TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN CUỐI KỲ MÔN NHẬP MÔN CÔNG NGHỆ MÁY TÍNH**

**TÊN ĐỀ TÀI: FREQUENT ITEMSET MINING USING N-LIST**

Người hướng dẫn: **Thầy Huỳnh Quốc Bảo**

Người thực hiện: **Võ Lê Thanh Phú – 518C5002**

Lớp : **18CF0511**

Khóa : **17**

**THÀNH PHỐ HỒ CHÍ MINH NĂM 2019**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN CUỐI KỲ MÔN NHẬP MÔN CÔNG NGHỆ MÁY TÍNH**

**TÊN ĐỀ TÀI: FREQUENT ITEMSET MINING USING N-LIST**

Người hướng dẫn: **Thầy Huỳnh Quốc Bảo**

Người thực hiện: **Võ Lê Thanh Phú – 518C5002**

Lớp : **18CF0511**

Khóa : **17**

**THÀNH PHỐ HỒ CHÍ MINH NĂM 2019**

# **LỜI CẢM ƠN**

Đầu tiên, em xin gởi lời cảm ơn đến thầy Huỳnh Quốc Bảo vì đã giúp đỡ em hoàn thành đề tài này bằng cách gửi gắm những tài liệu tham khảo quý giá

**ĐỒ ÁN ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi và được sự hướng dẫn của Thầy Huỳnh Quốc Bảo. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệt trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu nhập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình**. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 20 tháng 3 năm 2019*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

*Võ Lê Thanh Phú*

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

**Phần xác nhận của GV hướng dẫn**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

**Phần đánh giá của GV chấm bài**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

# **TÓM TẮT**

Frequent itemset mining, lần đầu tiên được đưa ra bởi Agrawal, Imielinski, and Swami vào năm 1993, đã trở thành một phương pháp khai phá dữ liệu phổ biến. Giới thiệu mô hình N-list để giải quyết bài toán FIM và ứng dụng code phát triển từ pseudo code và source code để thiết kế PPC-tree (Pre-order Post-order Coding tree)

**MỤC LỤC**

[LỜI CẢM ƠN 3](#_Toc5702041)

[PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN 5](#_Toc5702042)

[TÓM TẮT 6](#_Toc5702043)

[MỤC LỤC 7](#_Toc5702044)

[DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT 8](#_Toc5702045)

[DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ 9](#_Toc5702046)

[CHƯƠNG 1 – MỞ ĐẦU 10](#_Toc5702047)

[1.1 Lý do chọn đề tài 10](#_Toc5702048)

[1.2 Mục đích nghiên cứu 10](#_Toc5702049)

[1.3 Đối tượng và phạm vi nghiên cứu 10](#_Toc5702050)

[1.4 Ý nghĩa thực tiễn và cơ sở khoa học 10](#_Toc5702051)

[CHƯƠNG 2 – TỔNG QUAN CƠ SỞ LÝ THUYẾT 11](#_Toc5702052)

[2.1 Những nghiên cứu liên quan 11](#_Toc5702053)

[2.2 Những nguyên tắc cơ bản 12](#_Toc5702054)

[2.2.1 Frequent Itemset 12](#_Toc5702055)

[2.2.2 Định nghĩa 1 12](#_Toc5702056)

[2.2.3 Định nghĩa 2 12](#_Toc5702057)

[2.2.4 Định nghĩa 3 14](#_Toc5702058)

[2.2.5 Định nghĩa 4 14](#_Toc5702059)

[2.2.6 Định nghĩa 5 14](#_Toc5702060)

[2.2.7 Định nghĩa 6 14](#_Toc5702061)

[2.2.8 Định nghĩa 7 15](#_Toc5702062)

[2.2.9 Tính Chất 1 16](#_Toc5702063)

[2.3 PrePost+ 16](#_Toc5702064)

[CHƯƠNG 3 – THỰC NGHIỆM 20](#_Toc5702065)

[3.1 Cài đặt 20](#_Toc5702066)

[3.2 Kết luận 21](#_Toc5702067)

[3.3 Trích dẫn và nguồn tham khảo 21](#_Toc5702068)

# **DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT**

**CÁC KÝ HIỆU**

*Sẽ được giới thiệu trong bài để tiện theo dõi*

**CÁC CHỮ VIẾT TẮT**

PPC Pre-order Post-order (PrePost) Coding

FIMFrequent Itemset Mining

DB Database (cơ sở dữ liệu)

T Transaction (Hay còn gọi là basket, chứa các itemset)

# **DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ**

**DANH MỤC HÌNH**

[Fig 1 Cây PPC 1](file:///D:\Project\TRINH_BAY_BAO_CAO\Template_TrinhbayDoAnCuoiKi&amp;BaiTapLonCacMon.dotx#_Toc387689394)3

[Fig 2 N-list của item cấp 1 1](file:///D:\Project\TRINH_BAY_BAO_CAO\Template_TrinhbayDoAnCuoiKi&amp;BaiTapLonCacMon.dotx#_Toc387689394)4

[Fig 3 N-list của item cấp 2, 3 1](file:///D:\Project\TRINH_BAY_BAO_CAO\Template_TrinhbayDoAnCuoiKi&amp;BaiTapLonCacMon.dotx#_Toc387689394)5

[Fig 4 Cây set-enumeration 1](file:///D:\Project\TRINH_BAY_BAO_CAO\Template_TrinhbayDoAnCuoiKi&amp;BaiTapLonCacMon.dotx#_Toc387689394)7

**DANH MỤC BẢNG**

[Bảng 1 Example database 1](file:///D:\Project\TRINH_BAY_BAO_CAO\Template_TrinhbayDoAnCuoiKi&amp;BaiTapLonCacMon.dotx#_Toc387689363)3

# **CHƯƠNG 1 – MỞ ĐẦU**

* 1. **Lý do chọn đề tài**

Hiện tại khai phá dữ liệu đang là một chủ đề phổ biến và được ứng dụng rất nhiều đặc biệt trong ngành marketing, khi cần phải tìm ra những mặt hàng bán chạy để ra chính sách phù hợp, thế nên bài toán Frequent Itemset Mining (Tìm kiếm chuỗi vật phẩm phổ biến) cần phải được nghiên cứu kỹ lưỡng.

* 1. **Mục đích nghiên cứu**

Tìm hiểu vì sao hiện tại thuật toán sử dụng N-list lại là tối ưu nhất cho bài toán FIM.

* 1. **Đối tượng và phạm vi nghiên cứu**

Ở đây chúng ta sẽ giới thiệu thuật toán sử dụng N-list trong bài toàn FIM Mining và ứng dụng trong thực tế thông qua code bằng python.

* 1. **Ý nghĩa thực tiễn và cơ sở khoa học**

Cơ sở của đề tài nghiên cứu đến từ thực nghiệm. Ví dụ: ta có 1 chuỗi các cửa hàng hoặc siêu thị, thông qua trực quan và suy luận bình thường, ta có thể biết là khách hàng khi mua xúc xích sẽ mua thêm tương, bởi vì đơn giản là họ sẽ ăn xúc xích kèm tương. Thông qua kết luận này, ta có thể giảm giá xúc xích nhưng tăng giá tương. Khi xúc xích giảm giá khách hàng chắc chắn sẽ mua nhiều hơn, đồng thời mua tương ăn kèm, khách hàng phần lớn sẽ không biết rằng tương vừa tăng giá hoặc không để ý. Như vậy chúng ta có thể tăng được lợi nhuận. Vậy bài toán đặt ra là “Liệu có còn bất kỳ chuỗi thực phẩm bán chạy nào mà ta có thể khai thác? Nếu có thì tìm chúng như thế nào?” Và thế là ta có bài toán FIM, và thông qua ứng dụng các nhà marketing đã phát hiện ra rằng tã giấy và bia tuy nhìn bề ngoài không liên quan nhưng lại là 1 chuỗi phổ biến trong các hóa đơn và họ sử dụng phương pháp tương tự như trên để thu thêm lợi nhuận.

Ngoài ra còn nhiều ứng dụng khác như là: tìm kiếm những nội dung liên quan đến nhau, phát hiện đạo văn, phát hiện các gen và bệnh tương ứng…

**CHƯƠNG 2 – TỔNG QUAN CƠ SỞ LÝ THUYẾT**

**2.1 Những nghiên cứu liên quan**

Đã có nhiều nghiên cứu để đưa ra các thuật toán để giải quyết vấn đề, các thuật toán được đưa vào hai nhóm chính:

**Apriori**: dựa trên việc chạy liên tục các vòng lặp trên tập dữ liệu để tạo ra các candidate (ứng viên) cấp (k + 1) từ các ứng viên cấp k. Dẫn đến tốn thời gian và bộ nhớ do phải chạy vòng lặp quét tập dữ liệu nhiều lần.

**Frequent Pattern Tree**: đầu tiên phương pháp này nén toàn bộ tập dữ liệu lên cây FP (một cấu trúc cây với các node), sau đó duyệt cây để khai thác tập phổ biến. Ưu điểm của phương pháp này là tiết kiệm bộ nhớ do nén được tập dữ liệu trên cây FP, tuy nhiên lại tốn thời gian duyệt cây FP để khai thác FI, đặc biệt là khi số nút trên cây nhiều.

Những thuật toán này thể hiện sự hiệu quả của mình trong việc giải quyết bài toán tuy nhiên lại có những mặt hạn chế là tốn quá nhiều bộ nhớ và thời gian. Vào năm 2010 và 2012, Deng và các đồng sự[1] giới thiệu mô hình N-list cho bài toán FIM. Mô hình đều dựa trên cấu trúc PPC-tree là một cấu trúc cây với các node.

Cấu trúc PPC có 3 ưu thế chính: Thứ nhất, tất cả thông tin của dataset được gói gọn lại trong một cấu trúc cây với các node, có thể tiết kiệm bộ nhớ. Thứ hai, tần suất xuất hiện của các item (vật phẩm), đều được trữ trong các node, nên khi cần tìm chuỗi item nào phổ biến, ta chỉ cần xét các node chứa các item đó. Thứ ba, khi cần tìm một chuỗi phổ biến có độ dài (k + 1), ta chỉ cần giao các node-list có độ dài k.

Với những ưu điểm kể trên, thuật toán PrePost dựa trên cấu trúc N-list đã thể hiện hiệu suất vượt trội trong bài toán FIM, tuy nhiên do phải mã hóa các itemset trong tập dữ liệu thành các node và áp dụng phương pháp tương tự như thuật toán Apriori để tìm tập phổ biến nên việc tốn bộ nhớ là không thể tránh khỏi.Trong báo cáo này, ta sẽ giới thiệu các tính chất của thuật toán PrePost, sau đó xây dựng thuật toán PrePost+, một phiên bản đã được tối ưu hóa của thuật toán PrePost và cuối cùng là kiểm tra hiệu suất của thuật toán PrePost + so với các thuật toán Apriori và Frequent Pattern.

**2.2 Những nguyên tắc cơ bản**

**2.2.1 Frequent Itemset**

Cho I = { i1, i2, …, in } biểu diễn cho các item xuất hiện trong tập dữ liệu và DB = { T1, T2, …, Tm }là tập cơ sở dữ liệu cho các transaction, với mỗi Tk sao cho (1 ≤ k ≤ m) là một transaction chứa một itemset thỏa Tk ⸦ I. P được gọi là một itemset nếu P chứa các item được định nghĩa trong I. Ta nói Tk chứa P khi và chỉ khi P ⸦ Tk. **Support**,hay ta có thể gọi là tần số, của itemset P là số transaction trong DB có chứa P. Đặt α (0 < α ≤ 1) là giới hạn support nhỏ nhất và |DB| là số transaction trong DB. Itemset P được cho là phổ biến nếu support của P không bé hơn α x |DB|.

### **2.2.2 Định nghĩa 1**

Cho một tập dữ liệu các transaction DB, và mức giới hạn α, phiên bản thu gọn của DB là một cơ sở dữ liệu được tạo ra bởi việc xóa đi các item không phổ biến trong mỗi transaction và sắp xếp các item còn lại theo thứ tự support giảm dần.

Phiên bản rút gọn của DB chứa các thông tin cần thiết liên quan đến các itemset phổ biến. Vì thế chúng ta chỉ cần khai thác các itemset phổ biến từ phiên bản rút gọn của DB thay vì trên chính DB. Trong phần còn lại của bản cáo cáo, khi nói đến DB, ta nói đến phiên bản thu gọn của nó

### **2.2.3 Định nghĩa 2**

Cho phiên bản rút gọn của DB, cây PPC là một cấu trúc được định nghĩa như sau:

1. Bao gồm một root được dán nhãn là “null”, và một tập các item tạo thành các cây phụ là tập con của root.
2. Mỗi node trên các cây con gồm có 5 năm trường: item-name, count, children-list, pre-order, and post-order. item-name là tên của item mà node chứa. count là số transaction được thể hiện bằng các path chỉ đến node. children-list là các tập con của node. Với một node, pre-order của nó là trình tự của node khi duyệt cây theo pre-order traversal và post-order của nó là trình tự của node khi duyệt cây theo post-order traversal

**Ví dụ 1**: Cho một tập dữ liệu DB, với thông tin ở hai cột trái của Bảng 1 và α = 0.5. I = { a, b, c, d, e, f, g } . Tập phổ biến cấp 1 là F1 = { a : 5, b : 4, c : 4, f : 3, d : 3 }. Chú ý: cột cuối cùng chính là DB sau khi thu gọn

**Table 1**:

Tập transaction

|  |  |  |
| --- | --- | --- |
| ID | Items | Ordered Frequent Items |
| 1 | a, b, c, d | a, b, c, d |
| 2 | a, b, e, f | a, b, f |
| 3 | b, c, d, f, g | b, c, d, f |
| 4 | a, c, d, e | a, c, d |
| 5 | a, b | a, b |
| 6 | a, c, f | a, c, f |

**Fig 1**: Cây PPC xây dựng từ **Ví dụ 1**. (x, y) = (pre-order, post-order)

d : 1

f : 1

d : 1

f : 1

c : 1

c : 1

f : 1

d : 1

c : 1

b : 1

b : 3

{}

a : 5

(0, 12)

(4, 0)

(3, 1)

(2, 3)

(1, 7)

(9, 11)

(6, 6)

(10, 10)

(11, 9)

(8, 5)

(7, 4)

(5, 2)

(12, 8)

### **2.2.4 Định nghĩa 3**

PP-code (Pre-order Post-order code) : Cứ mỗi node N trong cây PPC ta gọi {(N.pre-order, N.post-order) : N.count} là PP-code của N.

### **2.2.5 Định nghĩa 4**

N-list: Cho cây PPC, N-list của một item phổ biến là trình tự của tất cả PP-code của các node chứa item ấy được cài đặt trên cây PPC. Được sắp xếp theo thứ tự tăng dần của pre-order.

**Fig 2**: N-list của các item phổ biến trong ví dụ 1

a 🡪 {(1, 7): 5}

b 🡪 {(2, 3): 3}---{(9, 11): 1}

c 🡪 {(3, 1): 1}---{(6, 6): 1}---{(10, 10): 1}

f 🡪 {(5, 2): 1}---{(8, 5): 1}---{12, 8): 1}

d 🡪 {(4, 0): 1}---{(7, 4): 1}---{(11, 9): 1}

### **2.2.6 Định nghĩa 5**

Cho hai item phổ biến i1 và i2, ta kí hiệu i1 ≻ i2 (hay i2 ≺ i1) khi và chỉ khi i1 đứng trước i2 trong DB.

### **2.2.7 Định nghĩa 6**

N-list của chuỗi cấp 2: Cho hai item phổ biến i1 và i2, có N-list lần lượt là {( x11, y11): z11}, {( x12, y12): z12}, …, {( x1m, y1m): z1m} và {( x21, y21): z21}, {( x22, y22): z22}, …, {( x2n, y2n): z2n}. N-list của itemset cấp 2 i1i2 là trình tự của các PP-codes theo thứ tự tăng dần của pre-order và được tạo ra bằng cách giao N-list của i1 và i2, theo quy tắc sau:

Đầu tiên, cứ mỗi {( x1p, y1p): z1p} N-list của i1 (1 ≤ p ≤ m) và {( x2q, y2q): z2q} N-list của i2 (1 ≤ q ≤ n), nếu {( x1p, y1p): z1p} là cha của {( x2q, y2q): z2q}, thì {( x1p, y1p): z2q} được thêm vào N-list của i1i2. Sau đó ta có được N-list khởi tạo của i1i2.

Sau đó, kiểm tra lại N-list khởi tạo của i1i2. Hợp các nodes có dạng {(x1b, y1b): z1b}, {( x1b, y1b): z2b}, …, {( x1b, y1b): zrb}thành một node mới {(x1b, y1b): (z1b + z2b + … + zrb)}.

Từ giờ trở đi ta kí hiệu itemset cấp n là n-itemset.

Thực hiện tương tự, ta có định nghĩa cho N-list của k-itemset (k ≥ 3) như sau.

### **2.2.8 Định nghĩa 7**

N-list của k-itemset: Cho P = ixiyi1i2…i(k - 2) là một itemset (k ≥ 3 và mỗi item trong P đều phổ biến), N-list của P1 = ixi1i2...i(k - 2) là {( x11, y11): z11}, {( x12, y12): z12}, …, {( x1m, y1m): z1m}, và N-list của P2 = iyi1i2...i(k - 2) là{( x21, y21): z21}, {( x22, y22): z22}, …, {( x2n, y2n): z2n}. N-list của P là trình tự của các PP-codes theo thứ tự tăng dần của pre-order và được tạo ra bằng cách giao N-list của P1 và P2, theo quy tắc sau:

Đầu tiên, cứ mỗi {( x1p, y1p): z1p} N-list của P1 (1 ≤ p ≤ m) và {( x2q, y2q): z2q} N-list của P2 (1 ≤ q ≤ n), nếu {( x1p, y1p): z1p} là cha của {( x2q, y2q): z2q}, thì {( x1p, y1p): z2q} được thêm vào N-list của i1i2. Sau đó ta có được N-list khởi tạo của ixiyi1i2…i(k - 2).

Sau đó, kiểm tra lại N-list khởi tạo của ixiyi1i2…i(k - 2). Hợp các nodes có dạng {(x1b, y1b): z1b}, {( x1b, y1b): z2b}, …, {( x1b, y1b): zrb}thành một node mới {(x1b, y1b): (z1b + z2b + … + zrb)}.

**Fig 3**: N-list của ab, bc, abc

ab 🡪 {(1, 7): 3}

bc 🡪 {(2, 3): 1}---{(9, 11): 1}

abc 🡪 {(1, 7): 1}

### **2.2.9 Tính Chất 1**

Cho N-list của bất kỳ k-itemset P = i1 i­2 …ik là {( x1, y1): z1}, {( x2, y2): z2}, …, {( xm, ym): zm}. Support của P được tính bằng z1 + z2 + … + zm.

## **2.3 PrePost+**

Mô hình PrePost+ có cơ sở tương tự như mô hình PrePost, gồm có:

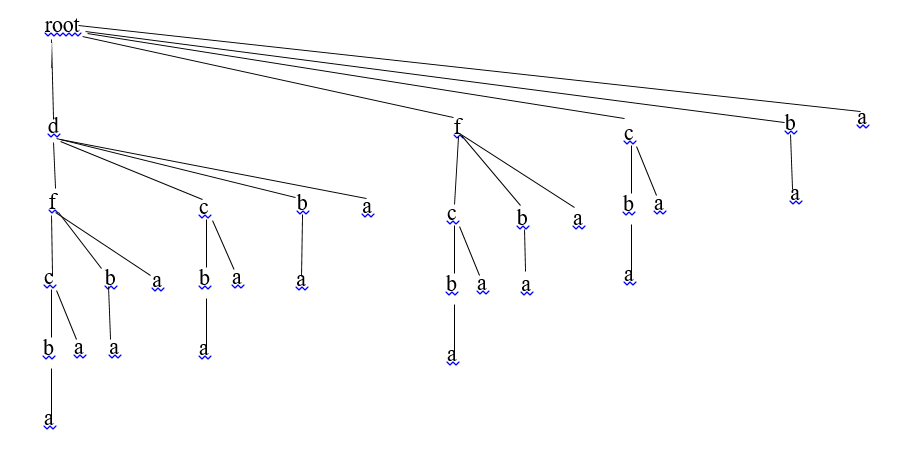
1. Xây dựng cây PPC và xác định các 1-itemset phổ biến
2. Dựa trên cây PPC, xây dựng N-list của từng 1-itemset phổ biến
3. Duyệt cây PPC để tìm các 2-itemset phổ biến
4. Khai thác toàn bộ các (k > 2)-itemset.

Khác biệt chính giữa PrePost và PrePost+ chính là chiến thuật “tỉa cây”, PrePost sử dụng phương pháp “Single path property of N-list”, trong khi PrePost+ sử dụng phương pháp “Superset Equivalence”.

Để tiến trình khai thác dữ liệu dễ dàng, PrePost+ sử dụng cây set-enumeration[2] để thể hiện không gian tìm kiếm. Cho một tập I = {i1, i2,…, im} sao cho i1≺i2≺…≺ im một cây set-enumeration được xây dựng như sau.

1. Root của cây được tạo ra.
2. m phần tử thành các node con của root.
3. Với mỗi node thể hiện tập {ijs i-1...ij1} và ijs, (m – js) node con thể hiện tập {ijs+1ijsi-1...ij1}, {ijs+2ijsi-1...ij1},..., {imiijs-1...ij1} và ijs+1, ijs+2,..., im lần lượt được tạo ra
4. Lặp lại bước ba cho đến khi toàn bộ các lá được tạo và cây set-enumeration được xây dựng hoàn tất

Từ tập phổ biến ở **Ví dụ 1** F1 = { a, b, c, f, d } ta có thể xây dựng một cây set-emuration như sau:

**Fig 4**: ví dụ cho cây set-enumeration****.

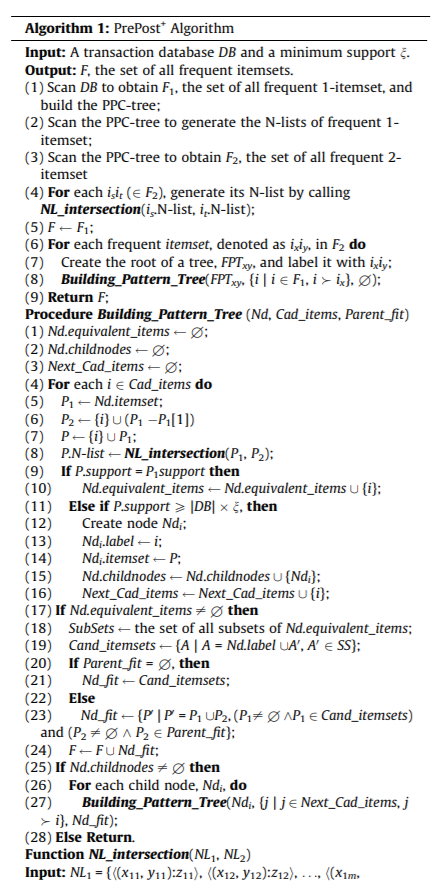
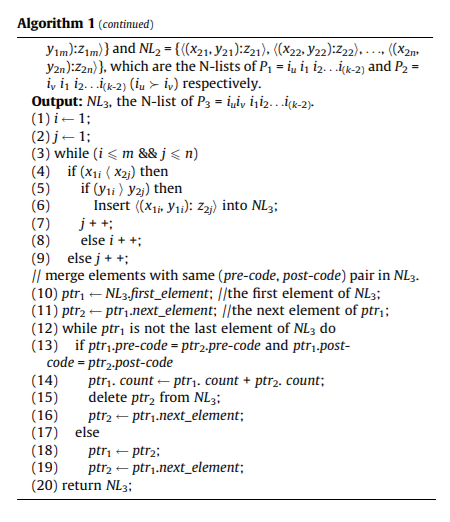
**Tính Chất 2**: Cho tập P và item i (i P), nếu support của P bằng với support của P {i}, với mỗi tập A (A P = i A), support của A P bằng với support của A P {i}.

**Lý luận**: Support của P bằng với support của P {i} tức là, bất kỳ transaction nào chứa P thì cũng chứa i. Cho transaction T, nếu T chứa A P, thì T cũng chứa i. Vì thế support của A P cũng bằng với support của A P {i}.

Áp dụng **Tính Chất 2**, PrePost+ có thể giảm được không gian tìm kiếm một cách đáng kể. Kiểm tra **Fig 4**. Khi tạo ra các node thể hiện *ab*, ta biết được rằng support của *ab* bằng với support của *b* là 3. Cho A là tập bất kỳ chứa *b* mà không chứa *a*. Chú ý, A có thể viết lại là (A-{b}) {b}. Vì vậy, support của A bằng với support của A {b} theo **Tính Chất 2**. Ta kí hiệu tập tất cả itemset phổ biến bên dưới node thể hiện *b* là FISb. FISb/a được xác định là tập các itemset phổ biến không chứa *a* trong FISb và FISb,a là tập itemset phổ biến chứa *a* trong FISb. Hợp của FISb,a và FISb/a là FISb. Để có được FISb,a, ta tìm cây con có phát sinh từ node chứa *ab*. Để có được FISb/a ta tìm các cây con phát sinh từ các node con của node chứa *b* trừ những node con thể hiện *ab*. Dựa vào **Tính Chất 2**, FISb,a tương đương với {A {a}| A FISb/a}. Vì vậy, ta chúng ta có được FISb mà không cần tìm toàn bộ các cây con phát sinh từ node thể hiện *ab*.

**Algorithm 1** là pseudo-code của PrePost+. Bước (1) – (4) của PrePost+ tương tự như quy trình của PrePost, gồm có tạo ra tập phổ biến cấp 1, tập phổ biến cấp 2 và N-list của chúng. Hàm **NL\_intersection()** được sử dụng để tạo ra N-list cho các tập phổ biến cấp (k + 1) bằng cách giao các N-list của tập cấp k. Bước (6) – (9) gọi hàm **Building\_Pattern\_Tree()** để tìm trong cây set-enumeration (từ cấp 2 trở đi) và tìm tất cả tập phổ biến cấp k (k ≥ 3), mở rộng từ mỗi tập phổ biến cấp 2.

**NL\_intersection()** là thuật toán tuyến tính, do đó có hiệu suất cao. Bước (1) – (9) sử dụng phép so sánh hai chiều để tìm tất cả đối tượng trong *NL1*, mỗi đối tượng như thế là một cha của một vài đối tượng trong *NL2*. Tất cả đối tượng tạo nên một kết quả trung gian. Bước (10) – (19) nhập các đối tượng có cùng (pre-code, post-code) trong kết quả trung gian để có được kết quả cuối cùng. Cho *m* và *n* là số đối tượng lần lượt của *NL1* và *NL2*. Độ phức tạp của phép so sánh hai chiều là O(*m* + *n*). Dựa vào phép so sánh hai chiều, số đối tượng của kết quả trung gian ít hơn số đối tượng của *NL2*. Vì thế, độ phức tạp của **NL\_intersection()** là O(*m* + *n*).

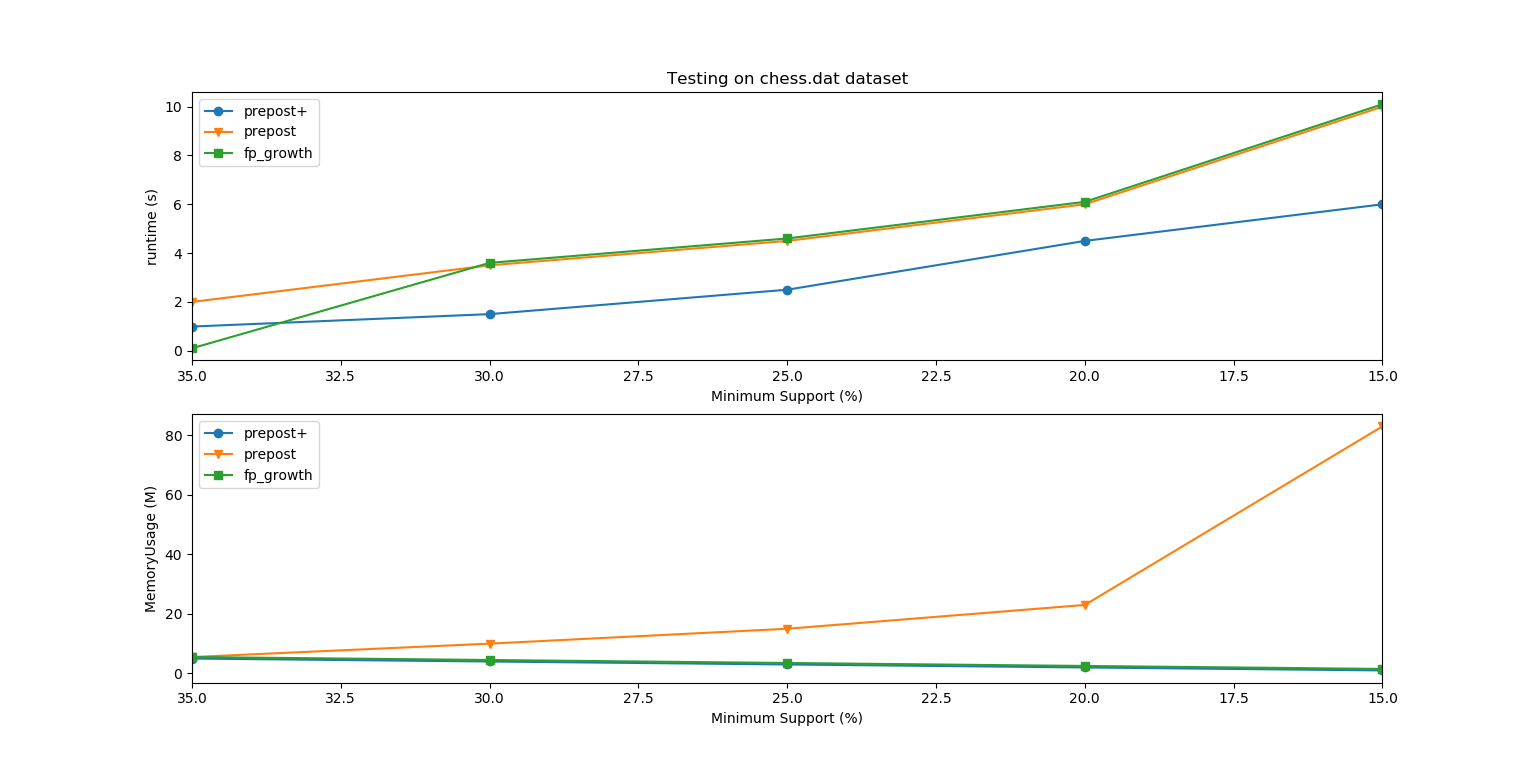
Hàm **Building\_Pattern\_Tree()** áp dụng **Tính Chất 2** để xây dựng cây tập dữ liệu phổ biến được thu gọn mà không cần tạo ra các cây con tại các node con của một node nếu support của tập dữ liệu được thể hiện bởi các node con bằng với support của tập dữ liệu thể hiện bởi node. Bước (4) – (16) kiểm tra từng item trong *Cad\_items*, được dung để mở rộng *Nd*. Trong bước (6), *P1*[1] tức là item đầu tiên trong *P1*. *P* trong bước (7) là một tập với *i* và item đầu tiên và *Nd.itemset* là phần còn lại. Bước (8) tạo ra N-list cho P. Như được thể hiện trong bước (9) và (10), nếu support của *P* bằng với support của *Nd.itemset*, chỉ *i* được thêm vào *Nd.equivalent\_items* mà không tạo ra node thể hiện *P*. Đây được gọi là chiến thuật tỉa cây “Children – Parent Equivalence”. Bước (11) – (16) tìm các item được sử dụng để xây dựng nên các node con của *Ndi* và lưu vào trong *Next\_Cad\_items* để tiếp tục mở rộng. Bước (17) – (24) tìm tất cả các chuỗi phổ biến chứa trong *Nd*. Bước (26) – (27) tiếp tục mở rộng các node con của *Nd* bằng cách đệ quy **Building\_Pattern\_Tree()**.

**CHƯƠNG 3 – THỰC NGHIỆM**

**3.1 Cài đặt**

Ở phần này ta kiểm thử kết quả thuật toán với tập dữ liệu và so sánh hiệu suất với các thuật toán khác. Ở đây ta sẽ sử dụng tập dữ liệu chess.dat với 3196 transaction, do phần cứng giới hạn chỉ có thể kiểm tra trên dữ liệu nhỏ. Dữ liệu được lấy từ <http://fimi.uantwerpen.be/data/>

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset | Chiều dài trung bình mỗi transaction | Số item | Tổng transaction |
| chess.dat | 37 | 75 | 3196 |



Ở đây chúng ta sẽ so dánh hiệu suất khai thác, sự tương quan giữa runtime, bộ nhớ sử dụng và minimum support ở 3 thuật toán PrePost+, PrePost và Frequent Pattern Growth (không so sánh với Apriori vì thực nghiệm cho thấy hiệu suất kém hơn và chênh lệch rất nhiều so với các thuật toán trên)

Ta có thể thấy là PrePost+ có hiệu suất vượt trội hơn hẳn hai thuật toán còn lại khi minimum support giảm xuống mức thấp, tuy nhiên, khi mức minimum support cao thì thuật toán Frequent Pattern lại nhanh hơn rất nhiều lần.

Vấn đề tốn bộ nhớ của PrePost đã được cải thiện bởi PrePost+, nhưng do đây là dữ liệu nhỏ (chỉ từ 3196 transaction và 75 item) khi sang các dữ liệu lớn thì hay khi nhu cầu về minimum support càng lớn thì ta cần phải có các giải pháp để giảm chi phí bộ nhớ ví dụ như áp dụng mô hình MapReduce được cài đặt trên framework Hadoop, chia dữ liệu lớn thành từng gói dữ liệu nhỏ và xử lý song song từng gói dữ liệu.

## **3.2 Kết luận**

Trong bản báo cáo này, ta giới thiệu cấu trúc dữ liệu N-list và thuật toán PrePost+ để giải quyết bài toán Frequent itemset mining. Như thực nghiệm cho thấy, thuật toán PrePost+ có hiệu suất vượt trội tuy nhiên vẫn còn một nhược điểm là tốn bộ nhớ, hiện tại vấn đề này đang được tiếp tục nghiên cứu cải thiện

## **3.3 Trích dẫn và nguồn tham khảo**

[1] <https://www.philippe-fournier-viger.com/spmf/prepost+.pdf>

[2] <https://ieeexplore.ieee.org/document/1512035>

<https://github.com/jacksonpradolima/SPMF/tree/master/src/ca/pfv/spmf>

<https://github.com/evandempsey/fp-growth>

<http://infolab.stanford.edu/~ullman/mmds/ch6.pdf>

Được hỗ trợ bởi các thư viện matplotlib, numpy, panda, …