TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

DỰ ÁN CÔNG NGHỆ THÔNG TIN 2

ĐỀ TÀI: TÌM HIỂU VỀ ITEMSET MINING VÀ CÀI ĐẶT THUẬT TOÁN HIGH OCCUPANCY ITEMSET

GIÁO VIÊN HƯỚNG DẪN: DOÃN XUÂN THANH

TRẦN VĂN SÁNG - 51703173
TRẦN GIA THÁI - 51703184

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

DỰ ÁN CÔNG NGHỆ THÔNG TIN 2

ĐỀ TÀI: TÌM HIỂU VỀ ITEMSET MINING VÀ CÀI ĐẶT THUẬT TOÁN HIGH OCCUPANCY ITEMSET

GIÁO VIÊN HƯỚNG DẪN: DOÃN XUÂN THANH

TRẦN VĂN SÁNG - 51703173

TRẦN GIA THÁI - 51703184

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

LỜI CẢM ƠN

Dự án của chúng em đề cập đến việc tìm hiểu và triển khai thuật toán High Occupancy Itemset. Trong quá trình tìm hiểu về Itemset Mining và cài đặt thuật toán High Occupancy Itemset, chúng em gặp nhiều khó khăn do chưa có kinh nghiệm về đề tài. Chúng em xin gửi lời cảm ơn chân thành đến người hướng dẫn – thầy **Doãn Xuân Thanh** đã giúp đỡ và hỗ trợ chúng em trong quá trình thực hiện dự án.

Và nhóm em xin gửi lời cảm ơn đến Trường Đại học Tôn Đức Thắng đã tạo điều kiện cho chúng tôi được học hỏi thêm thực tế từ người hướng dẫn tận tâm – thầy **Doãn Xuân Thanh**. Nhờ sự hướng dẫn và am hiểu chuyên môn của thầy, chúng em đã có thể hoàn thành dự án của mình.

Em xin chân thành cảm ơn!

TP. Hồ Chí Minh, ngày ... tháng ... năm 2024

Tác giả

(ký tên và ghi rõ họ tên)

Trần Văn Sáng

Trần Gia Thái

LỜI CAM ĐOAN

Tôi xin cam đoan đây là sản phẩm dự án của riêng chúng tôi và được hướng dẫn của GV Doãn Xuân Thanh. Các nội dung nghiên cứu, kết quả trong dự án này là trung thực.

TP. Hồ Chí Minh, ngày ... tháng ... năm 2024

Tác giả

(ký tên và ghi rõ họ tên)

Trần Văn Sáng

Trần Gia Thái

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

_____

_____

_____

_____

_____

_____

_____

_____

TP. Hồ Chí Minh, ngày … tháng … năm 2024

(ký tên và ghi rõ họ tên)

Phần đánh giá của GV chấm bài

_____

_____

_____

_____

_____

_____

_____

_____

TP. Hồ Chí Minh, ngày … tháng … năm 2024

(ký tên và ghi rõ họ tên)

MỤC LỤC

DANH MỤC CÁC TỪ VIẾT TẮT

HEP	High Efficiency Pruning
FHOI	Fast High Occupancy Itemset Mining
DFHOI	Depth First Search for High Occupancy Itemset Mining
StSet	Support Transaction Set
UBO	Upper-bound Occupancy

DANH MỤC CÁC HÌNH VẼ

PHẦN 1 - GIỚI THIỆU CHUNG ITEMSET MINING

1.1. Khái niệm

Itemset Mining (khai thác tập hợp) là quá trình phân tích dữ liệu để tìm kiếm các tập hợp (itemset) của các item xuất hiện trong database. Itemset Mining cung cấp các phương pháp khai thác tập hợp dựa trên các chỉ số để xác định xem itemset đó có được xem là đặc biệt cho so với các itemset khác hay không. Một itemset có thể chứa 1 hoặc nhiều item dựa trên các phép kết hợp tồn tại trong database.

Itemset Mining có thể áp dụng trong nhiều lĩnh vực cụ thể để khai thác hành vi của khách hàng dựa trên các transaction được lưu trữ. Trong bài này, chúng tôi tập trung triển khai thuật toán High Occupancy Itemset (HOI). Thuật toán High Occupancy Itemset có 2 biến thể gồm FHOI (Fast High Occupancy Itemset) và DFHOI (DFS for High Occupancy Itemset). Đây là thuật toán nhận đầu vào (threshold) là 1 ngưỡng chấp nhận do người dùng thuật toán cài đặt, dựa trên chỉ số Occupancy, Support và UBO (Upper-bound Occupancy) để xác định 1 tập hợp có được xem là chiếm dụng cao hay không.

1.2. Đặc điểm của Itemset Mining

Transaction: Trong khai thác dữ liệu, một transaction là một chuỗi các sự kiện xảy ra liên tục trong một khoảng thời gian nhất định. Ví dụ, một chuỗi sự kiện mua hàng online như "đăng nhập -> tìm kiếm sản phẩm -> thêm vào giỏ hàng -> thanh toán" có thể được xem là một transaction.

Tập hợp (Itemset): Một nhóm một hoặc nhiều item trong transaction. Ví dụ, trong một siêu thị, một tập hợp mục có thể bao gồm "sữa" và "bánh mì".

Threshold: ngưỡng chấp nhận do người dùng thuật toán cài đặt.

Support: Chỉ số cho biết tần suất xuất hiện của một tập hợp cụ thể trong toàn bộ tập dữ liệu. Ví dụ, nếu 100 trong tổng số 1000 giao dịch bán hàng bao gồm cả "sữa" và "bánh mì", thì tập hợp mục đó có mức hỗ trợ là 10%.

Frequent Itemsets: Những tập hợp mục xuất hiện cùng nhau trong tập dữ liệu với mức độ thường xuyên vượt qua một ngưỡng hỗ trợ tối thiểu đã được định trước.

Occupancy: độ chiếm dụng trong transaction. Chỉ số này xác định xem 1 itemset chiếm bao nhiêu trong 1 transaction có trong database. Ví dụ: trong một

transaction có chiều dài 10 item, trong đó có item “sữa” và “bánh mì” (tập hợp gồm 2 item) => Occupancy = 2/10 = 1/5.

UBO (Upper-bound Occupancy): ngưỡng chiếm dụng trên của tập hợp trong transaction. Đây là chỉ số dùng để tối ưu thuật toán, từ đó thuật toán sẽ không cần phải duyệt toàn bộ các itemset có thể có trong database nhưng vẫn giữ được hiệu quả do trả về các kết quả phù hợp với ngưỡng chấp nhận.

Occupancy-list: cấu trúc dữ liệu được áp dụng trong thuật toán HOI. Cấu trúc dữ liệu này gồm có 2 phần: item đang xét, transaction chứa item đó và chiều dài tương ứng của transaction. Ví dụ: xét item A, ta có Occupancy-list của item A = {'A': [(T1, 3), (T2, 3), (T4, 2), (T6, 5)]}. Trong đó, T1, T2, T4, T6 là id của transaction, 3, 3, 2, 5 là chiều dài tương ứng của mỗi Tid.

StStet (Support Transaction Set): tương tự như Occupancy-list, nhưng StSet chỉ cần lưu trữ các Tid có chứa tập hợp đang xét thay vì chứa cả chiều dài của transaction đó. Từ đó khi thực hiện thuật toán sẽ giúp giảm sử dụng dữ liệu, từ đó tăng tốc thuật toán. Đây là cấu trúc dữ liệu chính khi áp dụng thuật HOI. Ví dụ: xét item A, ta có StSet của item A = {'A': [T1, T2,T4,T6]}. Trong đó, T1, T2, T4, T6 là id của transaction.

High occupancy itemset: Một itemset được gọi là High Occupancy Itemset khi và chỉ khi tập đó có các chỉ số Occupancy, Support, UBO phù hợp với ngưỡng chấp nhận được đặt ra.

1.3. Các bước cơ bản thực hiện Itemset Mining

Xác định ngưỡng hỗ trợ tối thiểu (threshold): Đây là bước đầu tiên, quyết định một tập hợp được xem là Frequent Itemset hoặc trong bài báo cáo này là High Occupancy Itemset hay không khi và chỉ khi vượt qua được ngưỡng này.

Tìm kiếm các tập hợp mục tần suất cao: Sử dụng các thuật toán như Apriori, FP-growth để tìm ra các tập hợp mục tần suất cao.

Tìm kiếm các tập hợp mục có chiếm dụng cao: sử dụng thuật toán HOI đang được đề cập trong bài.

Phân tích và ứng dụng kết quả: Các tập hợp mục tần suất cao hoặc chiếm dụng cao có thể được sử dụng để phân tích mối quan hệ giữa các mục, giúp trong việc đưa ra quyết định về quản lý hàng tồn kho, tiếp thị, và khuyến mãi ….

1.4. Ứng dụng

Phân tích giỏ hàng: Giúp cửa hàng tối ưu hóa việc sắp xếp sản phẩm và tăng doanh số bán hàng bằng cách tìm hiểu nhóm sản phẩm mà khách hàng thường mua cùng nhau.

Gợi ý sản phẩm: Tạo ra các gợi ý sản phẩm dựa trên những gì khách hàng đã chọn, nhằm tăng cơ hội bán hàng chéo.

Phát hiện gian lận: Phân tích các mẫu giao dịch bất thường để phát hiện gian lận.

Như vậy, Itemset mining cung cấp các phương pháp mạnh mẽ cho việc phân tích dữ liệu và có thể áp dụng trong nhiều lĩnh vực khác nhau, từ bán lẻ đến ngân hàng và y tế, cung cấp thông tin chi tiết giúp cải thiện quyết định kinh doanh và chiến lược tiếp thị.

PHẦN 2 - THUẬT TOÁN HEP

2.1. Khái niệm

Thuật toán HEP(mining High Occupancy Itemset with Efficient Pruning strategy) là một trong những thuật toán nền tảng trong việc khai thác tập hợp chiếm dụng cao trong database. Thuật toán sử dụng chỉ số Occupancy để xác định 1 tập hợp có phải là high occupancy itemset khi và chỉ khi chỉ số Occupancy của tập đó lớn hơn hoặc bằng ngưỡng chấp nhận.

Trong quá trình khai thác tập chiếm dụng cao, không gian tìm kiếm trở nên rất lớn do mỗi tập hợp đều có thể là tập hợp chiếm dụng cao. Như vậy nếu chỉ đơn thuần thực hiện kiểm tra toàn bộ các tập hợp có thể có sẽ gặp vấn đề về bộ nhớ và xử lý. HEP ra đời nhằm mục đích tối ưu không gian tìm kiếm bằng cách sử dụng thêm chỉ số UBO (Upper-bound Occupancy). Như đã đề cập ở phần trên, chỉ số này giúp xác định từ đầu xem tập hợp đó có thỏa mãn ngưỡng chấp nhận, nếu thỏa mãn ngưỡng chấp nhận mới tiếp tục xét đến chỉ số Occupancy của tập hợp.

2.2. Thành phần cơ bản của HEP

- Đầu vào của thuật toán HEP: dữ liệu đầu vào để thuật toán hoạt động cần 2 cột: Tid (chứa các id của transaction), Items (danh sách các item xuất hiện trong Tid tương ứng, lưu ý các item có trong danh sách là các item duy nhất).

	Tid	Items
0	T1	[a, c, d]
1	T2	[a, b, d]
2	T3	[b, c, d, e]
3	T4	[a, d]
4	T5	[c, d, e]
5	T6	[a, b, c, d, e]

- Occupancy: độ chiếm dụng của 1 itemset trong transaction được xác định dựa trên công thức sau:

$$O(P) = \sum_{T \in STSet(P)} \frac{|P|}{|T|}$$

Trong đó:

P: đại diện cho itemset đang xét.

O(P): chỉ số Occupancy của itemset đang xét.

- StSet(P): Support transaction Set của itemset đang xét.

|P| và |T|: chiều dài của tập itemset đang xét và chiều dài của transaction có chứa itemset đó.

Công thức trên định nghĩa các tính Occupancy cho itemset. Ví dụ: sử dụng data mẫu đã cung cấp, xét itemset {a}, ta có itemset {a} tồn tại trong các Tid T1, T2, T4, T6.

$$O(\{a\}) = \sum_{T \in T1,T2,T4,T6} \frac{1}{|T|} = \frac{1}{3} + \frac{1}{3} + \frac{1}{2} + \frac{1}{5} = 1.36$$

- UBO (Upper-bound Occupancy): ngưỡng trên Occupancy của itemset được xác định dựa trên công thức sau:

$$\sum_{i=x}^{u} n_i \times \frac{l_x}{l_i}$$

Trong đó:

n: số lượng chiều dài phân biệt của các transaction chứa itemset đang xét. Số lượng này được sắp xếp từ nhỏ đến lớn dựa trên chiều dài của transaction.

l: chiều dài phân biệt của transaction, được sắp xếp từ nhỏ đến lớn.

UBO của 1 itemset được xác định dựa trên UBO tối đa mà itemset đó có thể có. Ví dụ: sử dụng data mẫu đã cung cấp, xét itemset {a} tồn tại trong các Tid T1, T2, T4, T6. Như vậy chiều dài phân biệt của transaction chứa a bằng l({a}) = {2,3,5}; số lượng chiều dài phân biệt của transaction chứa a bằng n({a}) = {1,2,1}.

$$UBO(\{a\}) = \max_{1 \le x \le u} UBO^{x}_{\{(a)\}} = \max_{1 \le x \le 3} \sum_{i=x}^{3} n_i(\{a\}) \times \frac{l_x(\{a\})}{l_i(\{a\})}$$

$$= \max(1 \times \frac{2}{2} + 2 \times \frac{2}{3} + 1 \times$$

$$\frac{2}{5}, 2 \times \frac{3}{3} + 1 \times \frac{3}{5}, 1 \times \frac{5}{5}) = 2.73.$$

=> Sử dụng các chỉ số UBO và Occupancy sẽ tính toán được itemset đó có được gọi là high occupancy itemset hay không nếu các 2 chỉ số này của itemset đều không nhỏ hơn threshold.

PHẦN 3 - CẢI TIẾN THUẬT TOÁN HEP

3.1. Vấn đề của thuật toán HEP

Thuật toán HEP cung cấp một phương thức mới để khai thác các itemset so với việc thực hiện khai khác frequent itemset. Thuật toán HEP cho ra kết quả các itemset được gọi là chiếm dụng cao có số lượng ít hơn nhiều khi so với frequent itemset. Vì số lượng kết quả nhỏ hơn nhiều, các nhà phân tích có thể tập trung vào các itemset để đi sâu phân tích sự liên quan giữa các item trong itemset với nhau.

Tuy nhiên, thuật toán HEP lại phải thực hiện tạo rất nhiều các itemset ứng viên, từ đó xác định các chỉ số thỏa mãn thuật toán rồi mới ra được kết quả. Quá trình này diễn ra rất lâu, cá nhân tôi khi thực hiện cài đặt thuật toán này trong một số dataset cũng thường xuyên phải chờ đợi một khoảng thời gian rất dài, điều này làm mất rất nhiều thời gian. Trong thực tế, nếu muốn triển khai thuật toán này đòi hỏi một cỗ máy có hiệu năng xử lý rất cao thì mới ra được kết quả. Dù thuật toán HEP cung cấp một chỉ số UBO để cắt tỉa đi các itemset không đạt yêu cầu, nhưng nhìn chung việc tính toán UBO cũng gây ra rất nhiều khó khăn vì quy trình này phức tạp, nhiều vòng lặp và cũng cần phải tính toán toàn bộ UBO cho các itemset ứng viên. Trong trường hợp nếu toàn bộ transaction của dataset cần tính toán độ chiếm dụng có chiều dài bằng nhau, việc tính toán UBO cho dataset này không khả thi do tốn quá nhiều thời gian.

=> Như vậy, các vấn đề khiến thuật toán HEP trở nên chậm chạp là khi thuật toán này yêu cầu quét toàn bộ dataset để tạo các itemset ứng viên và từ đây tiếp tục tính toán UBO cho các itemset ứng viên này.

3.2. Phương pháp cải tiến

Dựa trên các vấn đề được đề cập ở trên, thuật toán FHOI và DFHOI ra đời nhằm khắc phục các vấn đề của thuật toán HEP, từ đó tăng tốc quá trình tính toán để tạo ra các itemset có độ chiếm dụng cao. Từ thực nghiệm cá nhân, tôi nhận ra rằng hai thuật toán này đều đạt hiệu suất tốt hơn nhiều so với HEP. Ở Phần 4 của bài báo cáo, tôi có thực hiện cùng chạy một tập dataset trên cả ba thuật toán HEP, FHOI và DFHOI, thuật toán FHOI và DFHOI đều tỏ ra vượt trội hơn so với HEP khi trả ra kết quả sớm nhất trong khi HEP vẫn đang tiếp tục tính toán.

Đóng góp chính của 2 thuật toán FHOI và DFHOI xuất phát từ việc loại bỏ các itemset ứng viên không có tiềm năng trở thành itemset chiếm dụng cao. Các chiến lược để loại bỏ itemset ứng viên và cải thiện không gian tìm kiếm như sau:

- Thay vì sử Occupancy-list làm cấu trúc dữ liệu chính để thực hiện thuật toán, FHOI và DFHOI yêu cầu sử dụng StSet để làm giảm bộ nhớ lưu trữ.

- Sau khi tính toán tạo ra các itemset ứng viên có một phần tử (sau đây gọi là 1-itemset), gọi k là số lượng item có trong itemset ứng viên (sau đây gọi là k-itemset). Khi k >= 2, dựa trên StSet được tạo ra sau khi có itemset ứng viên, nếu chiều dài của StSet không lớn hơn hoặc bằng ngưỡng tối thiểu do người dùng định nghĩa, các itemset này sẽ bị loại bỏ trước khi tính toán UBO.

- Áp dụng tính chất lớp tương đương để tăng tốc quá trình tạo ứng viên. Trong quá trình thực hiện HEP, tôi nhận ra khi thuật toán khởi tạo các itemset ứng viên, các itemset ví dụ như {a,b} và {b,a} để được xét đến dù rằng chúng đều có vai trò tương đương trong tình huống tính toán itemset chiếm dụng cao. Việc áp dụng tính chất lớp tương đương sẽ rút ngắn được quá trình tạo ứng viên, các ứng viên như ví dụ đã trình bày sẽ không cần phải tính toán hai lần, từ đó đưa tốc độ tính toán nhìn chung của hai thuật toán tăng lên đáng kể.

- Việc áp dụng tính toán UBO là không cần thiết trong trường hợp tất cả các giao dịch trong cơ sở dữ liệu có cùng độ dài với nhau. Nhờ đó, khi FHOI và DFHOI sử dụng cho dataset mà các transaction có độ dài bằng nhau sẽ giúp tăng tốc hơn quá trình ra kết quả do không phải thực hiện tính toán UBO.

- Thuật toán DFHOI bắt nguồn từ FHOI. Tuy nhiên so với FHOI, bài nghiên cứu chỉ ra FHOI tiêu tốn nhiều bộ nhớ để khai thác tập itemset chiếm dụng cao. Thuật toán DFHOI áp dụng kĩ thuật để DFS bằng cách thực hiện đệ quy, từ đó sẽ không cần phải mang theo một lượng lớn data khi chạy thuật toán. Tuy nhiên với thực nghiệm của tôi, thuật toán FHOI và DFHOI hầu như cho thời gian không khác biệt đáng kể, thậm chí ngay trong Phần 4 trình bày chi tiết về cách cài đặt thuật toán và dữ liệu mẫu chúng tôi có, thuật toán DFHOI tỏ ra chậm chạp hơn nhiều khi so với FHOI dù cả hai cùng cung cấp số lượng itemset chiếm dụng cao như nhau.

3.3. Thuật toán FHOI

Thuật toán FHOI bắt đầu tính toán bằng việc quét qua toàn bộ dataset để tạo tập hợp các 1-itemset và danh sách các Tid tương ứng có chứ 1-itemset đó. Thuật

toán cũng đánh index cho độ dài của mỗi transaction và xác định xem toàn bộ transaction trong dataset có cùng độ dài hay không. Tiếp theo, thuật toán gọi phương thức Mine-HOI-1itemset để tính toán các itemset có độ chiếm dụng cao bằng cách sử dụng vòng lặp qua từng item trong tập hợp 1-itemset. Tại đây tính Support của itemset đó và sẽ xảy ra hai trường hợp:

- Nếu các transaction trong dataset không có cùng độ dài, thuật toán sẽ tiến hành tính UBO. Nếu UBO không nhỏ hơn ngưỡng chấp nhận thì tập itemset đó là một tập itemset có độ chiếm dụng cao. Itemset này cũng sẽ được thêm vào các itemset ứng viên cho vòng lặp xử lý tiếp theo (k >= 2).

- Nếu tất cả transaction trong dataset có cùng độ dài thì tập itemset đó sẽ được thếm vào tập itemset ứng viên để chuẩn bị cho vòng lặp tiếp theo. Nếu tập itemset đó cũng đồng thời có Occupancy không bé hơn ngưỡng chấp nhận, itemset này được xem là itemset có độ chiếm dụng cao.

Thuật toán sẽ tiếp tục xử lý cho đến khi không còn tập itemset ứng viên nào. Khi đó chúng ta có 2 tập itemset gồm C1 là tập hợp các 1-itemset ứng viên thõa mãn các điều kiện trên và tập HOI1 là tập hợp các 1-itemset được gọi là itemset có độ chiếm dụng cao.

Tập C1 và HOI1 vừa nhận được ở trên sẽ tiếp tục trở thành đầu vào cho phương thức Mine-HOI-Kitemset. Mỗi itemset ứng viên sẽ được gán cho biến Pi và thực hiện vòng lặp với các Pi+1 trong đó Pi và Pi+1 đều thuộc tập CK-1 (lúc này tập CK-1 chính là tập C1 vừa tính được). Khi thực hiện vòng lặp, xét điều kiện nếu Pi và Pi+1 cùng thuộc một lớp tương đương, thuật toán sẽ thực hiện hợp item giữa Pi và Pi+1 tạo ra P. từ P thực hiện tính toán UBO, nếu UBO không nhỏ hơn ngưỡng chấp nhận, P sẽ được thêm vào tập itemset ứng viên và nếu đồng thời Occupancy của P không nhỏ hơn ngưỡng chấp nhận thì P được gọi là itemset có độ chiếm dụng cao. Thuật toán sẽ tiếp tục thực hiện khởi tạo P mới, và đồng thời lưu trữ lại danh sách itemset ứng viên sau mỗi k lần. Thuật toán sẽ dừng khi tập itemset ứng viên không còn ứng viên nào. Hình bên dưới là mã giả được cung cấp cho quá trình cài đặt thuật toán. Ngầm hiểu trong mã giả có các hàm tính toán như tính UBO, hàm intersection, hàm Occupancy, hàm Support, … đều là các hàm đã được cài đặt từ trước. Phần mã giả chỉ tập trung vào các bước sử dụng các chỉ số để lọc các itemset sao cho hiệu quả nhất mà vẫn đảm bảo tính đúng và đủ của kết quả trả về.

FHOI()

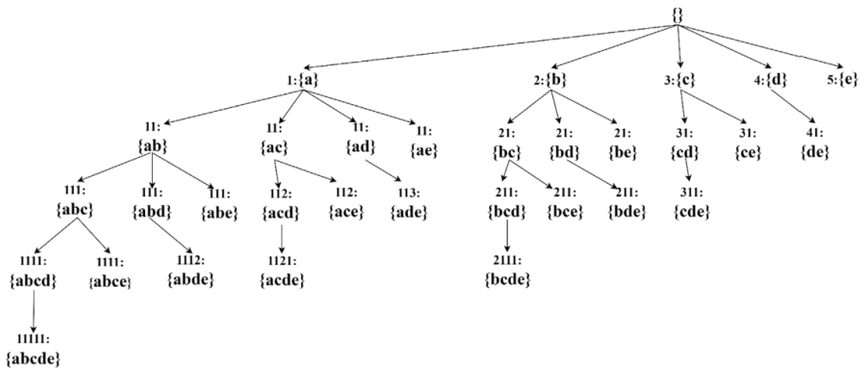
1. Scan D to generate a set of 1-itemset S, index of length L and tidset of each 1-itemset, determine if D has the same length (*hastheSameLengh*).
2. $HOIs \leftarrow \emptyset$;
3. $C_1 \leftarrow \emptyset$; $HOI_1 \leftarrow \emptyset$;
4. $\{C_1, HOI_1\} \leftarrow$ Mine-HOI-1Itemset(S);
5. $HOIs \leftarrow HOIs \cup HOI_1$;
6. $k = 2$;
7. **While** ($C_{k-1} \neq \emptyset$) **do**
8. $\{C_k, HOI_k\} \leftarrow$ Mine-HOI-KItemset (C_{k-1});
9. $HOIs \leftarrow HOIs \cup HOI_k$;
10. $k++$;
11. Return $HOIs$;

Mine-HOI-1Itemset(S)

12. **for each** 1 — itemset P in S **do**
13. **if** $Sup(P) \geq \xi$ **then**
14. **if** *hastheSameLengh* = *False* **then**
15. Scan P.STSet to calculate $UBO(P)$ based on L;
16. **if** $UBO(P) \geq \xi$ **then**
17. $C_1 \leftarrow C_1 \cup \{P\}$;
18. **if** $O(P) \geq \xi$ ***then***
19. $HO_1 \leftarrow HO_1 \cup \{P\}$;
20. **else**
21. $C_1 \leftarrow C_1 \cup \{P\}$;
22. **if** $O(P) \geq \xi$ **then**
23. $HO_1 \leftarrow HO_1 \cup \{P\}$;
24. Return $\{C_1, HO_1\}$;

Mine-HOI-KItemset (C_{k-1})

25. $C_k \leftarrow \emptyset$; // Dictionary of EquivalenceClass and list of candidates
26. $HO_k \leftarrow \emptyset$;
27. **While** $|C_{k-1}| > 0$ **do**
28. $P_1 = C_{k-1}[0]$;
29. **for** *each* itemset $P_2 \in C_{k-1}$ **do**
30. **if** IsSameEquivalenceClass(P_1, P_2) **then**
31. $P \leftarrow P_1 \cup P_2$;
32. P.STSet $\leftarrow P_1$.STSet $\cap$ P_2.STSet;
33. **if** $|P.STSet| \geq \xi$ **then**
34. **if** *hastheSameLengh* = *False* **then**
35. Scan $P.STSet$ to calculate $UBO(P)$ based on *P.STSet;*
36. **if** $UBO(P) \geq \xi$ **then**
37. $P_{tmp} \leftarrow \{P_0, P_1, \ldots, P_{P.length-1}\}$
38. $C_k \leftarrow C_k \cup \{P\}$ & IndexEquivalenceClass(P, P_{tmp})
39. **else**
40. $C_k \leftarrow C_k \cup \{P\}$;
41. $C_{k-1} = C_{k-1} \setminus C_{k-1}[0]$;
42. $HO_k \leftarrow \{P \mid P \in C_k \wedge O(P) \geq \xi\}$;
43. Return $\{C_k, HO_k\}$;



3.4. Thuật toán DFHOI

Như đã trình bày từ phần trước, thuật toán DFHOI là một biến thể của FHOI khi thuật toán này có khả năng khai thác các itemset theo DFS từ đó tối ưu được không gian lưu trữ khi không cần phải mang theo quá các danh sách itemset cũng nhưng các dữ liệu tương ứng.

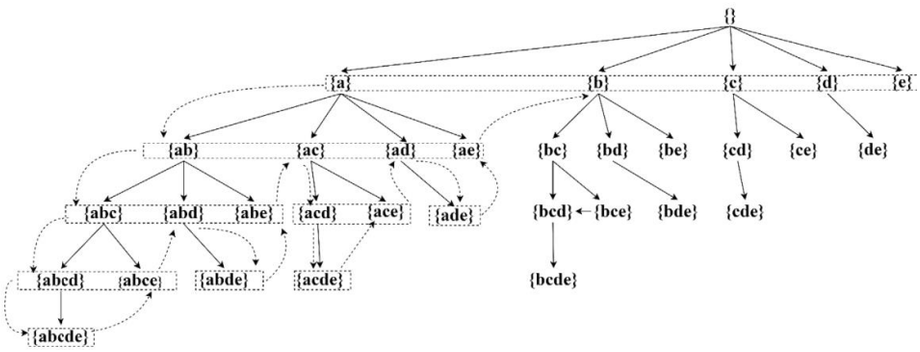
Cách thuật toán hoạt động cũng hầu hết dựa trên thuật toán FHOI vì vậy tôi tái sử dụng lại các hàm tính toán chỉ số tương tự như trên để rút ngắn thời gian cài đặt thuật toán. Trong thuật toán DFHOI, sau khi tạo được tập itemset ban đầu và tính toán danh sách itemset ứng viên và itemset có độ chiếm dụng cao nếu có, thay vì cần sử dụng biến k để đánh dấu số vòng tạo ra các itemset, DFHOI thực hiện đệ quy và áp dụng các chỉ số tính toán tương tự như FHOI cho tới khi không còn tạo ra thêm ứng viên nữa. Bên dưới đây là phần mã giả tương ứng của thuật toán này:

DFHOI()

1. Scan D to generate a set of 1-itemset S, with an index of length L and tidset of each 1-itemset, and determine if D has the same length (*hastheSameLengh*).
2. $HOIs \leftarrow \emptyset$;
3. $C_1 \leftarrow \emptyset$; $HOI_1 \leftarrow \emptyset$;
4. $\{C_1, HOI_1\} \leftarrow$ Mine-HOI-1Itemset(S);
5. $HOIs \leftarrow HOIs \cup HOI_1$;
6. **If** $(C_1 \neq \emptyset)$ **do**
7. $HOIs \leftarrow HOIs \cup$ Mine-Depth-HOIs (C_1);
8. Return $HOIs$;

Mine-Depth-HOIs (C)

9. **for** *each* itemset $P_1 \in C$ **do**
10. $C_l \leftarrow \emptyset$;
11. **for** *each* itemset $P_2 \in C$ with P_2 following P_1 **do**
12. $P \leftarrow P_1 \cup P_2$;
13. $P.STSet \leftarrow P_1.STSet \cap P_2.STSet$;
14. **if** $|P.STSet| \geq \xi$ **then**
15. **if** *hastheSameLengh* = *False* **then**
16. Scan $P.STSet$ to calculate $UBO(P)$ based on L;
17. **if** $UBO(P) \geq \xi$ **then**
18. $C_l \leftarrow C_l \cup \{P\}$
19. **else**
20. $C_l \leftarrow C_l \cup \{P\}$;
21. **Mine-Depth-HOIs** (C_l);
22. $HOIs \leftarrow HOIs \cup \{P \mid P \in C_l \wedge O(P) \geq \xi\}$;


{}
{a}
{b}
{c}
{d}
{e}
{ab}
{ac}
{ad}
{ae}
{bc}
{bd}
{be}
{cd}
{ce}
{de}
{abc}
{abd}
{abe}
{acd}
{ace}
{ade}
{bcd}
{bce}
{bde}
{cde}
{abcd}
{abce}
{abde}
{acde}
{bcde}
{abcde}

PHẦN 4 - CÀI ĐẶT THUẬT TOÁN VÀ XÂY DỰNG TRANG WEB DEMO THUẬT TOÁN

4.1. Ý tưởng

- Sử dụng thuật toán DFHOI để tính toán và đưa ra gợi ý cho khách hàng khi họ thêm sản phẩm vào giỏ hàng của mình. Trong đó các Itemset để thực hiện là dựa vào các đơn đặt hàng trước đây, từ đó sẽ tính được các sản phẩm thường xuyên được mua chung trong cùng một đơn hàng. Dựa vào đó sẽ lấy ra các sản phẩm gợi ý cho khách hàng khi họ thêm một sản phẩm vào giỏ hàng.

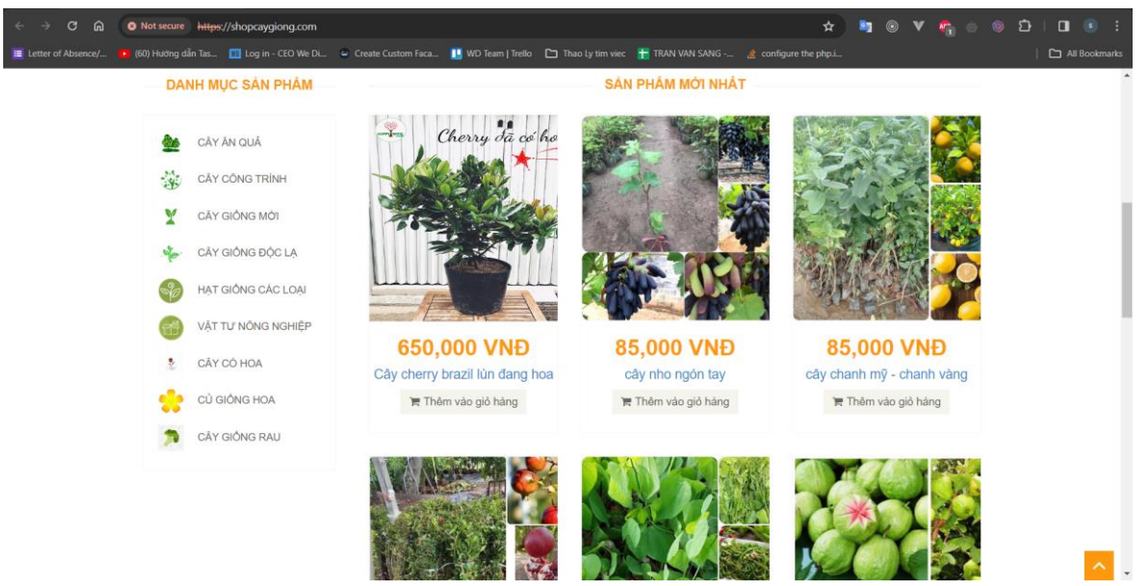
4.2. Các bước thực hiện

a) Tạo website bán hàng:

Tạo website đơn giản sử dụng framawork laravel gồm các chứ năng như hiện thị sản phẩm, thêm vào giỏ hàng, đặt hàng….

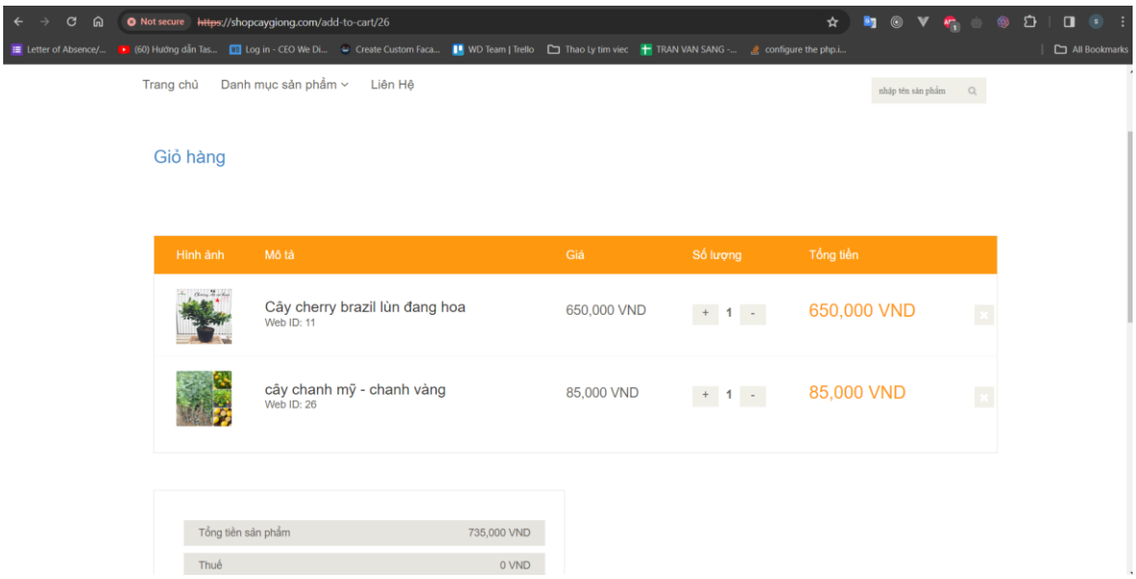
Deploy website bằng máy ảo homestead với tên miền shopcaygiong.com

Trang chủ website:



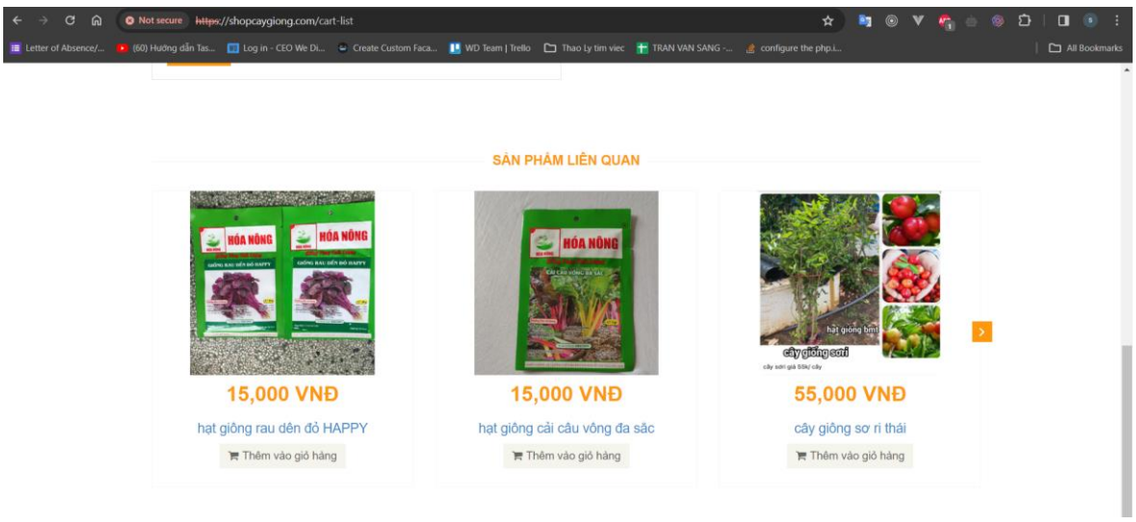
Hình 4.1 Trang chủ Website

Trang giỏ hàng;



Hình 4.2 Trang giỏ hàng

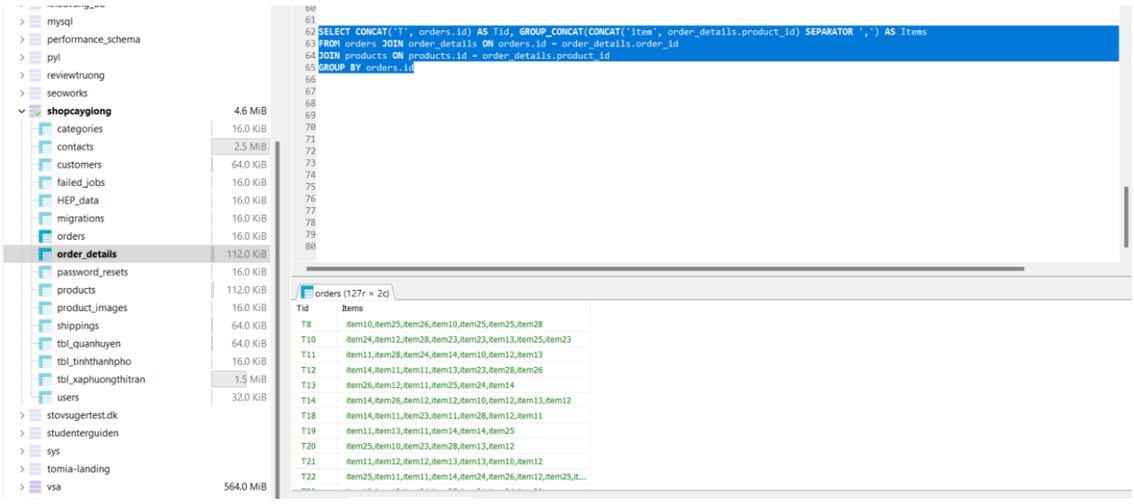
Tạo một slider hiện thị sản phẩm liên quan khi khách thêm sản phẩm vào giỏ hàng.



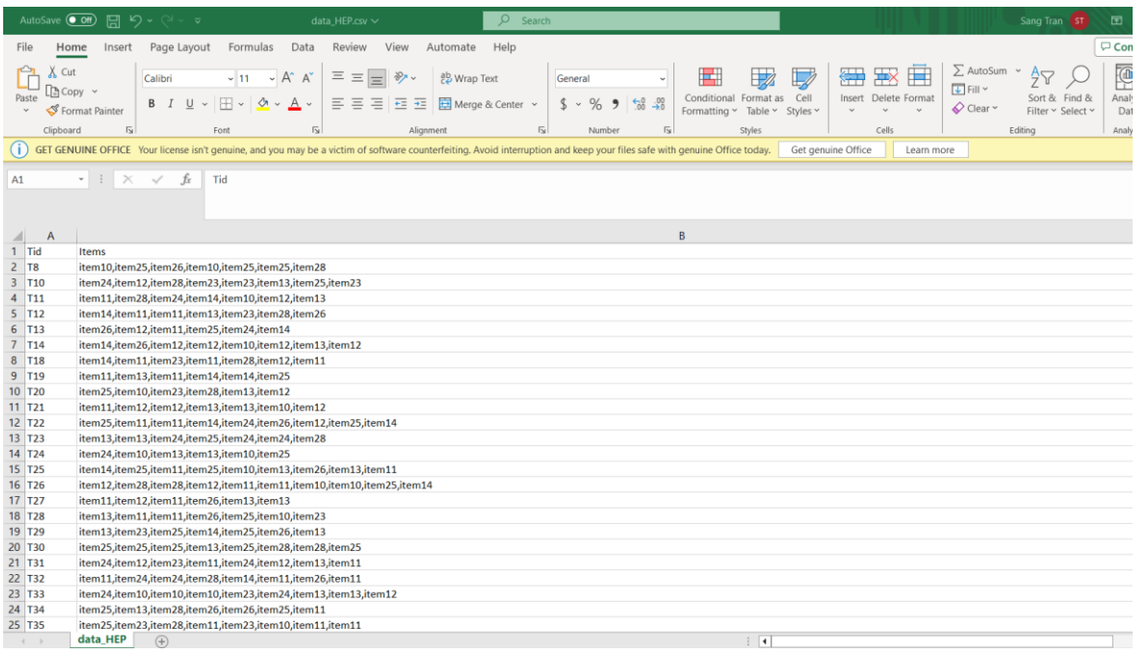
Hình 4.3 Trang menu sản phẩm hàng bán

b) Chuẩn bị dữ liệu để chạy thuật toán.

- Chuẩn bị Item: sử dụng câu sql và lấy tất cả mã đơn hàng và tất cả mã sản phẩm thuộc đơn hàng đó sau đó xuất dữ liệu thành file csv.



Hình 4.4 Câu sql



Hình 4.5 File csv

c. Tiến hành chạy thuật toán với dữ liệu mẫu.

- Import một số thư viện cần thiết:

```
import pandas as pd
from collections import Counter
import time
import matplotlib.pyplot as plt
# from fruithut.fruithut import *
# from mushroom.mushroom import *
# from foodmart.foodmart import *
# from chess.chess import *
from demo_web.demo_web import *
```

- Đọc, in dữ liệu từ file csv và chuyển thành dạng dữ liệu mong muốn:

```
df['Item_Length'] = df['Items'].apply(lambda items: len(items))
len_df = len(df)
df
```

✓ 0.0s

	Tid	Items	Item_Length
0	T8	[item11, item10, item27, item23, item24, item2...	9
1	T10	[item10, item27, item23, item24, item25, item2...	10
2	T11	[item11, item10, item27, item23, item24, item2...	10
3	T12	[item11, item27, item24, item25, item28, item1...	9
4	T13	[item11, item10, item24, item25, item28, item1...	9
...	...	...	...
122	T134	[item10, item12]	2
123	T135	[item24, item23, item12]	3
124	T136	[item23]	1
125	T137	[item27, item26, item28, item13]	4
126	T138	[item24, item27, item13, item12]	4

127 rows × 3 columns

Hình 4.6 Đọc dữ liệu từ file csv

- Transform từ dữ liệu ban đầu về cấu trúc dữ liệu StSet để thực hiện thuật toán:

```
# calculate stset: {'a': [T1, T2, T4, T6]} - list Tid containing unique item
def cal_stset(df):
    df_unpivot = df.explode('Items')
    df_stset = df_unpivot.groupby('Items').agg({'Tid': [list], 'Item_Length': list}).reset_index()
    df_stset.columns = ['Items', 'StSet', 'Length_transaction']
    df_stset['Items'] = df_stset['Items'].apply(lambda x: [x])
    df_stset
    return df_stset
```

✓ 0.0s

Hình 4.7 Hàm transform dữ liệu thành StSet

- Viết hàm tính toán Support cho itemset:

```
# calculate support - count number of Tid containing unique item
def cal_support(df_stset):
    df_stset['Support'] = df_stset['StSet'].apply(len)
    return df_stset
```

✓ 0.0s

Hình 4.8 Hàm tính Support

- Viết hàm tính toán Occupancy cho itemset:

```python
# calculate occupancy - O(P) = ∑ T ∈ STSet(P) |P|/|T|
# |P|: len(unique item) itemset {a} =>1
# |T|: len(Tid) 1/3 + 1/3 + 1/2 + 1/5
def cal_occupancy(df_stset):
    occupancy_data = []
    for index, row in df_stset.iterrows():
        item = row['Items']
        length_transaction = row['Length_transaction']
        total = 0
        for length in length_transaction:
            total += len(item) / length
        occupancy_data.append({'Items': item, 'Occupancy': round(total, 2)})

    df_occupancy = pd.DataFrame(occupancy_data)
    df_stset['Occupancy'] = df_occupancy['Occupancy']
    return df_stset
```
✓ 0.0s

Hình 4.9 Hàm tính Occupancy

- Viết hàm tính UBO cho itemset. Vì công thức này phức tạp hơn các công thức khác trong bài nên tôi chia giai đoạn tính toán thành nhiều phần khác nhau và các phần được gọi trong một hàm tổng:

```python
# UBO calculation methods: main function
def cal_UBO(df_stset):
    df_stset = df_prepare_UBO(df_stset)
    df_stset = calculate_maxUBO(df_stset)
    return df_stset
```
✓ 0.0s

- Hàm df_prepare_UBO() được sử dụng để chuyển đổi dữ liệu từ StSet sẽ định dạng tính toán UBO phù hợp trước khi thực sự thực hiện tính toán UBO:

```python
# ex: 'a': {'l(a)': [2, 3, 5], 'n(a)': [1, 2, 1]}
def df_prepare_UBO(df_stset):
    l_item_list = []
    n_item_list = []
    for index, row in df_stset.iterrows():
        item = row['Items']
        length_transaction = row['Length_transaction']

        l_item = sorted(set(length_transaction)) # get unique len(Tid) => sort ascending

        counter = Counter(length_transaction)
        n_item = [counter[i] for i in l_item] # count unique len(Tid) in occupancy_list => same index with l_item

        l_item_list.append(l_item)
        n_item_list.append(n_item)

    df_stset = df_stset.assign(l_item=l_item_list, n_item=n_item_list)
    return df_stset
```
✓ 0.0s

- Hàm calculate_maxUBO() được sử dụng để nhận đầu vào của StSet, sau đó lần lượt chuyển đổi thành các list để thuận tiện cho việc tính toán:

```python
# get max from summarize => save max value in UBO by key
def calculate_maxUBO(df_UBO):
    df_UBO['List_UBO'] = None # create new column
    df_UBO['Max_UBO'] = None # create new column
    for index, row in df_UBO.iterrows():
        length = row['l_item'] #get list of len(Tid) containing unique item
        number_transaction = row['n_item'] # count unique len(Tid) in occupancy_list

        ubo = ubo_final(length, number_transaction) # get list of UBO by i. ex: [2.73, 2.6, 1.0]
        max_ubo = max(ubo) # max list of UBO

        df_UBO.at[index, 'List_UBO'] = ubo # save result in df
        df_UBO.at[index, 'Max_UBO'] = max_ubo # save result in df

    return df_UBO
```

- Hàm ubo_final() nhận đầu vào từ hàm calculate_maxUBO(), sau mỗi vòng lặp khi nhận được hai list giá trị của hàm này, hàm ubo_final() sẽ bỏ đi các phần tử đã tính toán và tiếp tục gọi đến hàm call_ubo():

```
# summarize: ∑ni x lx/li => save to list
def ubo_final(length, number_transaction):
    ubo = []
    for i in range(len(length)):
        # ex: len = [2,3,5], num_trans = [1,2,1]
        # i = 0 => len = [2,3,5], num_trans = [1,2,1]
        # i = 1 => len = [3,5], num_trans = [2,1]
        # ...
        ubo.append(cal_ubo(length[i:], number_transaction[i:])) # save result cal_ubo for each i => get maxUBO
    return ubo
✓ 0.0s
```

- Hàm cal_ubo() thực hiện tính toán total cho mỗi vòng lặp và trả về một giá trị để ghi nhận vào hàm ubo_final(). Hàm ubo_final() trả về list các UBO đã được tính của itemset đó và trả về giá trị lớn nhất tồn tại trong list:

```
# calculate according to the formula: ni x lx/li
def cal_ubo(l, n):
    total = 0
    for i in range(len(l)):
        total += n[i] * l[0] / l[i]
    return round(total, 2)
✓ 0.0s
```

- Hàm mine_hoi_1_itemset() được sử dụng để khởi tạo các tập 1-itemset ứng viên và tập 1-itemset chiếm dụng cao nếu có. Đây là hàm dùng chung cho hai thuật toán:

```
def mine_hoi_1_itemset(threshold, hastheSameLengh, df_stset):
    C1 = []
    HOI1 = []
    for index, row in df_stset.iterrows():
        item = row['Items'] # 1-itemset in row
        support = row['Support'] # support of 1-itemset
        occupancy = row['Occupancy'] # occuopancy of 1-itemset
        max_ubo = row['Max_UBO'] # max_ubo of 1-itemset

        if support >= threshold:
            if hastheSameLengh is False:
                if max_ubo > threshold:
                    C1.append(item)
                    if occupancy >= threshold:
                        HOI1.append(item)
            else:
                C1.append(item)
                if occupancy >= threshold:
                    HOI1.append(item)

    return C1, HOI1
```

- Hàm df_intersection() thực hiện nhận đầu vào là list của các itemset đang xét. Hàm này thực hiện tìm kiếm trong StSet đã tạo trước đó, sau đó lấy các Tid giao nhau giữa hai itemset này:

```python
def df_intersection(items1, items2, df_stset):
    df_intersection = pd.DataFrame(columns=['Items', 'StSet'])
    # set1 = set(items1)
    # set2 = set(items2)
    list_item = list(set(items1) | set(items2))

    list_occupancy_item = []

    for i in list_item:
        list_occupancy_item.append(df_stset[df_stset['Items'].apply(lambda item: i in item)]["StSet"].iloc[0])

    intersection_list = set(list_occupancy_item[0])
    for sublist in list_occupancy_item[1:]:
        intersection_list = intersection_list.intersection(sublist)

    df_intersection = df_intersection.append({'Items': list_item, 'StSet': intersection_list}, ignore_index=True)
    df_intersection['Length_transaction'] = df_intersection['StSet'].apply(lambda x: sorted([len(df[df['Tid'] == tid]['Items'].iloc[0]) for tid in x]))

    # list_length = df_intersection['StSet'].apply(lambda x: [len(df[df['Tid'] == tid]['Items'].iloc[0]) for tid in x]).iloc[0]
    # sort_data = sorted(zip(intersection_list, list_length))

    return df_intersection
```

- Hàm is_same_equivalenct_class() xác định xem hai itemset có phải cùng một node cha không:

```python
def is_same_equivalence_class(P1, P2):
    if len(P1) == len(P2):
        if len(P1) == 1:
            if P1 == P2:
                return False
            else:
                return True
        else:
            if P1 == P2:
                return False
            else:
                new_P1 = P1[:-1]
                new_P2 = P2[:-1]
                if new_P1 == new_P2:
                    return True
                else:
                    return False
    else:
        return False
```

- Hàm cal_occupancy_candidate() phục vụ cho quá trình tính toán Occupancy sau khi thuật toán đã lấy được hết toàn bộ các itemset ứng viên tìm năng:

```python
def cal_occupancy_candidate(items):
    df_candidate = pd.DataFrame(columns=['Items', 'StSet'])

    list_stset_item = []

    for i in items:
        list_stset_item.append(df_stset[df_stset['Items'].apply(lambda item: i in item)]["StSet"].iloc[0])

    intersection_list = set(list_stset_item[0])
    for sublist in list_stset_item[1:]:
        intersection_list = intersection_list.intersection(sublist)

    intersection_list = sorted(intersection_list, key = lambda x: x[0])

    df_candidate = df_candidate.append({'Items': items, 'StSet': intersection_list}, ignore_index=True)
    df_candidate['Length_transaction'] = df_candidate['StSet'].apply(lambda x: [len(df[df['Tid'] == tid]['Items'].iloc[0]) for tid in x])
    df_candidate = cal_occupancy(df_candidate)
    return df_candidate
```

- Hàm mine_hoi_kitemset() là hàm chính của thuật toán FHOI, áp dụng theo mã giả và quá trình tìm hiểu tài liệu của tôi:

```python
def mine_hoi_k_itemset(threshold, hastheSameLengh, CK_minus_1, df_stset):
    CK = []
    HOIK = []

    while len(CK_minus_1) > 0:
        # P1 = sorted(CK_minus_1[0])
        P1 = CK_minus_1[0]
        for P2 in CK_minus_1:
            if is_same_equivalence_class(P1, P2):
                P = df_intersection(P1, P2, df_stset)
                P_items = P['Items'].iloc[0]
                P_stset = P['StSet'].iloc[0]
                if len(P_stset) >= threshold:
                    if hastheSameLengh is False:
                        P_ubo = cal_UBO(P)['Max_UBO'].iloc[0]
                        if P_ubo >= threshold:
                            CK.append(P_items)
                    else:
                        CK.append(P_items)

        CK_minus_1.pop(0)

    for i in CK:
        if cal_occupancy_candidate(i)['Occupancy'].iloc[0] >= threshold:
            HOIK.append(i)

    return CK, HOIK
```

- Hàm mine_depth_hois() phục vụ cho thuật toán DFHOI khi áp dụng đệ quy vào để khai thác toàn bộ itemset có thể có:

```python
def mine_depth_hois(threshold, hastheSameLengh, C1, df_stset, HOIS):
    if len(C1) == 0:
        return HOIS

    for i in range(len(C1)):
        P1 = C1[i]
        C_l = []
        for j in range(i + 1, len(C1)):
            P2 = C1[j]
            P = df_intersection(P1, P2, df_stset)
            P_items = P['Items'].iloc[0]
            P_stset = P['StSet'].iloc[0]
            if len(P_stset) >= threshold:
                if hastheSameLengh is False:
                    P_ubo = cal_UBO(P)['Max_UBO'].iloc[0]
                    if P_ubo >= threshold:
                        C_l.append(P_items)
                else:
                    C_l.append(P_items)

        mine_depth_hois(threshold, hastheSameLengh, C_l, df_stset, HOIS)

        for i in C_l:
            if cal_occupancy_candidate(i)['Occupancy'].iloc[0] >= threshold:
                HOIS.append(i)

    # return HOIS
```

- Hàm test_fhoi() nhận đầu vào là danh sách các threshold để thử nghiệm tốc độ cũng như tính toán số lượng itemset chiếm dụng cao cho thuật toán FHOI:

```python
#test FHOI
def test_fhoi(threshold_list):
    start_time = time.time()
    execution_time_list = []
    len_itemset_list = []
    HOIS_list = []
    for threshold_percent in threshold_list:
        HOIS = []
        C1 = []
        HOI1 = []
        CK_minus_1 = []

        k = 2 # loop to create 2-itemset
        threshold = threshold_percent * len_df # ex: threshold = 25% of len(database)
        start_time = time.time()

        #create candidate 1 and HOI1 itemset
        C1, HOI1 = mine_hoi_1_itemset(threshold, hastheSameLengh, df_stset)
        HOIS = HOI1
        CK_minus_1 = C1

        while CK_minus_1:
            CK, HOIK = mine_hoi_k_itemset(threshold, hastheSameLengh, CK_minus_1, df_stset)
            if HOIK:
                for i in HOIK:
                    HOIS.append(i)
                CK_minus_1 = CK
                k += 1
            else:
                continue

        for i in HOIS:
            print(i)

        end_time = time.time()
        execution_time = end_time - start_time
        print("Threshold: " + str(threshold_percent))
        print("Length HOI: " + str(len(HOIS)))
        print(f"Execution time: {execution_time} seconds")
        print("--------------------")
        len_itemset_list.append(len(HOIS))
        execution_time_list.append(execution_time)
        HOIS_list.append(HOIS)
    return len_itemset_list, execution_time_list, HOIS_list
```

- Hàm test_dfhoi() nhận đầu vào là danh sách các threshold để thử nghiệm tốc độ cũng như tính toán số lượng itemset chiếm dụng cao cho thuật toán DFHOI:

```python
#test DFHOI
def test_dfhoi(threshold_list):
    start_time = time.time()


    execution_time_list = []
    len_itemset_list = []
    HOIS_list = []
    for threshold_percent in threshold_list:
        HOIS = []
        C1 = []
        HOI1 = []

        # threshold_percent = i
        threshold = threshold_percent * len_df # ex: threshold = 25% of len(database)

        #create candidate 1 and HOI1 itemset
        C1, HOI1 = mine_hoi_1_itemset(threshold, hastheSameLengh, df_stset)
        HOIS = HOI1

        HOIS.append(mine_depth_hois(threshold, hastheSameLengh, C1, df_stset, HOIS))
        HOIS = list(filter(lambda x: x is not None, HOIS))

        for i in HOIS:
            print(i)

        end_time = time.time()
        execution_time = end_time - start_time
        print("Threshold: " + str(threshold_percent))
        print("Length HOI: " + str(len(HOIS)))
        print(f"Execution time: {execution_time} seconds")
        print("--------------------")
        len_itemset_list.append(len(HOIS))
        execution_time_list.append(execution_time)
        HOIS_list.append(HOIS)
    return len_itemset_list, execution_time_list, HOIS_list
```

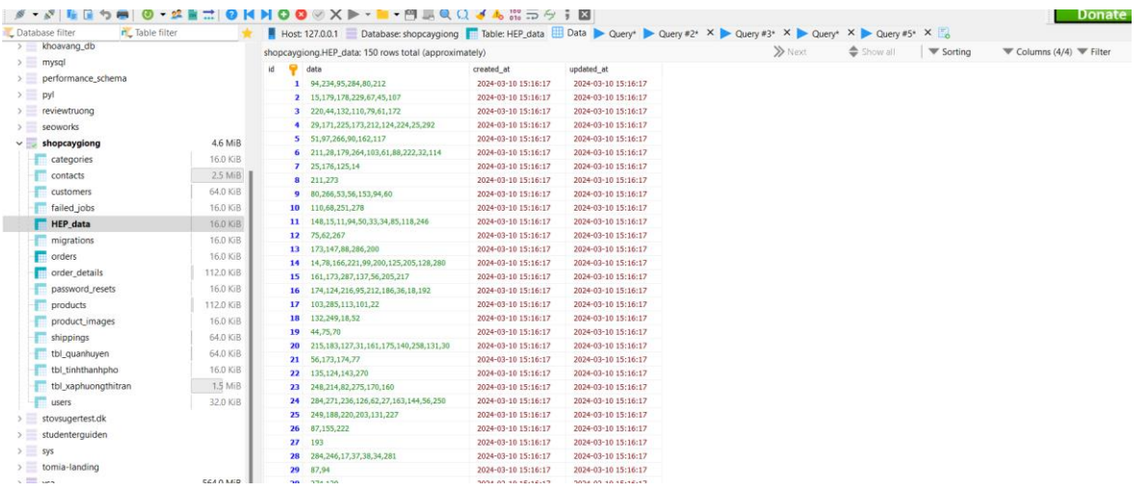
- Đoạn code sau dùng để vẽ biểu đồ mô tả với tương ứng các threshold đầu vào thì thời gian và số lượng itemset chiếm dụng cao như thế nào. Dựa trên thực nghiệm của tôi, thuật toán FHOI cho kết quả tốt hơn dù sử dụng cùng các hàm tính toán chỉ thay đổi cách triển khai thuật toán. Cả hàm để cho kết quả số lượng itemset có độ chiếm dụng cao tương tự nhau, chỉ khác về thời gian chạy của hai thuật toán:

```python
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

#fhoi
ax1.plot(threshold_list_fhoi, execution_time_fhoi_list, label='FHOI - Runtimes')
ax1.plot(threshold_list_fhoi, len_itemset_fhoi_list, label='FHOI - Itemsets')
ax1.set_title('FHOI')
ax1.set_xlabel('Threshold')
ax1.set_ylabel('Value')
ax1.legend()

#dfhoi
ax2.plot(threshold_list_dfhoi, execution_time_dfhoi_list, label='DFHOI - Runtimes')
ax2.plot(threshold_list_dfhoi, len_itemset_dfhoi_list, label='DFHOI - Itemsets')
ax2.set_title('DFHOI')
ax2.set_xlabel('Threshold')
ax2.set_ylabel('Value')
ax2.legend()

min_x = min(min(threshold_list_fhoi), min(threshold_list_dfhoi))
max_x = max(max(threshold_list_fhoi), max(threshold_list_dfhoi))
min_y = min(min(execution_time_fhoi_list), min(len_itemset_fhoi_list), min(execution_time_dfhoi_list), min(len_itemset_dfhoi_list))
max_y = max(max(execution_time_fhoi_list), max(len_itemset_fhoi_list), max(execution_time_dfhoi_list), max(len_itemset_dfhoi_list))
ax1.set_xlim(min_x, max_x)
ax1.set_ylim(min_y, max_y)
ax2.set_xlim(min_x, max_x)
ax2.set_ylim(min_y, max_y)

plt.tight_layout()
plt.show()
```

- Đoạn code xuất file kết quả để sử dụng cho demo web:

```python
import csv
demo_dfhoi = HOIS_dfhoi_list[0]
df_demo_dfhoi = pd.DataFrame({'Items': demo_dfhoi})

with open(r"E:\demo_web.csv", "w") as f:
    for row in demo_dfhoi:
        f.write("%s\n" % ','.join(str(col) for col in row))
```

Import kết quả vào DB và hiển thị khi cần thiết:



Hình 4.10 Import kết quả vào DB và hiển thị khi cần thiết

e) Phương pháp hiện thị kết gợi ý trong giỏ hàng.

Hiện thị gợi ý sẽ bao gồm 9 sản phẩm khác nhau kết hợp giữa kết quả từ thuật toán DFHOI và kết quả query theo danh mục sản phẩm như sau:

- Trường hợp 1: Các phẩm trong giỏ hàng đều mặt trong một trong những itemset của kết quả sau khi chạy thuật toán. Lấy ra một itemset có nhiều sản phẩm nhất và loại trừ đi những sản phẩm đã có trong giỏ hàng sẽ còn lại số sản phẩm phù hợp để gợi ý. Nếu số sản phẩm gợi ý trong itemset đó quá ít (trong trường hợp này mặc định là ít hơn 9) thì sẽ lấy sản phẩm theo danh mục của sản phẩm cuối cùng trong giỏ hàng để bù vào.

- Trường hợp 2: Các sản phẩm trong giỏ hàng không thuộc bất cứ itemset nào trong kết quả thuật toán. Lúc này sẽ lấy từng sản phẩm để tìm itemset liên quan và số sản phẩm tìm được qua các item của từng sản phẩm riêng lẽ sẽ là sản phẩm gợi ý cho người dùng. Nếu số sản phẩm gợi ý quá ít (trong trường hợp này mặc định là ít hơn 9) thì sẽ lấy sản phẩm theo danh mục của sản phẩm cuối cùng trong giỏ hàng để bù vào.

f) Setup source demo.

Demo sẽ bao gồm các thành phần sau:

- Source code: sử dụng framework laravel 8.5 và php version từ 7.4 trở lên.

- Database: file sql để import tạo database.

- File thuật toán sử dụng python 3.9.2.

- File csv cung cấp data mẫu.

Các bước cài đặt:

- Sử dụng máy ảo homestead hoặc yaml để cài source code trên local. Sau đó và vào file .evn để cài đặt cấu hình database, host phù hợp với môi trường server.

- Sử dụng công cụ hỗ trợ cài database như phpMyAdmin hoặc Heidisql hoặc các công cụ khác để tạo database và import database bằng file sql được cung cấp.

- Tiến hành chạy website theo host đã cài đặt trong file .env và môi trường của server.

- Cài đặt python 3.9.2 và các thư viện xử lý số cơ bản của python.

- Cài đặt VS Code làm nơi chạy thuật toán.

TÀI LIỆU THAM KHẢO

[1]: Mining high occupancy itemsets

[2]: An efficient method for mining high occupancy itemsets based on equivalence class and early pruning