) $\leftarrow \{T.tid\}$;
5: **else**
6: TID_list(i) $\leftarrow$ TID_list(i) $\cup \{T.tid\}$;
7: TID_list(P) $\leftarrow$ TID_list(P.*first-item*);
8: **foreach** $i \in (P \,/\, \{P$.first-item$\})$ **do**
9: TID_list(P) $\leftarrow$ TID_list(P) $\otimes$ TID_list(i);
10: $DB_p \leftarrow \{T | T \in DB \wedge T.tid \in$ TID_list(P)$\}$
11: chạy thuật toán NT nhưng thay DB bằng DB_p;

Ví dụ chạy thuật toán trên cơ sở dữ liệu Bảng 1 với itemset $P = \{a, c\}$ và $k = 2$. NTI xây dựng DB_P trước bằng cách xây dựng danh sách TID_list như sau:

- TID_list(a)= {1,2,4}
- TID_list(b)= {1,2,5}
- TID_list(c)= {1,2,4,5}
- TID_list(d)= {2,5}
- TID_list(e)= {3,4}
- TID_list(f)= {2,3,4,5}
- TID_list(g)= {3}

Do $P = \{a, c\}$ nên ta giao TID_list(a) với TID_list(c) ta được TID_list(P)= {1,2,4}. Kết quả ta được $DB_P = \{T_1, T_2, T_4\} = \{\{a, b, c\}, \{a, b, c, d, f\}, \{a, c, e, f\}\}$.

Chạy lại thuật toán NT với đầu vào là DB_P.

2.5.2. Đánh giá

Ưu điểm của thuật toán là đã loại bỏ những giao tác không mong đợi, do đó giúp thời gian xử lý nhanh hơn.

2.6. Thuật toán PT

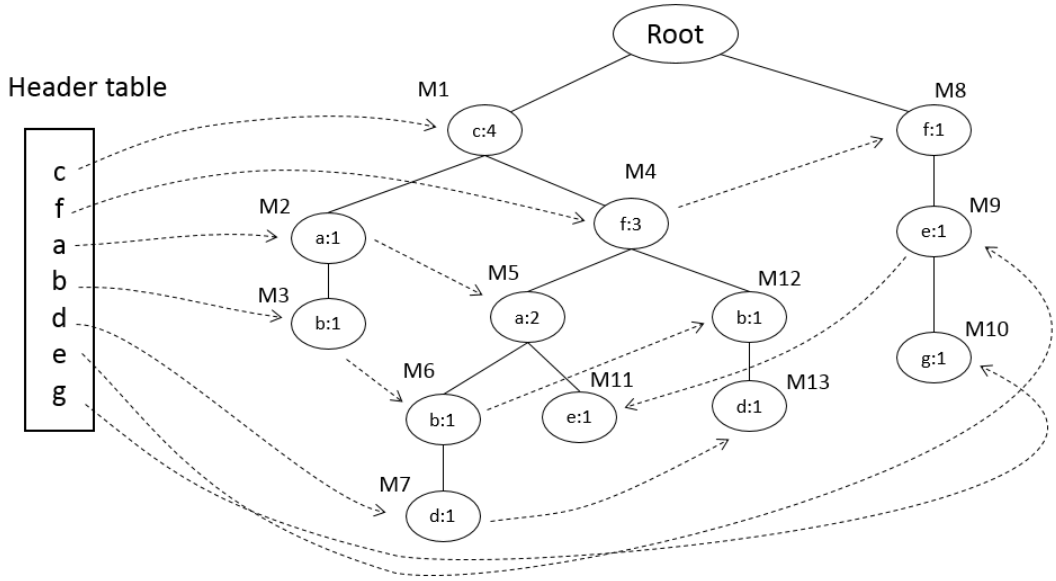
2.6.1. Cách xây dựng cây Pi-Tree

Cho một cơ sở dữ liệu, cây Pi-Tree có thể được xây dựng trong hai lần quét. Lần quét đầu tiên để tính độ hổ trợ của từng item. Lần quét thứ hai, một cây Pi-Tree với đỉnh góc đầu tiên gọi là Root được khởi tạo. Sau đó, các giao tác được chèn vào cây từng giao tác một. Trước khi chèn một giao tác vào cây, giao tác đó được sắp xếp thứ tự giảm dần theo độ hổ trợ của từng item, sau đó từng item sẽ được chèn vào cây theo thứ tự. Thuật toán sau trình bày cách xây dựng cây Pi-Tree:

Thuật toán xây dựng cây Pi-Tree
Input: Cơ sở dữ liệu DB **Output**: Cây Pi-Tree Ptr 1: quét toàn bộ DB để tìm tất cả các item và độ hổ trợ của nó; 2: khởi tạo Ptr với bảng chỉ mục; 3: **foreach** T in DB **do**

4:	sắp xếp T giảm dần theo độ hỗ trợ;
5:	$i \leftarrow T.first\text{-}item$; //$T.first\text{-}item$ là item đầu tiên trong giao tác T đã sắp xếp
6:	$T \leftarrow T - \{i\}$;
7:	gọi phương thức ***Insert_Tree***(i, T, Ptr);
8:	**return** Ptr;

Phương thức ***Insert_Tree***(i, T, Ptr) thực hiện như sau: Nếu Ptr có một đỉnh con N chẳn hạn như $N.label = i$, thì tăng $N.count$ lên 1. Ngược lại tạo một đỉnh N mới với giá trị *count* khởi tạo là 1, và thêm N vào danh sách *children-link* của Ptr và *node-link* trong bảng chỉ mục của item i tương ứng. Tiếp theo, nếu T vẫn còn chứa item, tiếp tục gọi lại phương thức ***Insert_Tree***$(T.fist\text{-}item,\ T - \{T.first\text{-}item\}, N)$. Hình bên dưới là cây Pi-Tree cho cơ sở dữ liệu Bảng 1 sau khi chạy thuật toán:



Hình 1. Cây Pi-Tree cho Bảng 1

2.6.2. Những định nghĩa và tính chất

Định nghĩa 1: Cho một cơ sở dữ liệu, chúng ta định nghĩa $>_{dsc}$ là một sắp xếp giảm dần theo độ hỗ trợ của tất cả các item. Cho hai item i_1 và i_2, $i_1 >_{dsc} i_2$ khi và chỉ khi i_1 đứng trước i_2.

Ví dụ đối với Bảng 1, ta có $>_{dsc}$ cho Bảng 1 là $\{c, f, a, b, d, e, g\}$. Bởi vì a đứng trước b nên ta có $a >_{dsc} b$. Lưu ý nếu hai item có cùng độ hổ trợ thì ta có thể sắp xếp chúng theo thứ tự alphabet.

Quy ước 1: Khi đề đề cập đến một itemset, có nghĩa là nó đã được sắp xếp theo $>_{dsc}$.

Tính chất 1: Cho i_1 và i_2 là hai item. Nd_1 và Nd_2 là hai đỉnh đăng ký i_1 và i_2 tương ứng, nếu $i_1 >_{dsc} i_2$ thì đỉnh Nd_2 không thể nào là đỉnh cha của Nd_1.

Tính chất 2: Nếu hai đỉnh Nd_1 và Nd_2 đăng ký hai item tương ứng i_1 và i_2. Nếu Nd_1 và Nd_2 nằm trên cùng đường dẫn và $i_1 >_{dsc} i_2$ thì đỉnh Nd_1 phải là đỉnh cha của Nd_2.

Tính chất 3: Một đường dẫn được gọi là *đường dẫn toàn phần* là một đường đi từ đỉnh lá đến *root.*

Tính chất 4: Cho một itemset P và một đường dẫn Pa, Pa được gọi là đăng ký P nếu lấy i bất kỳ chứa trong P, Pa cũng có một đỉnh tương ứng đăng ký i.

Tính chất 5: Cho itemset P và một item $i(\in P_{coi})$, $SFP(P, i)$ được định nghĩa là tập tất cả những đường dẫn toàn phần đăng ký $P \cup \{i\}$.

Tính chất 6: Tập các đỉnh đăng ký item i được ký hiệu là $NR(i)$.

Phép luận 1: Cho P là một itemset và i là một item trong tập P_{coi}. Item cuối cùng của P được ký hiệu là i_{LoP}. Ta có những kết luận sau:

1. Nếu $i_{LoP} >_{dsc} i, CO(P, i)$ bằng tổng *count* của tất cả các đỉnh đăng ký i mà nằm trên đường dẫn toàn phần thuộc $SFP(P, i)$. $CO(P, i) = \sum_{(N \in NR(i) \wedge (\exists Pa \in SFP(P,i), N \in Pa.Nset))} N.count$ với $Pa.Nset$ là tập tất cả các đỉnh trong đường dẫn Pa.
2. Nếu $i >_{dsc} i_{LoP}, CO(P, i)$ bằng tổng count của tất cả các đỉnh đăng ký i_{LoP} và nằm trên đường dẫn toàn phần thuộc $SFP(P, i)$. $CO(P, i) = \sum_{(N \in NR(i_{LoP}) \wedge (\exists Pa \in SFP(P,i), N \in Pa.Nset))} N.count$

2.6.3. Thuật toán PT

Mấu chốt của thuật toán là tìm tất cả đường dẫn toàn phần *SFP(P,i)* và item cuối của *P,* i_{LoP}. Ví dụ cho itemset $P = \{c, a\}$ ta có ba đường dẫn chứa c, a là $M1M2M3$, $M1M4M5M6M7$ và $M1M4M5M11$ và $i_{LoP} = a$. Khi đó ta có tập các item đồng xuất hiện của P là $P_{coi} = \{f, b, d, e\}$. Giả xử xét $i = b$ ta thấy có hai đường dẫn toàn phần chứa những đỉnh đăng ký b là $M1M2M3$, $M1M4M5M6M7$ và b nằm trong trường hợp đầu tiên $i_{LoP} >_{dsc} i$. Theo kết luận số 1 ta có $CO(P, b) = M3.count + M6.count = 1 + 1 = 2$. Lưu ý $NR(b) = \{M3, M6, M12\}$ nhưng chỉ tính tổng *count* của $M3$ và $M6$ vì $M12$ không nằm trong $SFP(P, b)$. Xét một trường hợp khác là $i = f$, đường dẫn toàn phần $SFP(P, f)$={ $M1M4M5M6M7$, $M1M4M5M11$ }. Trong trường hợp này $i >_{dsc} i_{LoP}$. Do đó ta xét $NR(i_{LoP}) = \{M2, M5\}$. Do $M2$ không thuộc bất kỳ đường dẫn nào thuộc $SFP(P, f)$ nên $CO(P, f) = M5.count = 2$. Tương tự $CO(P, d) = 1, CO(P, e) = 1$. Tổng kết ta có top-2 sự kiện đồng xuất hiện của $\{c, a\}$ là f và b.

Tính chất 7: Cho i_1, i_2 và i_3 là ba item với $i_1 >_{dsc} i_2 >_{dsc} i_3$. Giả sử đỉnh N_1 và N_3 đăng ký i_1 và i_3 tương ứng và tồn tại một đường dẫn ký hiệu là $Path_{31}$ từ N_1 đến N_3. Cho $Path_{30}$ là đường dẫn từ N_3 đến *Root*. Nếu không có đường dẫn $Path_{31}$ đăng ký i_2 thì không có đỉnh nào trên $Path_{30}$ có thể đăng ký i_2.

Thuật toán PT
Input: Một itemset $P = i_1, i_2, \dots, i_s$ và một cây Pi-Tree Ptr **Output**: L_k, tập top-k sự kiện đồng xuất hiện của P 1: Tìm CNS, tập tất cả các đỉnh đăng ký i_s, bằng cách tra chỉ mục của i_s trong bảng chỉ mục của Ptr 2: $NS \leftarrow \emptyset$; 3: **foreach** N in CNS **do** 4: $x \leftarrow s - 1$; 5: $flag \leftarrow$ false; 6: $Current\_node \leftarrow N.father$; 7: **do** 8: **if** $i_x = Current\_node.label$ **then**

9: $x \leftarrow x - 1$;
10: $Current\_node \leftarrow Current\_node.father$;
11: **else**
12: **if** $i_x >_{dsc} Current\_node.label$ **then**
13: $Current\_node \leftarrow Current\_node.father$;
14: **else**
15: $flag \leftarrow true$;
16: **until** $(x = 0) \vee (flag =$ true)
17: **if** $flag =$ false **then**
18: $NS \leftarrow NS \cup \{N\}$;
19: **foreach** N in NS **do**
20: Quét đường dẫn từ N đến *Root*. Đối với mỗi đỉnh Nd trong đường dẫn, nếu *label* của Nd không thuộc P, tăng $CO(P, Nd.label)$ lên $N.count$;
21: Duyệt cây con có đỉnh gốc tại N, đối với mỗi đỉnh Nd không phải N, tăng $CO(P, Nd.label)$ lên $Nd.count$;
22: Sắp xếp tất cả CO(P,i) theo thứ tự giảm dần và gán top-k đồng xuất hiện cho L_k;
23: **return** L_k;

Trong thuật toán, biến $flag$ được dùng để kiểm tra điều kiện của tính chất 7. Nếu $flag$ bằng **true** thì không cần xử lý đường dẫn đó nữa. Nếu $flag$ không thay đổi, bằng **false**, sau khi tất cả các đỉnh được duyệt thì có nghĩa là đường dẫn đó đăng ký P và đỉnh N đang xét là đỉnh cần tìm để tính top-k sự kiện đồng xuất hiện của P. Lưu ý là có rất nhiều đỉnh đăng ký i_s nhưng không phải tất cả đều dùng để tìm top-k. CNS là tập tất cả các đỉnh đăng ký i_s, còn NS là tập các đỉnh đăng ký i_s dùng để tìm to-k sự kiện đồng xuất hiện với P.

2.6.4. Đánh giá

Thuật toán đã vận dụng sáng tạo cây FP-Tree trong khai thác tập sự kiện phổ biến để xây dựng cây Pi-Tree. Bảng chỉ mục của Pi-Tree có lưu lại danh sách các đỉnh của từng chỉ mục giúp duyệt cây nhanh hơn. Sử dụng cây Pi-Tree thuật toán không phải duyệt toàn bộ cơ sở dữ liệu để tính toán số lần đồng xuất hiện của các

item giúp thời gian xử lý nhanh hơn. Tuy nhiên quá trình xây dựng cây Pi-Tree tốn nhiều thời gian và bộ nhớ. Thuật toán còn phức tạp và khó cài đặt.

2.7. Cấu trúc BitTable

2.7.1. Giới thiệu

Cấu trúc BitTable là cấu trúc dữ liệu đặc biệt được giới thiệu lần đầu tiên trong một công trình khoa học có tên **"BitTableFI: An efficient mining frequent itemsets algorithm"** [3] của hai tác giả Jie Dong và Min Han, năm 2006.

Bittable là cấu trúc dữ liệu dạng bảng mà mỗi bit biểu diễn một sự kiện có xảy ra hay không. Nếu giá trị là 1 có nghĩa là sự kiện có xảy ra. Ta có thể biểu diễn BitTable theo hai dạng, biểu diễn theo chiều ngang và biểu diễn theo chiều dọc. Gọi cơ sở dữ liệu giao tác là DB ta ký hiệu bảng BitTable theo chiều ngang là BTh và bảng BitTable theo chiều dọc là BTv. Ví dụ ta có thể chuyển Bảng 1 từ cơ sở dữ liệu giao tác thành cấu trúc bittable như sau:

	a	**b**	**c**	**d**	**e**	**f**	**g**
1	*1*	*1*	*1*	*0*	*0*	*0*	*0*
2	*1*	*1*	*1*	*1*	*0*	*1*	*0*
3	*0*	*0*	*0*	*0*	*1*	*1*	*1*
4	*1*	*0*	*1*	*0*	*0*	*1*	*0*
5	*0*	*1*	*1*	*1*	*0*	*1*	*0*

Bảng 2. Thể hiện BitTable theo chiều ngang của Bảng 1

Đối với bảng BitTable theo chiều ngang, mỗi cột tương ứng với một sự kiện, mỗi dòng tương ứng với một giao tác. Khi đó, nếu ta có một cơ sở dữ liệu giao tác với n sự kiện và m giao tác thì ta sẽ có một bảng BitTable với kích thước $n \times m$. Bảng BitTable theo chiều dọc cũng tương tự nhưng đổi cột thành dòng.

	1	**2**	**3**	**4**	**5**
a	*1*	*1*	*0*	*1*	*0*
b	*1*	*1*	*0*	*0*	*1*
c	*1*	*1*	*0*	*1*	*1*

d	*0*	*1*	*0*	*0*	*1*
e	*0*	*0*	*1*	*0*	*0*
f	*0*	*1*	*1*	*1*	*1*
g	*0*	*0*	*1*	*0*	*0*

Bảng 3. Thể hiện BitTable theo chiều dọc của Bảng 1

2.7.2. Những công trình nghiên cứu liên quan đến BitTable

Công trình nghiên cứu đầu tiên sử dụng BitTable đó là "***BitTableFI: An efficient mining frequent itemsets algorithm***" của hai tác giả Jie Dong và Min Han, năm 2006. Lần đâu tiên cấu trúc BitTable được đề xuất nhằm giải quyết các vấn để của thuật toán Apriori và những thuật toán tương tự khai thác tập sự kiện phổ biết. Trong nghiên cứu này, một cấu trúc dữ liệu đặc biệt được đề xuất, BitTable, để nén cơ sở dữ liệu và lưu trữ những tập sự kiện và một thuật toán, BitTableFI, để khai thác những tập sự kiện phổ biến từ những cơ sở dữ liệu lớn. BitTableFI có nhiều ưu điểm hơn những thuật toán khác. Thứ nhất thuật toán sử dụng toán tử AND/Or trên bit để tạo ra các tập ứng viên được nén trên BitTable. Các toán tử And/Or xử lý nhanh hơn phương pháp so sánh các item truyền thống được sử dụng trong những thuật toán giống Apriori. Thứ hai, thuật toán xây dựng một cấu trúc dữ liệu gọn nhẹ BitTable, cấu trúc này thường nhỏ hơn đáng kể so với cơ sở dữ liệu nguyên thủy và thuật toán có thể tạo ra các tập sự kiện ứng viên một cách nhanh chống và hổ trợ đếm hổ trợ của các tập ứng viên trực tiếp trên BitTable sử dụng toán tử AND. Việc này giúp tiết kiệm chi phí quét cơ sở dữ liệu trong quá trình khai thác các mẫu con tuần tự. Thứ ba, kỹ thuật phát sinh những tập ứng viên và đếm độ hổ trợ một cách nhanh chống có thể được áp dụng một cách dễ dàng vào những thuật toán tương tự Apriori. Tuy nhiên thuật toán chỉ tập trung giải quyết hai vấn đề phát sinh tập ứng viên và đếm độ hổ trợ, nên không giải quyết vấn đề cắt tỉa giảm bớt các ứng viên không mong đợi cũng như thời gian quét cơ sở dữ liệu giao tác.

Công trình nghiên cứu thứ hai là "***Index-BitTableFI: An improved algorithm for mining frequent itemsets***" [4] do nhóm tác giả Wei Song, Bingru Yang,

Zhangyan Xu. Trong nghiên cứu này, một thuật toán được đề xuất là Index-BitTableFI. Tương tự BitTableFI, thuật toán cũng sử dụng cấu trúc BitTable để giải quyết bài toán khai thác tập sự kiện phổ biến. Thuật toán BitTableFI đạt được hiệu suất xử lý tốt khi giảm được chi phí phát sinh tập ứng viên và tính toán độ hổ trợ. Tuy nhiên trong một số trường hợp với số lượng tập sự kiện phổ biến lớn, tập sự kiện dài hoặc ngưỡng độ hổ trợ khá thấp, thì nó tốn chi phí để quản lý một lượng lớn tập ứng viên. Để giải quyết vấn đề đó, thuật toán Index-BitTableFI được đề xuất. Thuật toán đã đề xuất mãng chỉ mục và phương pháp tính toán tương ứng bằng cách tính toán tập hợp chỉ mục, những tập sự kiện mà đồng xuất hiện với sự kiện tiêu biểu có thể được xác định nhanh chống.

Công trình nghiên cứu thứ ba là "***Dynamic bit vectors: An efficient approach for mining frequent itemsets***" [5] do nhóm tác giả Bay Vo, Tzung-Pei Hong và Bac Le đề xuất năm 2011. Công trình đã đề xuất một phương pháp mới cho khai thác tập sự kiện phổ biến từ cơ sở dữ liệu giao tác dựa trên sơ đồ vector bit động, giảm được thời gian xử lý cũng như bộ nhớ sử dụng.

Công trình nghiên cứu thứ tư là "***CBT-fi: Compact BitTable Approach for Mining Frequent Itemsets***" [6] do nhóm tác giả A.Saleem Raja và E.George Dharma Prakash Raj, năm 2014.

2.7.3. Ưu điểm và khuyết điểm của BitTable

Ưu điểm của BitTable là ta có thể dùng các toán tử bit And/Or trong quá trình tính toán cho ra thời gian xử lý nhanh hơn. Ngoài ra ta cũng có thể nén cơ sở dữ liệu khi chuyển các ký hiệu bit thành dãy số thập phân.

Nhược điểm của BitTable là kích thước của BitTable thì cố định tùy vào số lượng giao tác và số sự kiện (items). Do đó trong nhiều trường hợp khi chuyển cơ sở dữ liệu nguyên thủy sang BitTable có nhiều bit trống (giá trị bằng 0) dẫn đến thời gian xử lý và tiêu tốn bộ nhớ không cần thiết.

CHƯƠNG 3. KHAI THÁC TOP-K SỰ KIỆN ĐỒNG XUẤT HIỆN VỚI BITTABLE

3.1. Ý tưởng đề xuất

Nhiều thuật toán đã sử dụng BitTable để giải quyết bài toán và cho ra kết quả khá khả quan. Tuy nhiên tất cả những thuật toán trên mới chỉ áp dụng cho bài toán khai thác tập sự kiện phổ biến và chưa có thuật toán nào sử dụng BitTable để giải quyết bài toán tìm top-k sự kiện đồng xuất hiện. Đó chính là lý do đồ án nêu ra ý tưởng sử dụng BitTable để tìm top-k sự kiện đồng xuất.

3.2. Thuật toán *BT*

Ý tưởng của thuật toán BT có nhiều điểm tương tự như NT, điểm khác biệt ở chổ là NT xử lý trên cơ sở dữ liệu giao tác dạng văn bảng, còn BT xử lý trên bảng BitTable dạng ngang. BT cũng duyệt từng dòng của bảng BitTable để đếm số lần đồng xuất hiện của tất cả các sự kiện. Ví dụ đầu vào của NT là Bảng 1 thì của BT là Bảng 2.

Do mỗi dòng là một giao tác dạng bit, nên để kiểm tra itemset P có được chứa trong giao tác T hay không, BT chỉ cần thực hiện phép AND để kiểm tra. Ví dụ, xét bảng BitTable Bảng 2, để kiểm tra itemset $P = \{a, c\}$ có chứa trong giao tác T_2 hay không ta chỉ cần kiểm tra bit tại $(a: 2) = 1$ và $(c: 2) = 1$ bằng cách đơn giản là thực hiện phép AND giữa chuỗi bit *1110010* với *1010000* nếu kết quả bằng với *1010000* thì kết luận P chứa trong T_2. Tổng quát ta có $T_{bit}\ AND\ P_{bit} = P_{bit} \Leftrightarrow P \subseteq T$.

Thuật toán *BT*
Input: cơ sở dữ liệu giao tác DB, itemset P, và ngưỡng k.
Output: R_{TK}, Top-k sự kiện đồng xuất hiện của P
1: Chuyển DB thành BTh
2: Tạo P_{bit} từ P
3: **foreach** $T_{bit} \in BTh$ **do**
4: **if** $P_{bit}\ AND\ T_{bit} == P_{bit}$ **then**

5: **foreach** $i \in T_{bit}$ **do**

6: **if** $i == 1$ **then**

7: $CO(item) \leftarrow getItem(i.index)$;

8: **if** $CO(item)$ is $NULL$ **then**

9: $CO(item) \leftarrow 1$;

10: **else**

11: $CO(item) \leftarrow CO(item) + 1$;

12: $R_{TK} \leftarrow \{x | CO(x)$ là một trong top-k phần tử lớn nhất của $\{CO(item)\}\}$;

13: **return** R_{TK};

3.3. Thuật toán *BTI*

Rỏ ràng, BT không hiệu quả khi thuật toán quét tất cả các dòng giao tác không mong đợi (không chứa itemset P). Để loại trừ điều đó, BTI sử dụng Tid-set [7] tương tự như NTI để xây dựng một cơ sở dữ liệu chỉ bao gồm những giao tác có chứa itemset P. BTI sẽ xây dựng DB_P trước, sau đó sẽ chuyển DB_P thành bảng BitTable theo chiều ngang để xử lý.

Giải thuật của thuật toán *BTI*

Input: cơ sở dữ liệu giao tác DB, itemset P, và ngưỡng k.

Output: R_{TK}, top-k sự kiện đồng xuất hiện của P.

1: **foreach** $T \in DB$ **do**

2: **foreach** $i \in T$ **do**

3: **if** i is first visited **then**

4: TID_list(i) $\leftarrow \{T.tid\}$;

5: **else**

6: TID_list(i) $\leftarrow$ TID_list(i) $\cup \{T.tid\}$;

7: TID_list(P) $\leftarrow$ TID_list(P.*first-item*);

8: **foreach** $i \in (P \,/\, \{P$.first-item$\})$ **do**

9: TID_list(P) $\leftarrow$ TID_list(P) $\otimes$ TID_list(i);

10: $DB_p \leftarrow \{T | T \in DB \wedge T.tid \in$ TID_list$(P)\}$

11: chạy thuật toán BT nhưng thay DB bằng DB_p;

3.4. Thuật toán *BTIV*

$BTIV$ xử lý dựa vào bảng BitTable theo chiều dọc thay vì theo chiều ngang giống BT và BTI. Đồng thời $BTIV$ cũng loại bỏ những giao tác không chứa P trước khi xử lý. Ví dụ, đối với cơ sở dữ liệu Bảng 1, cho itemset $P = \{a, c\}$. $BTIV$ sẽ loại bỏ những giao tác không chứa P trước, khi đó ta sẽ có DB_P như bảng sau:

TID	Items
1	a, b, c
2	a, b, c, d, f
4	a, c, e, f

Bảng 4. Cơ sở dữ liệu sau khi loại bỏ các giao tác không chứa a, c

Sau đó $BTIV$ sẽ chuyển Bảng dữ liệu trên thành BitTable dạng dọc

	1	**2**	**4**
a	*1*	*1*	*1*
b	*1*	*1*	*0*
c	*1*	*1*	*1*
d	*0*	*1*	*0*
e	*0*	*0*	*1*
f	*0*	*1*	*1*

Bảng 5. Bảng BitTable dạng dọc của tất cả những giao tác đều chứa a, c

Tiếp theo $BTIV$ loại bỏ những dòng dữ liệu của những item chứa trong itemset P là a và c

	1	**2**	**4**
b	*1*	*1*	*0*
d	*0*	*1*	*0*

e	*0*	*0*	*1*
f	*0*	*1*	*1*

Bảng 6. Bảng BitTable mà $BTIV$ sẽ xử lý

Để đếm số lần đồng xuất hiện của mỗi sự kiện, $BTIV$ chỉ việc đếm số bit được set là 1 trong mãng bit tương ứng mà thôi. Rỏ ràng việc đếm số lượng bit 1 trong một mãng bit sẽ đơn giản và nhanh hơn rất nhiều so với việc duyệt từng bit để đếm số lượng đồng xuất hiện của mỗi sự kiện.

	1	**2**	**4**	
b	*1*	*1*	*0*	$\rightarrow CO(P,b) = 2$
d	*0*	*1*	*0*	$\rightarrow CO(P,d) = 1$
e	*0*	*0*	*1*	$\rightarrow CO(P,e) = 1$
f	*0*	*1*	*1*	$\rightarrow CO(P,f) = 2$

Hình 2. Minh họa kết quả đếm bit để ra kết quả của thuật toán $BTIV$

Thuật toán *BTIV*

Input: cơ sở dữ liệu giao tác DB, itemset P, và ngưỡng k.

Output: R_{TK}, top-k sự kiện đồng xuất hiện của P.

1: Xây dựng DB_P từ DB;

2: Chuyển DB_P thành bảng BitTable theo chiều dọc BTv;

3: **foreach** $i \in P$ **do**

4: loại bỏ $i \in BTv$;

5: **foreach** $i \in BTv$ **do**

6: $CO(i) \leftarrow countBitArray(i.bitArray)$;

7: $R_{TK} \leftarrow \{x | CO(x)$ is one of top-k biggest elements of $\{CO(i)\}\}$;

8: **return** R_{TK};

So với BT và BTI, BTIV chỉ tốn một vòng lặp để ra được kết quả, so với hai vòng lặp nhự BT và BTI thì tốc độ của BTIV sẽ nhanh hơn rất nhiều.

3.5. Ưu điểm và khuyết điểm

Ưu điểm của *BTIV* là thuật toán đơn giản, dễ cài đặt và cho kết quả xử lý nhanh. Nhược điểm là bộ nhớ sử dụng tăng theo tùy vào cơ sở dữ liệu đầu vào. Do đó cơ sở dữ liệu càng lớn thì bộ nhớ cần để xử lý trong quá trình tiền xử lý cũng càng lớn.

CHƯƠNG 4. KẾT QUẢ THỰC NGHIỆM

Nội dung Chương 4 trình bày kết quả thực nghiệm của các thuật toán mà đồ án đề xuất, đồng thời so sánh kết quả với thuật toán NT, NTI và PT của Zhi-Hong Deng. Đồ án không so sánh kết quả thực nghiệm với các thuật toán mở rộng của NT, NTI và PT là $NT\text{-}TA$, $NTI\text{-}TA$ và $PT\text{-}TA$, vì căn cứ trên kết luận của tác giả Zhi-Hong Deng [2], các thuật toán mở rộng không cho kết quả tốt hơn thuật toán nguyên thủy của nó, cho dù đã lấy giới hạn ngưỡng để không phải duyệt hết tất cả các ứng viên. Lý do tác giả nêu ra là vì hai nguyên nhân. Thứ nhất, việc cập nhật thường xuyên giá trị của giới hạn trên và giới hạn dưới tốn nhiều thời gian. Thứ hai, giới hạn trên và giới hạn dưới được sử dụng bởi những thuật toán mở rộng khá gần nhau nên không đủ giúp thuật toán kết thúc sớm như mong đợi. Tất cả mã lệnh được viết bằng ngôn ngữ C#, chạy trên hệ điều hành Win8 64bit, bộ vi xử lý Intel Core i-4200U 1.6 GHz.

4.1. Tập dữ liệu

Đồ án tiến hành thực nghiệm trên hai cơ sở dữ liệu thực tế và hai cơ sở dữ liệu tổng hợp. Bảng 7 thể hiện những thông số của các cơ sở dữ liệu. Cột cuối cùng của Bảng 7, chúng ta chia số lượng giao tác cho số lượng sự kiện để đánh giá tỷ trọng của cơ sở dữ liệu. $Tỷ\ trọng = \frac{Số\ lượng\ giao\ tác}{Số\ lượng\ sự\ kiện}$. Tỷ trọng càng lớn có nghĩa là cơ sở dữ liệu càng dày đặc.

Connect và Accidents là hai cơ sở dữ liệu thực tế và có thể được tải về tại địa chỉ web http://fimi.ua.ac.be/data . Chúng thường được sử dụng trong những công trình nghiên cứu trước về khai thác tập sự kiện phổ biến. Trong đồ án này, Connect và Accidents được dùng để chạy thực nghiệm là vì để so sánh với kết quả thực nghiệm của đồ án với kết quả thực nghiệm của tác giả Zhi-Hong Deng [2]. Để kiểm tra các thuật toán trên cơ sở dữ liệu lớn, đồ án tạo hai cơ sở dữ liệu tổng hợp bằng chương trình SPMF, một chương trình mã nguồn mở viết bằng Java, và có thể tải về tại địa chỉ web http://www.philippe-fournier-viger.com/spmf . Đồ án cũng muốn tạo ra hai cơ sở dữ liệu tổng hợp tương tự như trong bài báo gốc của tác giả Zhi-Hong Deng,

nhưng do dữ liệu được phát sinh ngẫu nhiên, nên không thể so sánh với dữ liệu đã có của bài báo gốc [2] vốn không được công bố. Cơ sở dữ liệu tổng hợp được đặc tên là Syn_data1 và Syn_data2 được sử dụng trong thực nghiệm. Để tạo ra Syn_data1, kích thước tối đa mỗi giao tác, số giao tác, số sự kiện khác nhau được thiết lập lần lượt là 20, 1000000 và 198. Những tham số để tạo Syn_data2 cũng tương tự nhưng số sự kiện khác nhau thì được thiết lập là 678, để làm cho Syn_data2 thưa thớt hơn Syn_data1.

Bên dưới đây, đồ án sẽ trình bày và phân tích các kết quả thực nghiệm trên nhiều phương diện khác nhau bao gồm thời gian tiền xữ lý, bộ nhớ sử dụng và thời gian xữ lý.

CSDL	Số Lượng Giao Tác	Số Sự Kiện Khác Nhau	Chiều Dài Trung Bình Mỗi Giao Tác	Tỉ Trọng
Connect	67,557	129	43	524
Accidents	340,183	468	34	727
Syn_data1	1,000,000	198	20	5,050
Syn_data2	1,000,000	678	20	1,475

Bảng 7. Đặc điểm của các cơ sở dữ liệu sử dụng trong thực nghiệm

4.2. Phương pháp thực nghiệm

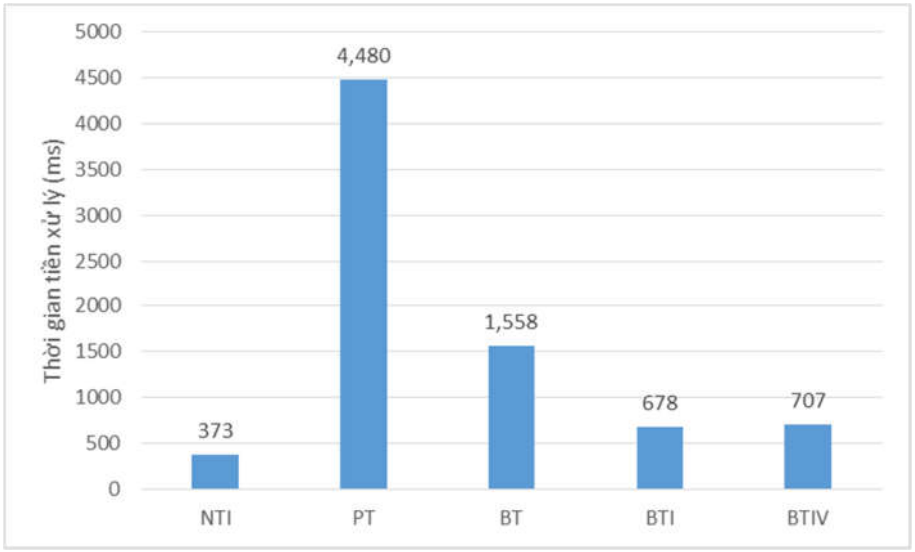
Thực nghiệm sẽ chạy lần lượt trên 4 cơ sở dữ liệu: Connect, Accidents, Syn_data1 và Syn_data2. Ba thuật toán gốc của tác giả NT,NTI,PT và ba thuật toán mới được đề xuất BT,BTI và $BTIV$ sẽ được chạy lần lượt trên mỗi cơ sở dữ liệu. Theo kết luận của tác giả bài báo gốc [2] thì các thuật toán mở rộng TA không đạt hiệu quả nên đồ án không chạy thực nghiệm các thuật toán đó. Mỗi thuật toán sẽ chạy lần lược với thông số $k = \{1,5,10,15\}$ và chiều dài itemset truy vấn P có độ dài $\{3,4,5,6,7\}$. Các mẫu itemset P đầu vào cho các thuật toàn là giống nhau. Mỗi cặp k và chiều dài itemset cho chạy 100 lần với 100 itemset P khác nhau.

4.3. So sánh về thời gian tiền xử lý

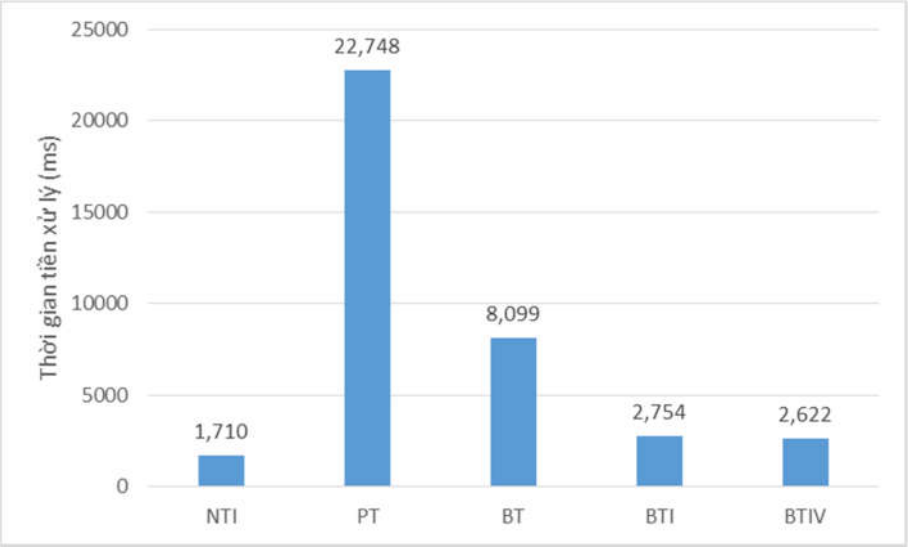
Thời gian tiền xử lý là thời gian xây dựng cấu trúc dữ liệu phục vụ cho quá trình xử lý của thuật toán. Ví dụ, đối với thuật toán NTI, thời gian tiền xử lý là thời gian rút tỉa cơ sở dữ liệu đầu vào để có được một cơ sở dữ liệu nhỏ gọn hơn và thuật toán sẽ xử lý trên cơ sở dữ liệu đó. Đối với thuật toán PT, thì thời gian tiền xử lý là thời gian xây dựng được cây Pi-Tree. Đối với các thuật toán BT,BTI và $BTIV$, thời gian tiền xử lý là thời gian để cho ra được bảng BitTable. Thuật toán NT không có giai đoạn tiền xử lý nên đồ án không so sánh thuật toán NT trong phần này.

Thời gian tiền xử lý của 5 thuật toán: NTI,PT và ba thuật toán đề xuất được thể hiện bằng đồ thị trong Hình 3, Hình 4, Hình 5 và Hình 6.

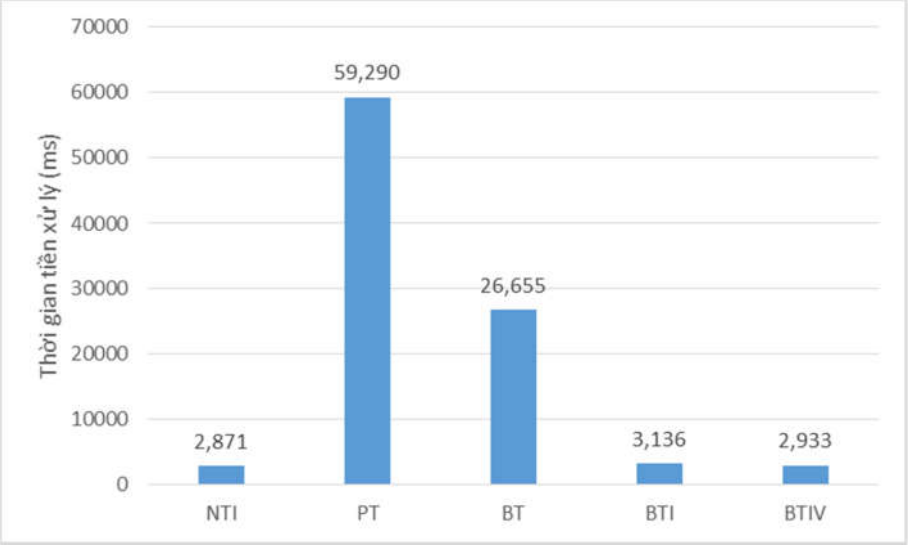
Từ 4 đồ thị trong Hình 3, Hình 4, Hình 5 và Hình 6, thấy được rằng: Thời gian tiền xử lý càng tăng khi tập dữ liệu đầu vào càng lớn. Thuật toán BT có thời gian tiền xử lý lâu hơn BTI và $BTIV$ là vì thuật toán BT không rút tỉa cơ sở dữ liệu đầu vào trước khi chuyển qua BitTable.



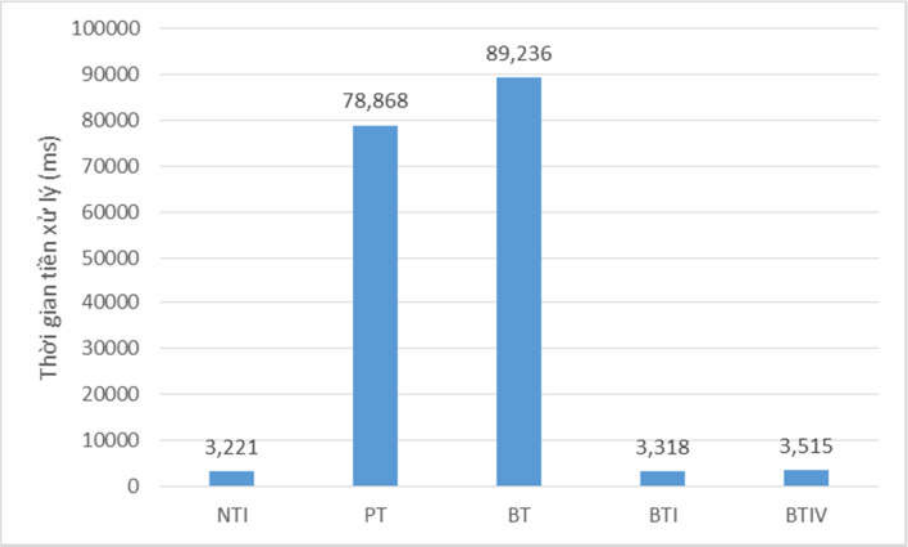
Hình 3. Thời gian tiền xử lý trên tập Connect



Hình 4. Thời gian tiền xử lý trên tập Accidents



Hình 5. Thời gian tiền xử lý trên tập Syn_data1



Hình 6. Thời gian tiền xử lý trên tập Syn_data2

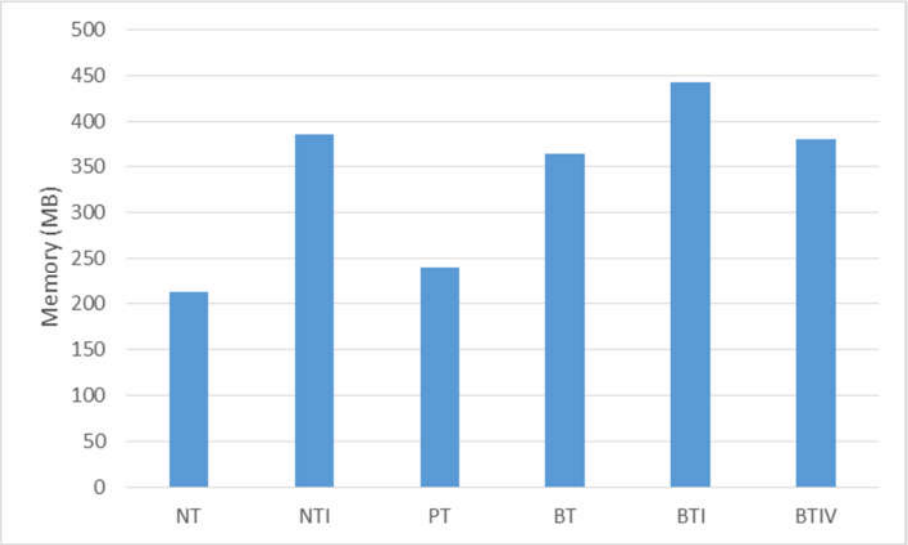
So sánh thời gian tiền xử lý của thuật toán BT trên 2 tập dữ liệu Syn_data1 và Syn_data2 có độ chênh lệch rất lớn khi mà số lượng giao dịch trên hai tập dữ liệu này như nhau. Nguyên nhân là do, tuy có cùng số lượng giao dịch nhưng số item riêng biệt của Syn_data2 lớn hơn Syn_data1 gấp 3.42 lần ($\frac{6\ 7\ 8}{198} = 3.42$) dẫn đến kích thước của BitTable cho Syn_data2 lớn hơn BitTable cho Syn_data1, do đó thời gian tiền xử lý trên tập Syn_data2 lâu hơn tập Syn_data1.

4.4. So sánh về bộ nhớ sử dụng

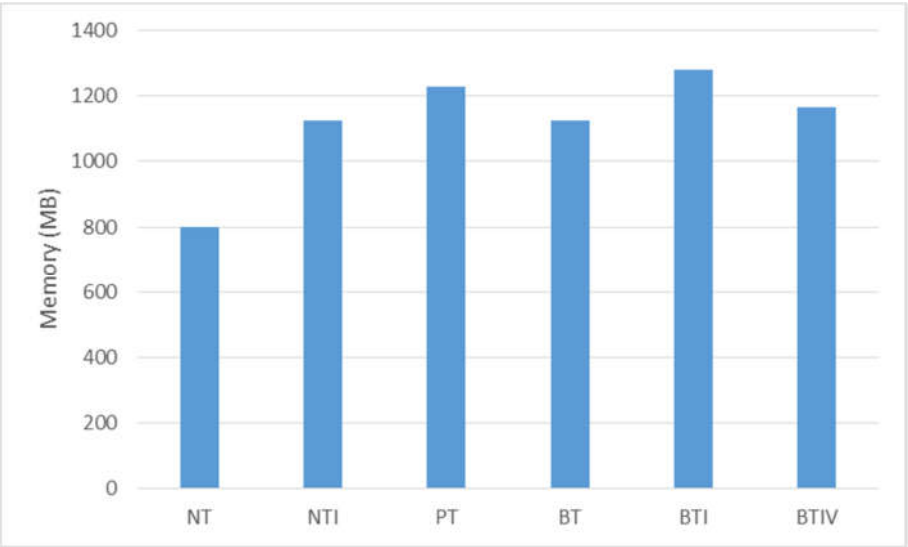
Bộ nhớ sử dụng là bộ nhớ cần thiết để chạy được thuật toán trong quá trình thực nghiệm. Kết quả được đo khi kết thúc 100 câu truy vấn và lấy giá trị trung bình. Bộ nhớ sử dụng của 6 thuật toán: NT,NTI,PT và ba thuật toán đề xuất được thể hiện trong đồ thị Hình 7, Hình 8, Hình 9 và Hình 10.

Dựa vào kết quả hiển thị trên đồ thị Hình 7, Hình 8, Hình 9 và Hình 10, thấy được rằng: Bộ nhớ sử dụng của ba thuật toán đề xuất không chênh lệch nhiều nếu so sánh với thuật toán NTI. Bộ nhớ sử dụng của thuật toán PT tăng lên đáng kể nếu so sánh trên tập Connect với những tập khác. Nguyên nhân là do thuật toán PT không rút tỉa cơ sở dữ liệu đầu vào trước khi xây dựng cây Pi-Tree và số đỉnh phát sinh trên

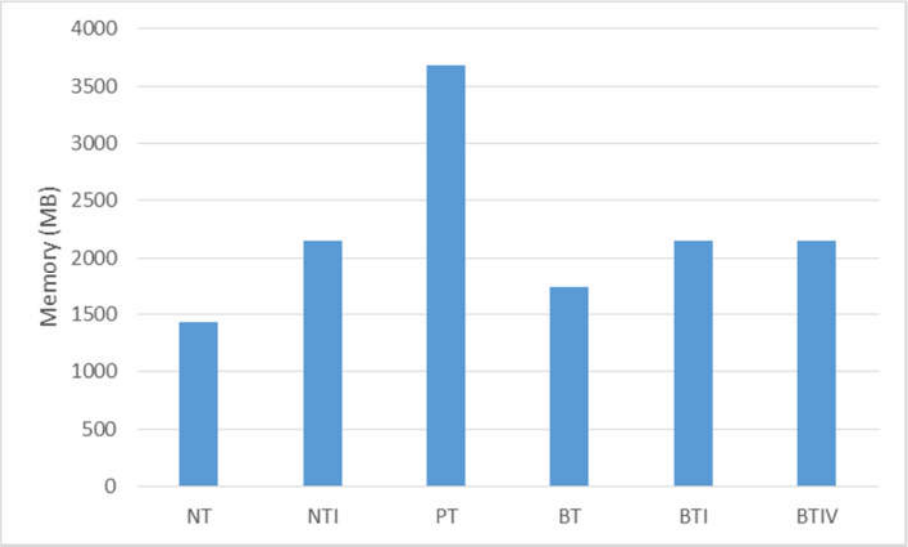
tập Connect là 359,291, trên tập Accidents là 4,243,241, trên tập Syn_data1 là 17,021,247 và trên tập Syn_data2 là 18,152,498.



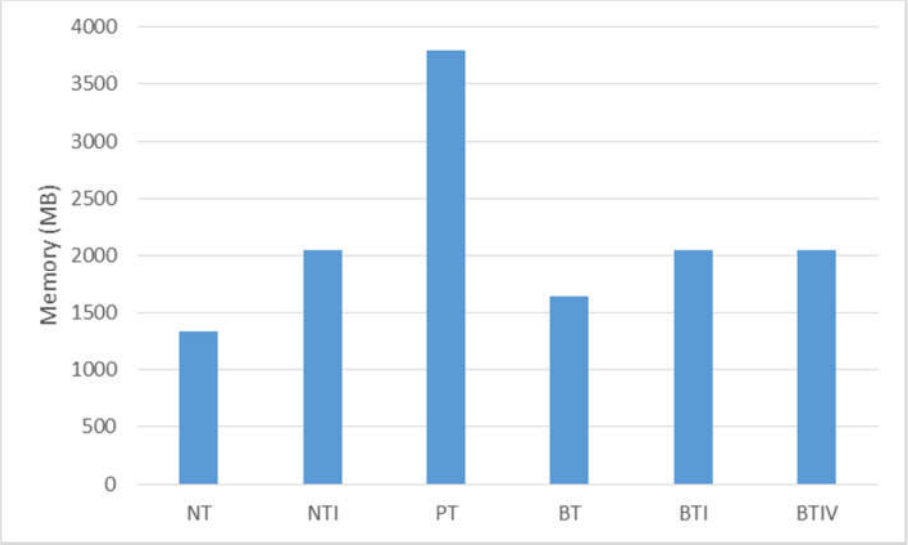
Hình 7. Bộ nhớ sử dụng khi chạy trên tập dữ liệu Connect



Hình 8. Bộ nhớ sử dụng khi chạy trên tập dữ liệu Accidents



Hình 9. Bộ nhớ sử dụng khi chạy trên tập dữ liệu Syn_data1



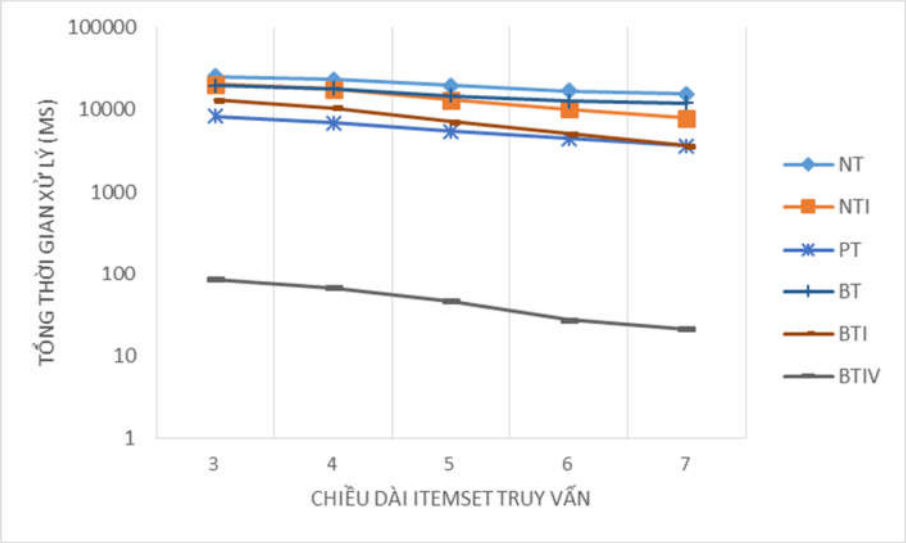
Hình 10. Bộ nhớ sử dụng khi chạy trên tập dữ liệu Syn_data2

4.5. So sánh về thời gian xử lý trên tập dữ liệu Connect

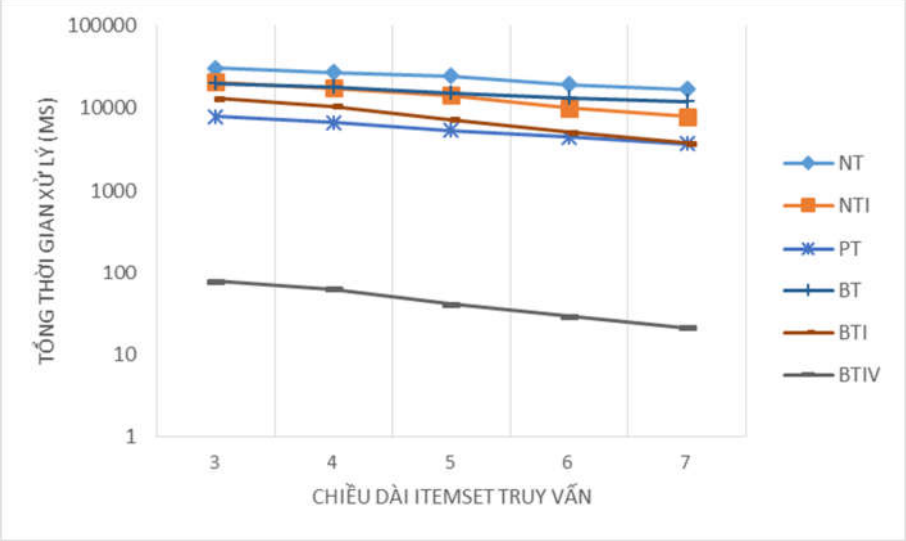
Thời gian xử lý là thời gian được tính từ thời điểm quá trình tiền xử lý kết thúc cho đến lúc thuật toán cho ra kết quả cuối cùng. Hoặc nếu thuật toán không có giai đoạn tiền xử lý thì thời gian xử lý được tính từ lúc bắt đầu chạy thuật toán đến lúc ra kết quả cuối cùng.

Dự vào kết quả hiển thị của đồ thị Hình 11, Hình 12, Hình 13 và Hình 14, thấy được rằng: Thời gian xử lý của thuật toán *BT* và *BTI* không nhanh hơn thuật toán *PT*

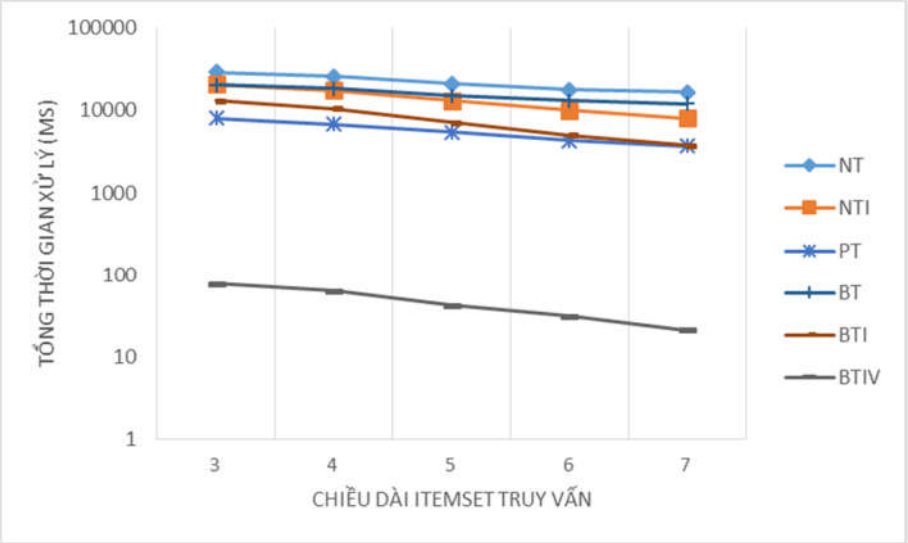
của tác giả bài báo gốc. Tuy nhiên, thuật toán *BTIV* có thời gian xử lý nhanh hơn đáng kể so với những thuật toán khác. Thời gian xử lý của các thuật toán không phụ thuộc vào tham số *k* đầu vào. Vì các thuật toán điều tìm tất cả các item đồng xuất hiện sau đó mới sắp xếp và lấy top-k item đồng xuất hiện. Thời gian xử lý càng nhanh khi chiều dài itemset truy vấn càng dài. Điều này có thể được giải thích là vì có thể độ dài itemset truy vấn càng dài thì số giao dịch có chứa itemset truy vấn càng giảm nên thời gian xử lý cũng giảm theo.



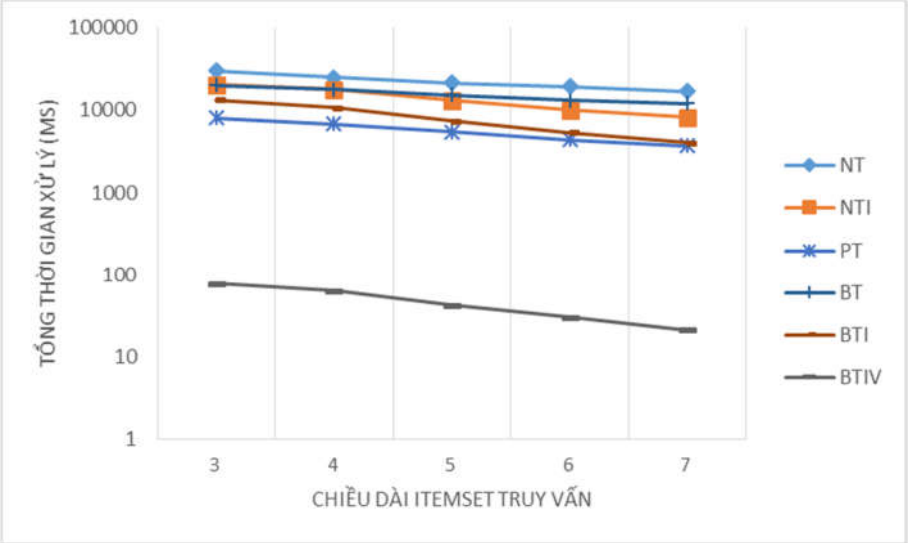
Hình 11. Thời gian xử lý trên cơ sở dữ liệu Connect với K=1



Hình 12. Thời gian xử lý trên cơ sở dữ liệu Connect với K=5



Hình 13. Thời gian xử lý trên cơ sở dữ liệu Connect với K=10



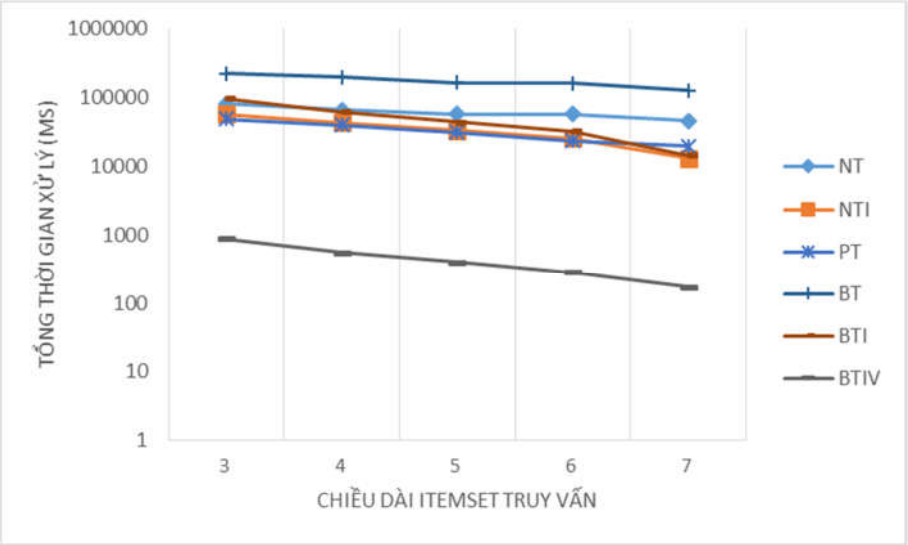
Hình 14. Thời gian xử lý trên cơ sở dữ liệu Connect với K=15

4.6. So sánh về thời gian xử lý trên tập dữ liệu Accidents

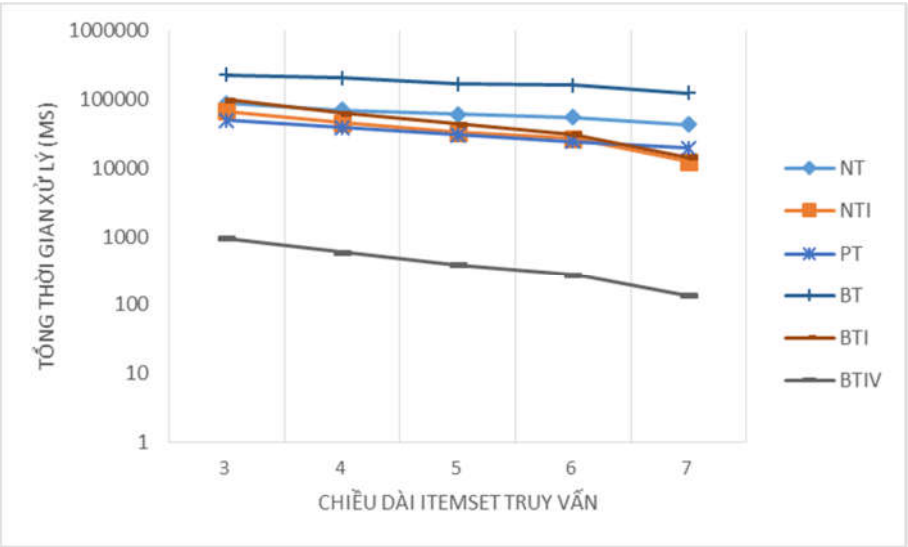
Thời gian xử lý trên tập Accidents được hiển thị trên đồ thị Hình 15, Hình 16, Hình 17 và Hình 18.

Từ kết quả hiển thị trên đồ thị Hình 15, Hình 16, Hình 17 và Hình 18, thấy được rằng: Tỷ lệ giữa các đường biểu diễn trên đồ thị tương tự như trên tập Connect, nhưng có một điểm khác biệt là thời gian xử lý của thuật toán *BT* tốn nhiều thời gian nhất.

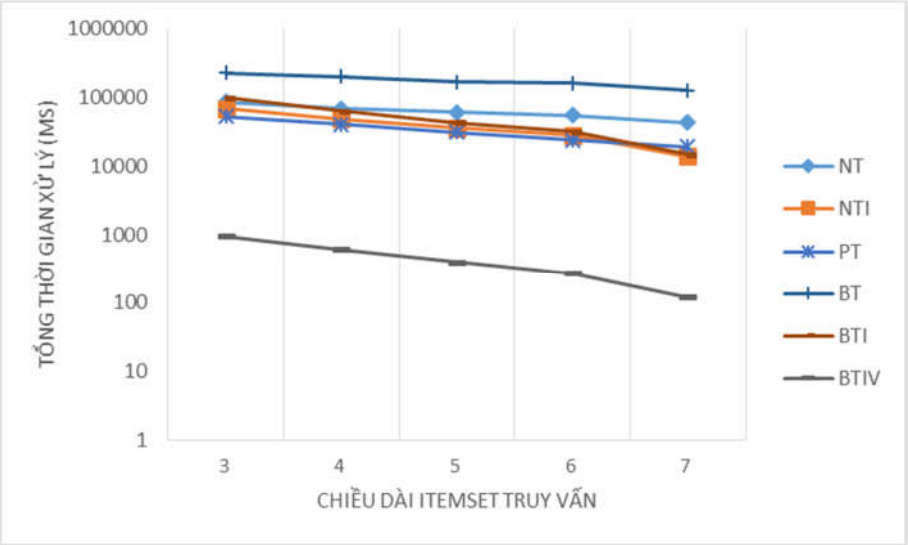
Nguyên nhân là vì khi cơ sở dữ liệu đầu vào bắt đầu lớn, khi chuyển qua BitTable, mật độ các bit 0 quá nhiều dẫn đến thuật toán phải duyệt những bit 0 không cần thiết, do đó thời gian xử lý của thuật toán bị chậm lại.



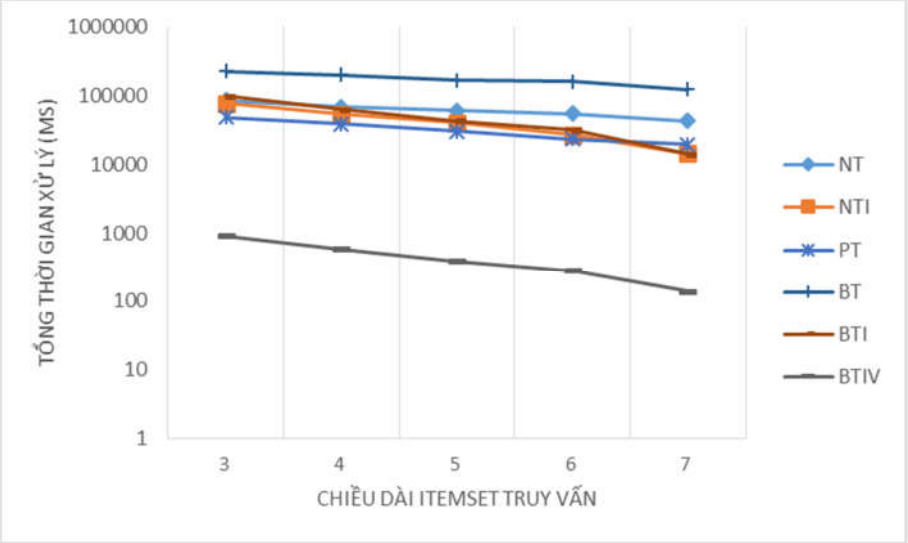
Hình 15. Thời gian xử lý trên cơ sở dữ liệu Accidents với K=1



Hình 16. Thời gian xử lý trên cơ sở dữ liệu Accidents với K=5



Hình 17. Thời gian xử lý trên cơ sở dữ liệu Accidents với K=10



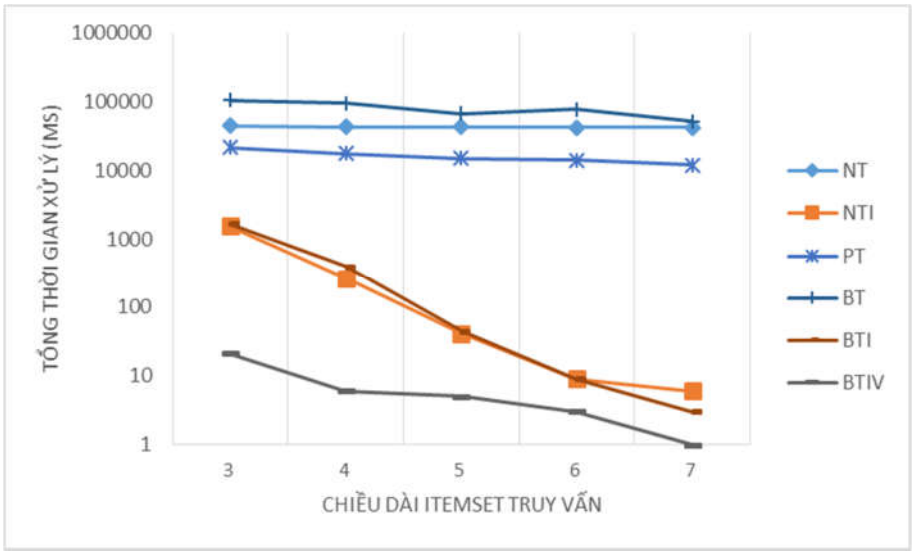
Hình 18. Thời gian xử lý trên cơ sở dữ liệu Accidents với K=15

4.7. So sánh thời gian xử lý trên tập Syn_data1

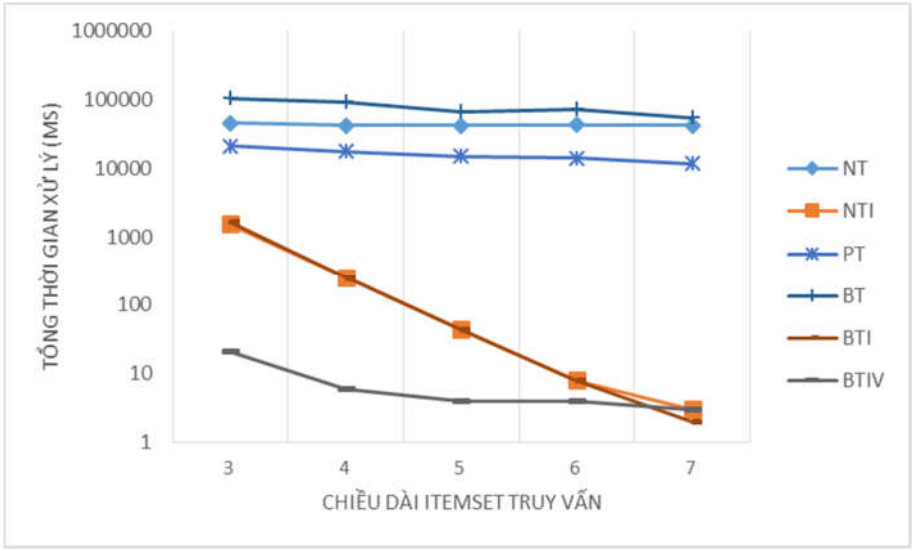
Thời gian xử lý trên tập dữ liệu Syn_data1 được thể hiện trên đồ thị Hình 19, Hình 20, Hình 21 và Hình 22.

Dựa trên kết quả hiện thị trên đồ thị Hình 19, Hình 20, Hình 21 và Hình 22, thấy được rằng: Thời gian xử lý của ba thuật toán NTI, BTI và $BTIV$ nhanh hơn đáng kể so với ba thuật toán còn lại. Nguyên nhân là vì ba thuật toán NTI,BTI và $BTIV$

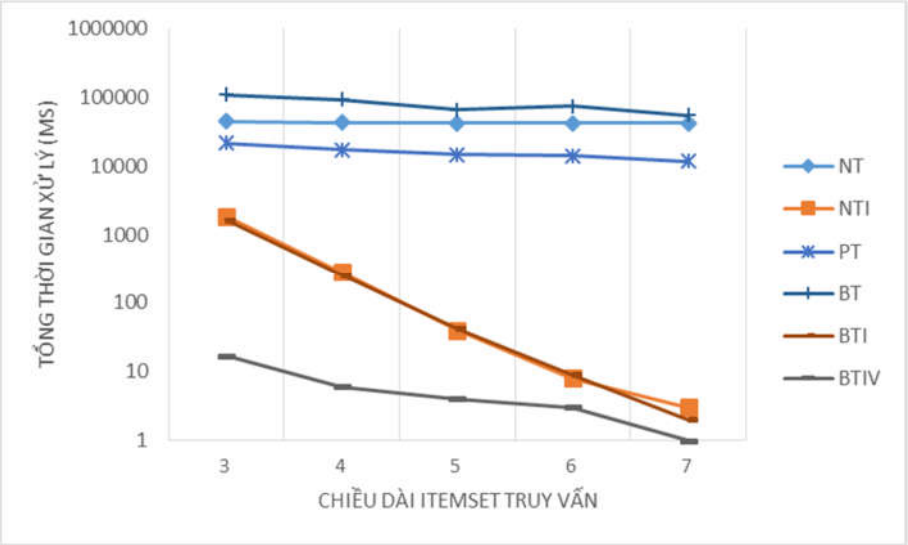
sau bước rút tỉa cơ sở dữ liệu ban đầu, thì chỉ còn trung bình khoản 350 giao dịch thỏa điều kiện để xử lý so với 1,000,000 giao dịch trong cơ sở dữ liệu ban đầu.



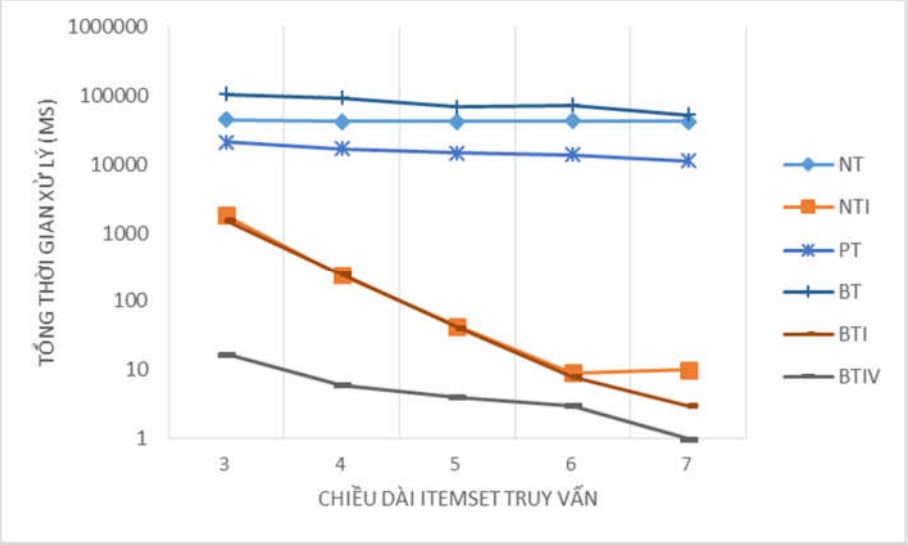
Hình 19. Thời gian xử lý trên tập Syn_data1 với K=1



Hình 20. Thời gian xử lý trên tập Syn_data1 với K=5



Hình 21. Thời gian xử lý trên tập Syn_data1 với K=10



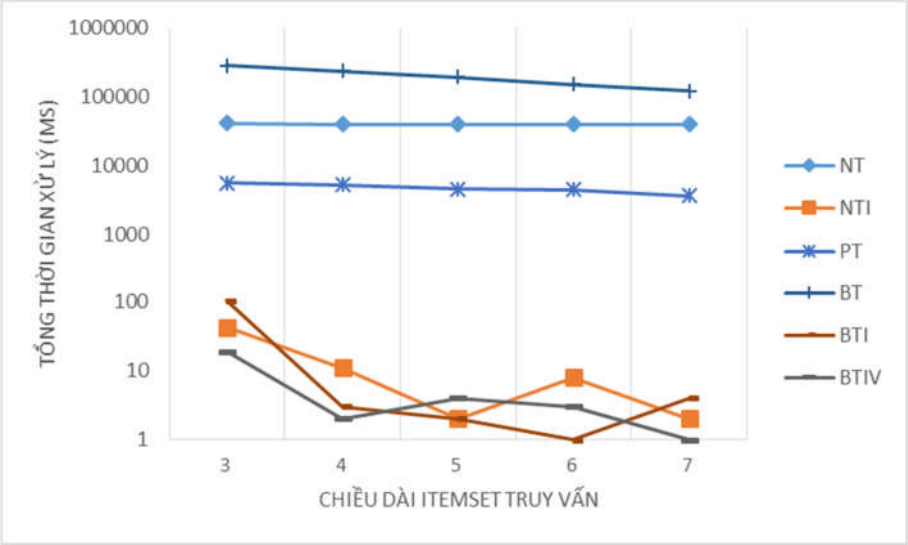
Hình 22. Thời gian xử lý trên tập Syn_data1 với K=15

4.8. So sánh thời gian xử lý trên tập Syn_data2

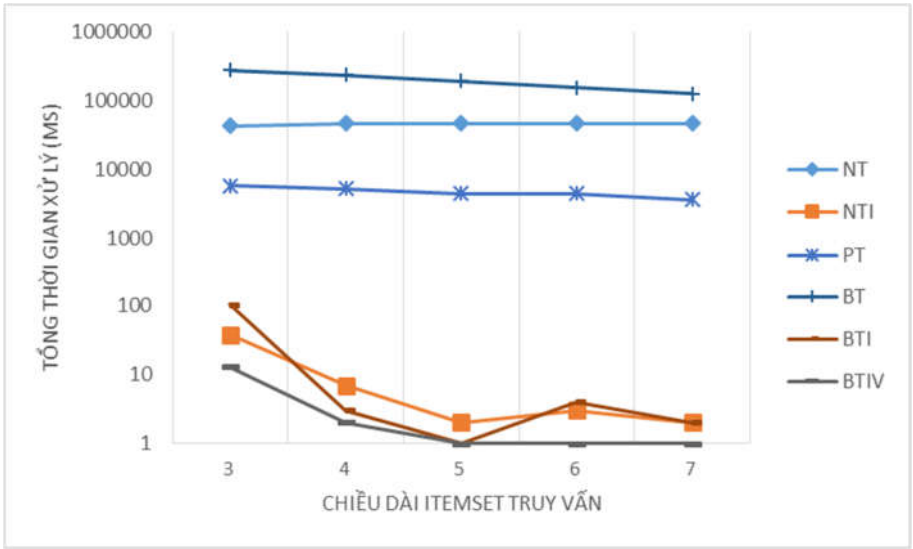
Thời gian xử lý trên tập Syn_data2 được hiển thị trên đồ thị Hình 23, Hình 24, Hình 25 và Hình 26.

Dựa vào kết quả hiển thị trên đồ thị Hình 23, Hình 24, Hình 25 và Hình 26, thấy được rằng: Tỷ lệ giữa các đường biểu diễn trên những đồ thị tương tự như của tập Syn_data1. Tuy nhiên thời gian xử lý của ba thuật toán NTI, BTI và $BTIV$ nhanh hơn rất nhiều so với Syn_data1 tuy có cùng số giao dịch là 1,000,000 giao dịch.

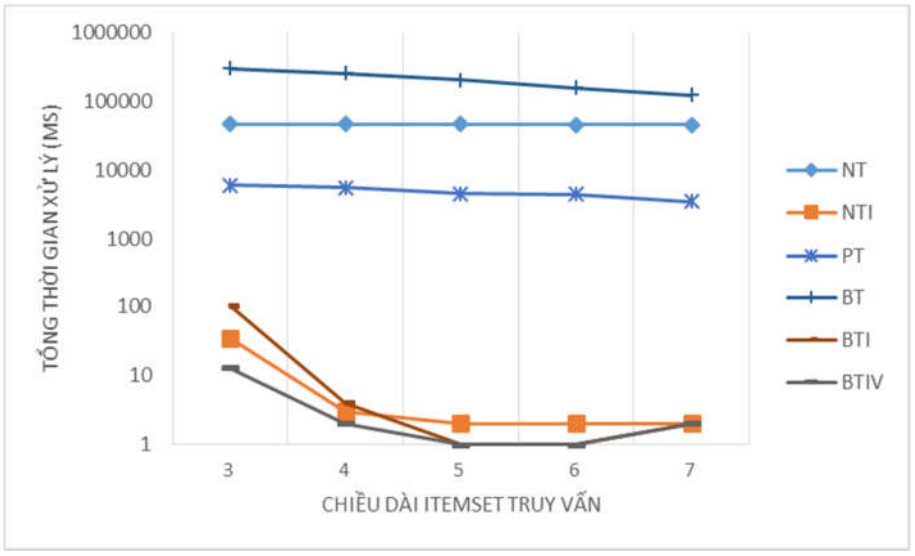
Nguyên nhân là vì trong tập Syn_data2 số lượng item riêng biệt lớn hơn tập Syn_data1 gấp 3.42 lần, nên tỷ lệ của itemset truy vấn xuất hiện trong các giao dịch bị giảm xuống, trung bình chỉ còn 7 trên 1,000,000 giao dịch so với 360 trên 1,000,000 như của tập Syn_data1.



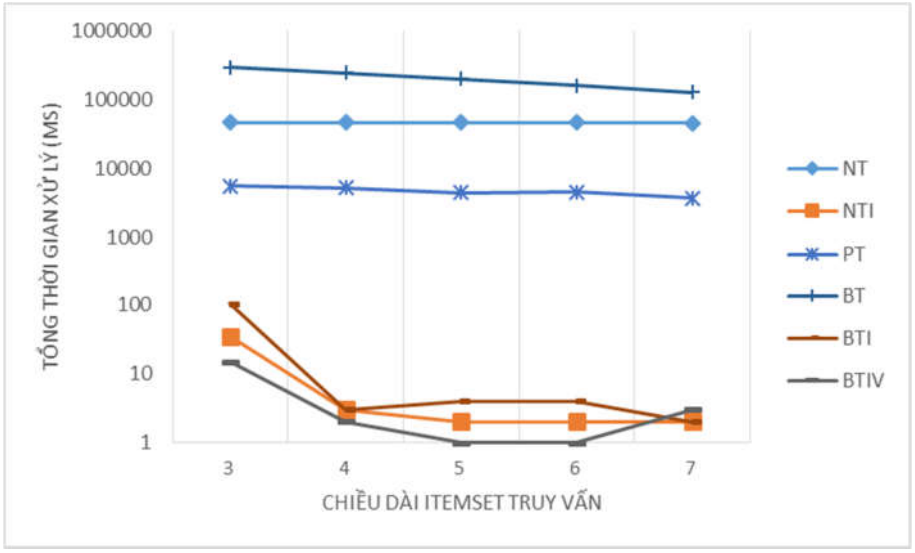
Hình 23. Thời gian xử lý trên tập Syn_data2 với K=1



Hình 24. Thời gian xử lý trên tập Syn_data2 với K=5



Hình 25. Thời gian xử lý trên tập Syn_data2 với K=10



Hình 26. Thời gian xử lý trên tập Syn_data2 với K=15

4.9. Kết luận

Từ kết quả thực nghiệm cho thấy rằng 2 thuật toán được đề xuất là *BT* và *BTI* không hiệu quả hơn *NT*,*NTI* và *PT*. Khi chạy thực nghiệm trên tập Connect thì kết quả rất tốt, nhưng chạy thực nghiệm trên tập Accidents, Syn_data1 và Syn_data2 thì cho ra kết quả ngược lại. Nguyên nhân là vì tập dữ liệu Connect là tập nhỏ và dầy đặc hơn rất nhiều so với ba tập dữ liệu còn lại. Nên khi chạy trên những tập lớn như Accidents, Syn_Data1 và Syn_data2 có mật độ dữ liệu thưa thớt hơn, bảng BitTable,

để đếm số lượng đồng xuất hiện của các item đồng xuất hiện, có quá nhiều bit 0 làm cho thuật toán tốn nhiều thời gian hơn để xử lý.

Thuật toán BT chỉ nhanh hơn thuật toán NT khi tập dữ liệu dầy đặc, vì khi đó không xuất hiện nhiều bit 0 và đồng thời phát huy được ưu điểm của BitTable là dùng phép AND để xác định giao tác có chứa itemset truy vấn hay không và nén dữ liệu gốc xuống gấp nhiều lần, tiết kiệm bộ nhớ và tăng tốc độ xử lý.

Thuật toán BTI chỉ hiệu quả khi dữ liệu thưa thớt, vì sau bước rút tỉa, nếu dữ liệu càng thưa thớt thì dữ liệu cho quá trình xử lý càng giảm, tốc độ xử lý càng tăng.

Thuật toán đề xuất còn lại là $BTIV$ cho kết quả xử lý nhanh hơn đáng kể so với NT,NTI và PT của tác giả Zhi-Hong Deng. Nhờ sự kết hợp ưu điểm của BTI và BitTable, BTI rút tỉa cơ sở dữ liệu đầu vào trước khi xử lý, $BTIV$ cũng vậy. Do đó đã loại bỏ tất cả những ứng viên không cần thiết ra khỏi quá trình xử lý. Đồng thời khi chuyển qua BitTable dạng dọc, gần như thuật toán đã có kết quả cuối cùng. Quá trình xử lý của thuật toán chỉ thực hiện việc đơn giản là đếm số bit 1 trong chuỗi bit tương ứng của từng sự kiện. Mà việc đếm số lượng bit 1 trong 1 chuỗi bit áp dụng kỹ thuật lập trình đặc biệt trên ngôn ngữ C# thì vô cùng nhanh. Thuật toán đơn giản và hiệu quả chính là ưu điểm của $BTIV$.

CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1. Kết luận

Đồ án trình bày được những vấn đề sau:

- Tìm hiểu về bài toán "Khai thác top-k sự kiện đồng xuất hiện" và nội dung chi tiết ba thuật toán NT,NTI và PT.
- Tiếp cận giải pháp biểu diễn dữ liệu bằng BitTable thông qua công trình gốc và những nghiên cứu liên quan.
- Đề xuất giải thuật cải tiến dựa trên phân tích ưu và khuyết điểm của các phương pháp trên.
- Trình bày kết quả thực nghiệm của các phương pháp đề xuất so với ba phương pháp NT,NTI và PT. Từ kết quả thực nghiệm cho thấy thuật toán đề xuất $BTIV$ cho kết quả tốt hơn so với ba thuật toán NT,NTI và PT.

5.2. Hướng phát triển

Thực nghiệm cho thấy thời gian xử lý với phương pháp đề xuất thấp. Điều này gợi mở khả năng áp dụng của phương pháp đề xuất trên dữ liệu quy mô lớn hơn. Tuy nhiên do hạn chế về tài nguyên xử lý và lưu trữ, đề tài xem đây là hướng phát triển tương lai.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] J. C.-W. L. V. T. C. Z. B. L. Philippe Fournier-Viger, "A Survey of Itemset Mining," 2017.

[2] Z.-H. Deng, "Mining Top-K Co-Occurrence Items," 2015.

[3] M. H. Jie Dong, "BitTableFI: An efficient mining frequent itemsets algorithm," *Elsevier B.V,* 2006.

[4] B. Y. Z. X. Wei Song, "Index-BitTableFI: An improved algorithm for mining frequent itemsets," 2008.

[5] T.-P. H. L. Bay Vo, "Dynamic bit vectors: An efficient approach for mining," 2011.

[6] E. D. P. R. A.Saleem Raja, "CBT-fi: Compact BitTable Approach for Mining Frequent Itemsets," 2014.

[7] M. J. Zaki, "Scalable algorithms for association mining," no. IEEE TKDE Journal, 2000.