MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

DANH MỤC TỪ VIẾT TẮT

KPTT	Khai phá tri thức.
KPDL	Khai phá dữ liệu.
FCP	Tập phổ biến đóng.
FCC	Khối phổ biến đóng.
RSM	Khai phá lát đại diện**.**

LỜI MỞ ĐẦU

Ngày nay, cuộc cách mạng của kỹ thuật số cho phép số hóa thông tin dễ dàng và chi phí lưu trữ thấp.Với sự phát triển của phần mềm, phần cứng và trang bị nhanh hệ thống máy tính trong kinh doanh. Số lượng dữ liệu khổng lồ được tập trung và lưu trữ trong cơ sở dữ liệu trên các thiết bị điện tử như: đĩa cứng, băng từ, đĩa quang,… Tốc độ tăng dữ liệu quá lớn . Từ đó dẫn đến kết quả là sự pha trộn của kỹ thuật thống kê vào các công cụ quản trị dữ liệu không thể phân tích đầy đủ dữ liệu rộng lớn được nữa.

Dữ liệu sau khi phục vụ cho một mục đích nào đó được lưu lại trong kho dữ liệu và theo ngày tháng khối lượng dữ liệu được lưu trữ ngày càng lớn. Trong khối lượng dữ liệu to lớn này có rất nhiều thông tin có ích mang tính tổng quát, thông tin có tính quy luật vẫn còn đang tiềm ẩn mà chúng ta chưa biết. Từ khối lượng dữ liệu rất lớn cần có những công cụ tự động rút các thông tin và kiến thức có ích. Một hướng tiếp cận có khả năng giúp các công ty khai thác các thông tin có nhiều ý nghĩa từ các tập dữ liệu lớn đó là khai phá dữ liệu (Data Mining).

Với sự bùng nổ và phát triển của công nghệ thông tin đã mang lại nhiều hiệu quả đối với khoa học cũng như các hoạt động thực tế, trong đó khai phá dữ liệu là một trong những lĩnh vực mang lại hiệu quả thiết thực cho con người. KPDL đã giúp người sử dụng thu được những tri thức hữu ích từ những cớ sở dữ liệu hoặc các kho dữ liệu khổng lồ khác. Đề tài đề cập đến các khái niệm và vấn đề cơ bản trong KPTT và KPDL, ngoài ra Đề tài còn đề cập đến một số phương pháp khai phá dữ liệu dạng đóng được áp dụng trong nhiều lĩnh vực thực tiễn.

Cấu trúc đồ án:

Chương 1 giới thiệu tổng quan về KPTT và KPDL.

Chương 2 Tìm hiểu phương pháp khai phá tập phổ biến.

Chương 3 Tìm hiểu phương pháp khai phá tập phổ biến đóng trong không gian.

Chương 4 Cài đặt chương trình thử nghiệm.

KẾT LUẬN.

TÀI LIỆU THAM KHẢO.

CHƯƠNG 1: TỔNG QUAN VỀ KPTT VÀ KPDL.

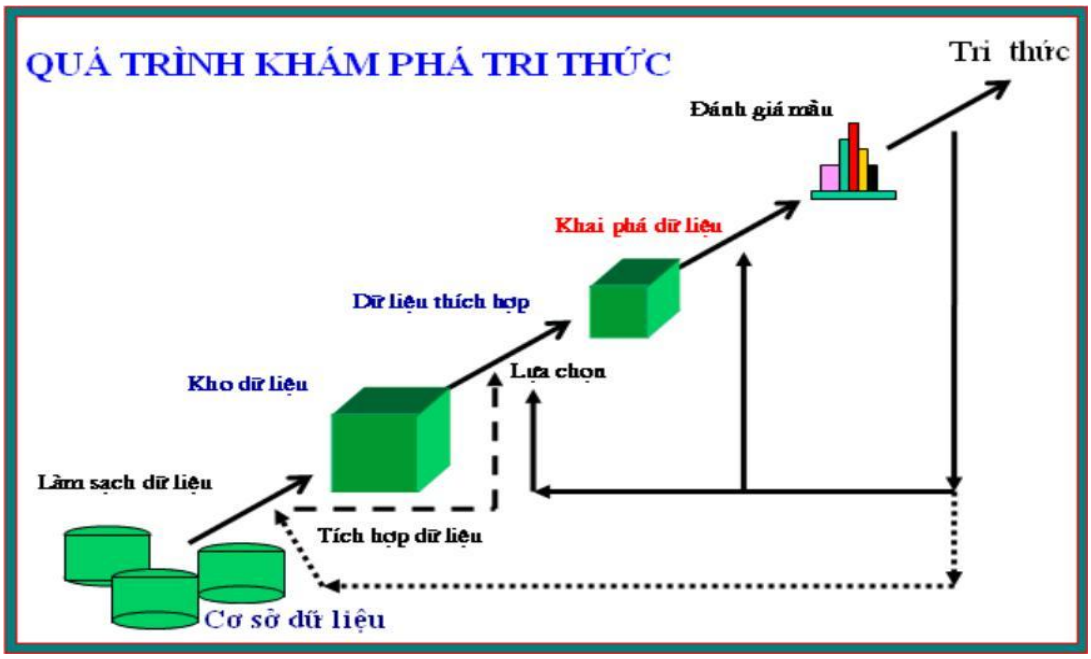
1.1 Giới thiệu chung về khai phá tri thức và khai phá dữ liệu.

- Nếu cho rằng, điện tử và truyền thông chính là bản chất của khoa học điện tử, thì dữ liệu, thông tin, và tri thức hiện đang là tiêu điểm của một lĩnh vực mới để nghiên cứu và ứng dụng, đó là khai phá tri thức và khai phá dữ liệu.
- Thông thường, chúng ta coi dữ liệu như là một chuỗi các bits, hoặc các số và các ký hiệu hay là các "đối tượng" với một ý nghĩa nào đó khi được gửi cho một chương trình dưới một dạng nhất định. Các bits thường được sử dụng để đo thông tin, và xem nó như là dữ liệu đã được loại bỏ phần tử thừa, lặp lại, và rút gọn tới mức tối thiểu để đặc trưng một cách cơ bản cho dữ liệu. Tri thức được xem như là các thông tin tích hợp, bao gồm các sự kiện và mối quan hệ giữa chúng, đã được nhận thức, khám phá, hoặc nghiên cứu. Nói cách khác, tri thức có thể được coi là dữ liệu ở mức độ cao của sự trừu tượng và tổng quát.
- Khái phá tri thức hay phát hiện tri thức trong CSDL là một quy trình nhận biết các mẫu hoặc các mô hình trong dữ liệu với các tính năng: Phân tích, tổng hợp, hợp thức, khả ích và có thể hiểu được.
- Khai phá dữ liệu là một bước trong quá trình khám phá tri thức, gồm các thuật toán khai thác dữ liệu chuyên dùng dưới một số qui định về hiệu quả tính toán chấp nhận được để tìm ra các mẫu hoặc các mô hình trong dữ liệu. Nói cách khác, mục tiêu của Khai phá dữ liệu là tìm kiếm các mẫu hoặc mô hình tồn tại trong CSDL nhưng ẩn trong khối lượng lớn dữ liệu.

1.2 Quá trình khai phá tri thức.

Bao gồm các bước sau:

- **Làm sạch dữ liệu (Data Cleaning):** Loại bỏ dữ liệu nhiễu và dữ liệu không nhất quán.
- **Tích hợp dữ liệu (Data Intergation):** Dữ liệu của nhiều nguồn có thể được tổ hợp lại.
- **Lựa chọn dữ liệu (Data Selection):** Lựa chọn những dữ liệu phù hợp với nhiệm vụ phân tích trích rút từ cơ sở dữ liệu.
- **Chuyển đổi dữ liệu (Data Transformation):** Dữ liệu được chuyển đổi hay được hợp nhất về dạng thích hợp cho việc khai phá.
- **Khai phá dữ liệu (Data Mining):** Đây là một tiến trình cốt yếu trong đó các phương pháp thông minh được áp dụng nhằm trích rút ra mẫu dữ liệu.
- **Đánh giá mẫu (Pattern Evaluation):** Dựa trên một độ đo nào đó xác định lợi ích thực sự, độ quan trọng của các mẫu biểu diễn tri thức.
- **Biểu diễn tri thức (Knowledge Presentation):** Ở giai đoạn này các kỹ thuật biểu diễn và hiển thị được sử dụng để đưa tri thức lấy ra cho người dùng.

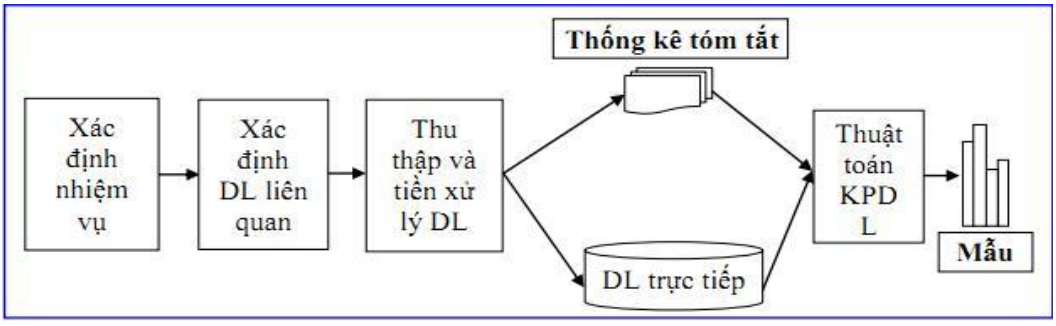


Hình 1.1: Quá trình KPTT.

1.3 Quá trình khai thác dữ liệu.

- **KPDL** là một giai đoạn quan trọng trong quá trình **KPTT**. Về bản chất, nó là giai đoạn duy nhất tìm ra được thông tin mới, thông tin tiềm ẩn có trong CSDL chủ yếu phục vụ cho mô tả và dự đoán.
- **Mô tả dữ liệu** là tổng kết hoặc diễn tả những đặc điểm chung của những thuộc tính dữ liệu trong kho dữ liệu mà con người có thể hiểu được.
- **Dự đoán** là dựa trên những dữ liệu hiện thời để dự đoán những quy luật được phát hiện từ các mối liên hệ giữa các thuộc tính của dữ liệu trên cơ sở đó chiết xuất ra các mẫu, dự đoán được những giá trị chưa biết hoặc những giá trị tương lai của các biến quan tâm.

Quá trình **KPDL** bao gồm các bước chính được thể hiện như Hình 1.2 sau:



Hình 1.2: Quá trình KPDL.

- Xác định nhiệm vụ: Xác định chính xác các vấn đề cần giải quyết.
- Xác định các dữ liệu liên quan: Dùng để xây dựng giải pháp.
- Thu thập và tiền xử lý dữ liệu: Thu thập các dữ liệu liên quan và tiền xử lý chúng sao cho thuật toán **KPDL** có thể hiểu được. Đây là một quá trình rất khó

khăn, có thể gặp phải rất nhiều các vướng mắc như: dữ liệu phải được sao ra nhiều bản (nếu được chiết xuất vào các tệp), quản lý tập các dữ liệu, phải lặp đi lặp lại nhiều lần toàn bộ quá trình (nếu mô hình dữ liệu thay đổi), vv...

- Thuật toán khai phá dữ liệu: Lựa chọn thuật toán **KPDL** và thực hiện việc PKDL để tìm được các mẫu có ý nghĩa, các mẫu này được biểu diễn dưới dạng luật kết hợp, cây quyết định... tương ứng với ý nghĩa của nó.

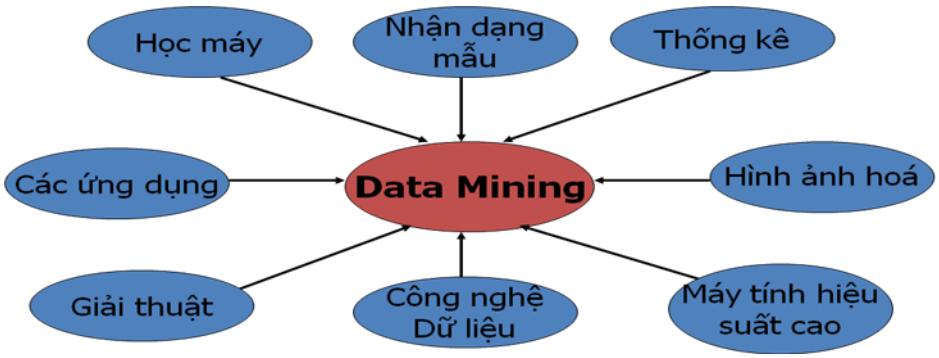
1.4 Các phương pháp khai phá dữ liệu.

Với hai mục đích khai phá dữ liệu là Mô tả và Dự đoán, người ta thường sử dụng các phương pháp sau cho khai phá dữ liệu:

- Luật kết hợp (association rules)
- Phân lớp (Classfication)
- Hồi qui (Regression)
- Trực quan hóa (Visualiztion)
- Phân cụm (Clustering)
- Tổng hợp (Summarization)
- Mô hình ràng buộc (Dependency modeling)
- Biểu diễn mô hình (Model Evaluation)
- Phân tích sự phát triển và độ lệch (Evolution and deviation analyst)
- Phương pháp tìm kiếm (Search Method)
- Tập phổ biến đóng(Frequent Closed Patterns)

Có nhiều phương pháp khai phá dữ liệu được nghiên cứu ở trên, trong đó có ba phương pháp được các nhà nghiên cứu sử dụng nhiều nhất đó là: Luật kết hợp, Phân lớp dữ liệu và Phân cụm dữ liệu.

1.5 Các lĩnh vực ứng dụng thực tiễn của khai phá dữ liệu.



Hình 1.3: Các lĩnh vực ứng dụng KPDL.

1.6 Các hướng tiếp cận trong khai phá dữ liệu.

Các hướng tiếp cận của **KPDL** có thể được phân chia theo chức năng hay lớp các bài toán khác nhau. Sau đây là một số hướng tiếp cận chính.

- **Phân lớp và dự đoán (classification & prediction):** xếp một đối tượng vào một trong những lớp đã biết trước. Ví dụ: phân lớp vùng địa lý theo dữ liệu thời tiết. Hướng tiếp cận này thường sử dụng một số kỹ thuật của machine learning

như cây quyết định (decision tree), mạng nơ ron nhân tạo (neural network), .v.v. Phân lớp còn được gọi là học có giám sát (học có thầy supervised learning).

- **Luật kết hợp (association rules):** là dạng luật biểu diễn tri thứ ở dạng khá đơn giản. Ví dụ: "60 % nam giới vào siêu thị nếu mua bia thì có tới 80% trong số họ sẽ mua thêm thịt bò khô". Luật kết hợp được ứng dụng nhiều trong lĩnh vực kinh doanh, y học, tin-sinh, tài chính & thị trường chứng khoán, .v.v.
- **Khai phá chuỗi theo thời gian (sequential/temporal patterns):** tương tự như khai phá luật kết hợp nhưng có thêm tính thứ tự và tính thời gian. Hướng tiếp cận này được ứng dụng nhiều trong lĩnh vực tài chính và thị trường chứng khoán vì nó có tính dự báo cao.
- **Phân cụm (clustering/segmentation):** xếp các đối tượng theo từng cụm (số lượng cũng như tên của cụm chưa được biết trước. Phân cụm còn được gọi là học không giám sát (học không có thầy – unsupervised learning).
- **Mô tả khái niệm (concept description & summarization):** thiên về mô tả, tổng hợp và tóm tắt khái niệm. Ví dụ: tóm tắt văn bản.
- **Khai phá tập phổ biến (mining frequent pattern):** thiên về mô tả, tổng hợp và tóm tắt khái niệm. Ví dụ: tóm tắt văn bản.

1.7 Phân loại các hệ khai phá dữ liệu.

- **KPDL** là một công nghệ tri thức liên quan đến nhiều lĩnh vực nghiên cứu khác nhau như CSDL, kỹ thuật máy học (machine learning), giải thuật, trực quan hóa (visualization), .v.v. Chúng ta có thể phân loại các hệ thống **KPDL** dựa trên các tiêu chí khác nhau.
- Phân loại dựa trên kiểu dữ liệu được khai phá: CSDL quan hệ (relational database), kho dữ liệu (data warehouse), CSDL giao dịch (transactional database), CSDL hướng đối tượng, CSDL không gian (spatial database), CSDL đa phương tiện (multimedia database), CSDL Text và WWW, .v.v.
- Phân loại dựa trên dạng tri thức được khám phá: tóm tắt và mô tả (summarization & description), luật kết hợp (association rules), phân lớp (classification), phân cụm (clustering), khai phá chuỗi (sequential mining), .v.v.
- Phân loại dựa trên kỹ thuật được áp dụng: hướng CSDL (database-oriented), phân tích trực tuyến (OnLine Analytical Processing – OLAP), machine learning (cây quyết định, mạng nơ ron nhân tạo, k-min, giải thuật di truyền, máy vectơ hỗ trợ - SVM, tập thô, tập mờ, .v.v.), trực quan hóa (visualization), .v.v.
- Phân loại dựa trên lĩnh vực được áp dụng: kinh doanh bán lẻ (retail), truyền thông (telecommunication), tin-sinh (bio-informatics), y học (medical treatment), tài chính & thị trường chứng khoán (finance & stock market), Web mining, .v.v.

1.8 Các thách thức - khó khăn trong KPTT và KPDL.

KPTT và **KPDL** liên quan đến nhiều ngành, nhiều lĩnh vực trong thực tế, vì vậy các thách thức và khó khăn ngày càng nhiều, càng lớn hơn. Sau đây là một số các thách thức và khó khăn cần được quan tâm:

- Các cơ sở dữ liệu lớn, các tập dữ liệu cần xử lý có kích thước cực lớn, Trong thực tế, kích thước của các tập dữ liệu thường ở mức tera-byte (hàng ngàn giga-byte).
- Mức độ nhiễu cao hoặc dữ liệu bị thiếu.
- Số chiều lớn.
- Thay đổi dữ liệu và tri thức có thể làm cho các mẫu đã phát hiện không còn phù hợp.
- Quan hệ giữa các trường phức tạp.

CHƯƠNG 2: PHƯƠNG PHÁP KHAI PHÁ TẬP PHỔ BIẾN.

2.1 Giới thiệu.

- Hiện nay, các cơ sở dữ liệu, các tập dữ liệu cần xử lý có kích thước cực lớn. Trong thực tế, kích thước của các tập dữ liệu thường ở mức tera-byte (hàng ngàn gigabyte). Các tập dữ liệu có mức độ nhiễu cao hoặc dữ liệu bị thiếu, số chiều lớn, quan hệ giữa các trường phức tạp dẫn đến việc các hướng tiếp cận phổ biến không còn hiệu quả và chính xác. Chính vì vậy phương pháp khai phá tập phổ biến đã được ra đời nhằm đáp ứng các nhu cầu trên.
- Tập phổ biến là tập các tập mục, chuỗi con, hoặc các cấu trúc nhỏ mà xuất hiện phổ biến trong bộ dữ liệu.
- Khai phá tập phổ biến đã được nghiên cứu và sử dụng rộng rãi trong việc khai phá dữ liệu, với nhiều thuật toán đã được đề xuất và thực hiện như: Apiorio, clospan, PrefixSpan…
- Chúng ta sẽ đi vào tìm hiểu một số thuật toán cơ bản trong khai phá tập phổ biến.

2.2 Giới thiệu một số thuật toán khai phá tập phổ biến.

2.2.1 Thuật toán Apriori.

Apriori là một thuật toán tốt để khai phá tập phổ biến cho luật kết hợp kiểu Boolean.

Apriori sử dụng một phương pháp tìm kiếm thông minh, với k-itemsets (một tập có chứa các mục k) được sử dụng để khai phá (k +1)-itemsets.

- Đầu tiên, tập mục phổ biến 1-itemsets được tìm thấy, ký hiệu là L_1. L_1 được sử dụng để tìm L_2 (tập phổ biến 2-itemsets). Tiếp tục như vậy, để tìm L_3, và tiếp tục như vậy, cho đến khi không còn tập phổ biến k-itemsets có thể được tìm thấy.
- Tìm tất cả các tập phổ biến: Các tập mục sẽ phổ biến khi độ hỗ trợ S ít nhất bằng với min_sup được xác định trước. Tiếp đó tạo ra các luật kết hợp mạnh từ các tập phổ biến. Những luật này phải phải thỏa mãn cả 2 ngưỡng min_sup và min_conf (luật kết hợp mạnh).

❖ **Có thể tóm lược thuật toán Apriori như sau:**

- **Bước 1:** Quét tập dữ liệu để có được độ hỗ trợ S của các tập phổ biến, so sánh S với min_sup, và lấy được tập phổ biến 1-itemsets L_1.
- **Bước 2:** Nhóm các L_{k-1} lại để tạo ra tập các ứng viên k-itemsets. Và sử dụng các tính chất của Apriori để lược bớt những tập k-itemsets không phải phổ biến từ tập ứng viên.
- **Bước 3:** Tiếp tục quét tập dữ liệu để có được độ hỗ trợ của mỗi tập ứng viên k-itemsets, so sánh S với min_sup, và lấy ra các tập phổ biến k-itemsets L_k thỏa mãn.
- **Bước 4:** lặp lại bước 2 và 3 cho đến khi tập ứng viên là rỗng.
- **Bước 5:** Với mỗi tập phổ biến l tạo ra tất cả các tập con không rỗng của l.

- **Bước 6:** Với mỗi tập con không rỗng của l, tạo ra các luật "s ➔ (l-s)" nếu độ tin cậy C của luật "s ➔ (l-s)" bằng với độ hỗ trợ S của l trên độ hỗ trợ S của s (kí hiệu min_conf).

❖ **Tính chất Priori[2]:** rút gọn không gian tìm kiếm nhằm tránh trường hợp mỗi l_k phải quét toàn bộ dữ liệu 1 lần.

- Nếu tập mục I không thỏa mãn ngưỡng tối thiểu min_sup thì I không phải là phổ biến, P(I) < min_sup.
- Nếu I không là tập mục phổ biến thì một mục A được thêm vào tập mục I, khi đó kết quả tập mục I ∪ A cũng không là tập phổ biến, P(I ∪ A) < min_sup.

- **Ví dụ:** (Hình 2.1) Minh họa cho thuật toán Apriori.

Example 2

Problem data

An example with a transactional data D contents a list of 5 transactions in a supermarket.

TID	List of items (item_IDs)
1	Beer(I1), Diaper(I2), Baby Powder(I3), Bread(I4), Umbrella(I5)
2	Diaper(I2), Baby Powder(I3)
3	Beer(I1), Diaper(I2), Milk(I6)
4	Diaper(I2), Beer(I1), Detergent(I7)
5	Beer(I1), Milk(I6), Coca Cola (I8)

Solution Procedure

Step 1 ***min_sup*** **= 40% (2/5)**

C1 → **L1**

Item_ID	Item	Support
I1	Beer	4/5
I2	Diaper	4/5
I3	Baby powder	2/5
~~I4~~	~~Bread~~	~~1/5~~
~~I5~~	~~Umbrella~~	~~1/5~~
I6	Milk	2/5
~~I7~~	~~Detergernt~~	~~1/5~~
~~I8~~	~~Coca cola~~	~~1/5~~

Item_ID	Item	Support
I1	Beer	4/5
I2	Diaper	4/5
I3	Baby	2/5
I6	Milk	2/5

$$support(A \Rightarrow B) = \frac{\#\_tuples\_containing\_both\_A\_and\_B}{total\_\#\_of\_tuples}$$

Solution Procedure

Step 2

C2

Item_ID	Item	Support
{I1, I2}	Beer, Diaper	3/5
~~{I1, I3}~~	~~Beer, Baby powder~~	~~1/5~~
{I1, I6}	Beer, Milk	2/5
{I2, I3}	Diaper, Baby powder	2/5
~~{I2, I6}~~	~~Diaper, Milk~~	~~1/5~~
~~{I3, I6}~~	~~Baby powder, Milk~~	~~0~~

Step 3

L2

Item_ID	Item	Support
{I1, I2}	Beer, Diaper	3/5
{I1, I6}	Beer, Milk	2/5
{I2, I3}	Diaper, Baby powder	2/5

Solution Procedure

Step 4: L2 is not Null, so repeat Step2

Item_ID	Item
{I1, I2, I3}	Beer, Diaper, Baby powder
{I1, I2, I6}	Beer, Diaper, Milk
{I1, I3, I6}	Beer, Baby powder, Milk
{I2, I3, I6}	Diaper, Baby powder, Milk

C3 =Null

Solution Procedure

Step 5

min_sup=40% min_conf=70%

Item_ID	Item	Support(A B)	Support A	Confidence
I1 I2	Beer Diaper	60%	80%	75%
~~I1 I6~~	~~Beer Milk~~	~~40%~~	~~80%~~	~~50%~~
~~I2 I3~~	~~Diaper Baby powder~~	~~40%~~	~~80%~~	~~50%~~
I2 I1	Diaper Beer	60%	80%	75%
I6 I1	Milk Beer	40%	40%	100%
I3 I2	Baby powder Diaper	40%	40%	100%

$$support(A \Rightarrow B) = \frac{\#\_tuples\_containing\_both\_A\_and\_B}{total\_\#\_of\_tuples}$$

$$confidence(A \Rightarrow B) = \frac{\#\_tuples\_containing\_both\_A\_and\_B}{\#\_tuples\_containing\_A}$$

Solution Procedure

TID	List of items (item_IDs)
1	Beer(I1), Diaper(I2), Baby Powder(I3), Bread(I4), Umbrella(I5)
2	Diaper(I2), Baby Powder(I3)
3	Beer(I1), Diaper(I2), Milk(I6)
4	Diaper(I2), Beer(I1), Detergent(I7)
5	Beer(I1), Milk(I6), Coca Cola (I8)

Item_ID	Item	Support(A B)	Support A	Confidence
I1 I2	Beer Diaper	60%	80%	75%
~~I1 I6~~	~~Beer Milk~~	~~40%~~	~~80%~~	~~50%~~
~~I2 I3~~	~~Diaper Baby powder~~	~~40%~~	~~80%~~	~~50%~~
I2 I1	Diaper Beer	60%	80%	75%
I6 I1	Milk Beer	40%	40%	100%
I3 I2	Baby powder Diaper	40%	40%	100%

Hình 2.1: Ví dụ Apriori.

2.2.2 Thuật toán Freespan.

Thuật toán **Apriori** vẫn tồn tại một số vấn đề khi bộ dữ liệu là lớn hoặc chuỗi mẫu khai phá được dài hoặc lớn. **Freespan** là thuật toán được cải tiến nhằm giải quyết các vấn đề trên vì vậy nó có tính hiệu quả cao hơn so với **Apriori**

Thuật toán **Freespan** sử dụng các mục phổ biến để đệ quy chuỗi dữ liệu thành các chuỗi dữ liệu nhỏ hơn.

Khai phá tập phổ biến sử dụng các chuỗi dữ liệu nhằm giới hạn việc tìm kiếm và sự gia tăng các chuỗi con.

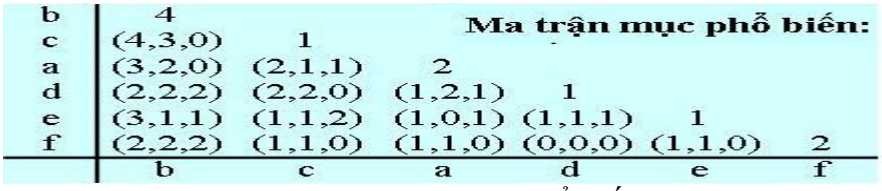
- Thuật toán **Freespan[3]**.

Tư tưởng thuật toán Freespan là cho tập các chuỗi tìm tất cả các chuỗi phổ biến con. Bằng cách đệ quy các chuỗi dữ liệu thành các chuỗi dữ liệu nhỏ hơn dựa trên các tập mẫu phổ biến. Khai phá các chuỗi dữ liệu để tìm ra các tập mẫu của nó.

- Quét dữ liệu, tìm các mục phổ biến từ tập dữ liệu danh sách mục thường xuyên với độ hỗ trợ giảm dần gọi là **f_list(danh sách mục thường xuyên).** Tất cả các chuỗi mẫu đều có thể chia nhỏ thành một vài chuỗi con không trùng lặp.
- (1)Xây dựng một ma trận mục phổ biến mỗi lần quét dữ liệu. Một ma trận mục phổ biến là một ma trận hình tam giác F[j,k] trong đó 1<=j<=m và 1<=k<=j, m là số mục phổ biến. F[j,j] chỉ có một bộ đếm ghi lại số lần xuất hiện của chuỗi <jj>.

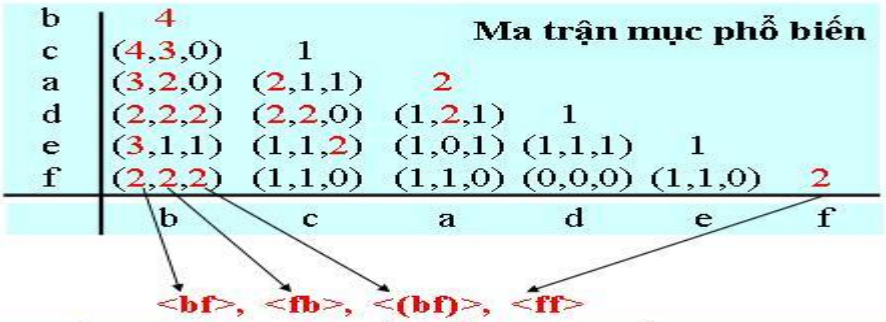
F[j,k] có 3 bộ đếm(A,B,C): A ghi lại mẫu <jk>, B ghi lại mẫu <k,j>, C ghi lại mẫu <(jk)> (Hình 2.2) Cho ngưỡng min_support = 2, **f_list**: <b:5, c:4, a:3, d:3, e:3, f:2>

Sequence_id	Sequence	Item pattern
10	<(bd) cd (ac)>	{a,b,c,d}
20	<(bf) (ce) b (fg)>	{b,c,e,f,g}
30	<(ah) (bf) abf>	{a,b,f,h}
40	<(be) (ce) d>	{b,c,d,e}
50	<a (bd) bcb (ade)>	{a,b,c,d,e}



Hình 2.2: Ma trận mục phổ biến.

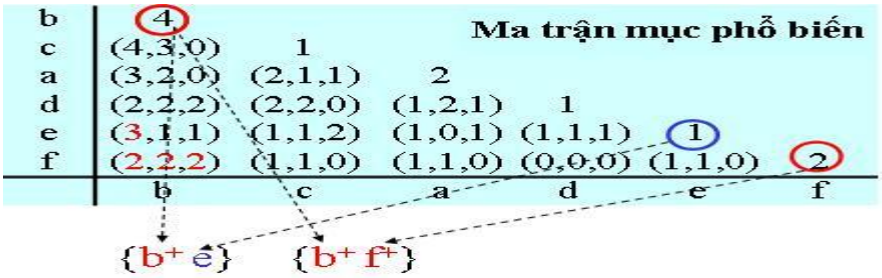
- (2)Tạo ra chuỗi các mẫu có độ dài bằng 2. Với mỗi bộ đếm , nếu giá trị trong bộ đếm lớn hơn hoặc bằng min_sup thì thu được chuỗi mẫu tương ứng (Hình 2.3).



Item	Chuỗi mẫu độ dài bằng 2
f	<bf>:2, <fb>:2, <(bf)>:2, <ff>:2
e	<be>:3, <(ce)>:2
d	<bd>:2, <db>:2, <(bd)>:2, <cd>:2,<dc>:2, <da>:2
a	<ba>:3, <ab>:2, <ca>:2, <aa>:2
c	<bc>:4, <cb>:3
b	<bb>:4

Hình 2.3: Chuỗi mẫu độ dài bằng 2.

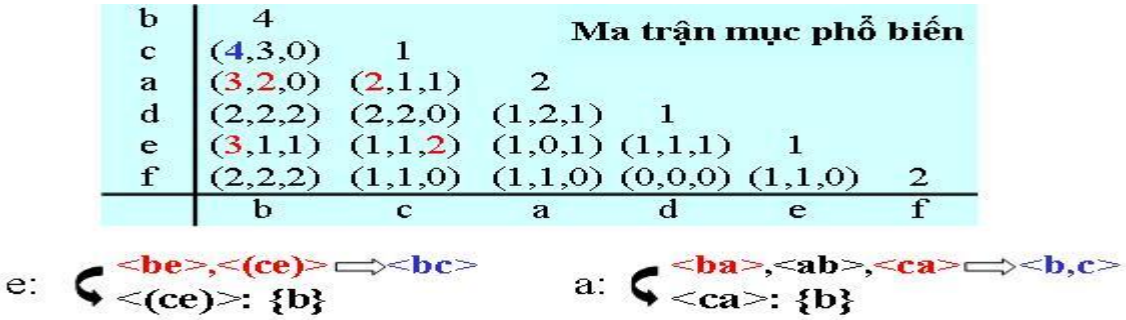
- (3)chú thích trên các **tập item-repeating**.
  - Cho dòng j: nếu f[j,j] >= min_sup thì tạo ra <jj+>.
  - Cho một cột i khác j, nếu f[i,i] >= min_sup thì tạo ra i+.Nếu f[j,j] >= min_sup thì tạo ra j+.
  - Nếu chỉ một trong 3 bộ đếm của f[i,j] là phổ biến, chuỗi được sử dụng như ghi chú và ngược lại thiết lập ban đầu được sử dụng (Hình 2.4).



Item	Repeating items
f	{b+f+}
e	<b+e>
d	{b+d}, <da+>
a	<aa+>, {a+b+}, <ca+>
c	{b+c}
b	<bb+>

Hình 2.4: Item-repeating.

- (4)**project database**. Cho ròng j: Với mỗi i<j, nếu f[i,j], f[j,k] và f[i,k](k<i) có thể hình thành 3 mẫu, k phải được thêm vào tập cột i-projected. Nếu có sự lựa chọn giữa chuỗi và tập thì chuỗi được ưu tiên (Hình 2.5).



Item	Projected DBs
f	Ø
e	<(ce)> :{b}
d	<da>:{b,c}, <(cd)>:{b}
a	<ca>:{b}
c	Ø
b	Ø

Hình 2.5: Project database.

- (5)Quét dữ liệu để tạo ra các **item-repeating** và **project database** (Hình 2.6).

annotation	<(ce)>:{b}	<da>:{b,c}
projected DB	<b(ce)b>,<b(ce)>	<(bd)cb(ac)>, <(bd)bcba>
sequential patterns	<b(ce)>:2	<(bd)a>:2, <dca>:2, <dba>:2, <(bd)ca>:2, <(bd)ba>:2, <dcba>:2, <(bd)cba>:2

{cd}:{b}	<ca>:{b}
<(bd)cbc>,<bcd>,<(bd)bcbd>	<bcba>, <bbcba>
<bcd>:2, <(bd)c>:2, <dcb>:2, <(bd)cd>:2, <(bd)bc>:2	<bca>:2, <cba>:2, <bcba>:2

Hình 2.6: Các chuỗi mẫu.

2.3 Tóm tắt.

Chúng ta đã vừa tìm hiểu chung về khai phá tập phổ biến. Hai thuật toán được giới thiệu ở trên có thể hoạt động rất hiệu quả với bộ dữ liệu nhỏ. Nhưng đối với những bộ dữ liệu lớn thì còn gặp nhiều hạn chễ. Ví dự như thuật toán Apriori:

- Nó tạo ra một tập ứng viên lớn: với 1000 chuỗi phổ biến độ dài bằng 1 tạo ra

$$1000 * 1000 + \frac{1000 * 999}{2} = 1\,499\,500$$

ứng viên độ dài bằng 2.

- Điều đó dẫn đến quá nhiều lần quét tập dữ liệu trong quá trình khai phá.

Gặp phải khó khăn khi khai phá chuỗi mẫu phổ biến dài tính theo cấp số nhân của số lượng ứng viên. Một chuỗi phổ biến độ dài 100 cần 10^{30} chuỗi ứng viên.

$$\sum_{i=1}^{100} \binom{100}{i} = 2^{100} - 1 \approx 10^{30}$$

Vì tất cả những hạn chễ ở các thuật toán trên. Chúng sẽ tìm hiểu một số thuật toán khai phá tập phổ biến mới ở chương tiếp theo.

CHƯƠNG 3: TÌM HIỂU PHƯƠNG PHÁP KHAI PHÁ TẬP PHỔ BIẾN ĐÓNG TRONG KHÔNG GIAN.

Khai phá tập phổ biến đóng đã được đề xuất để xác định tất cả các tập mục, tập thuộc tính mà thường xuyên xuất hiện trong các bản ghi của bộ dữ liệu. Số lượng các FCPs nhỏ hơn rất nhiều so với số lượng tập thường xuyên được khai phá ở chương 2 do vậy các thuật toán khai phá tập phổ biến đóng thường nhanh và hiệu quả cao hơn các thuật toán khai phá tập phổ biến. Một vài thuật toán khai phá FCP hiệu quả đã được đề xuất như: A-Close sử dụng phương pháp tìm kiếm breadth-first để tìm các FCPs, Closet và Closet++ sử dụng cây tập mẫu phổ biến để nén các lát dữ liệu, ngoài ra còn có rất nhiều các thuật toán khác như MAFIA, CHARM, D-Miner…

Dưới đây chúng ta sẽ đi vào tìm hiểu các phương pháp khai phá tập phổ biến đóng trong không gian 2 chiều, 3 chiều.

3.1 Phương pháp khai phá tập phổ biến đóng trong không gian 2 chiều.

3.1.1 Tổng quan.

Đầu tiên, tôi xin trình bày Khung cho phép chúng ta khai phá FCPS từ bộ dữ liệu dày đặc một cách hiệu quả và cải tiến. Khung này bao gồm hai phần.

Phần đầu tiên, không gian khai thác được phân chia thành một số không gian con nhỏ, như vậy (a) mỗi không gian con có thể được khai thác độc lập, và (b) tập hợp các FCPS từ tất cả các không gian con là một tập hợp các FCPS thu được từ không gian ban đầu.

Phần thứ hai, từng không gian con được khai thác độc lập để trả lại FCPS. Nhiệm vụ quan trọng trong giai đoạn này là để tỉa đi các FCPS dự thừa (những cái mà có thể được tạo ra trong không gian con khác) và giảm sai sót (những cái là FCPS trong không gian con, nhưng không phải FCPS trong không gian ban đầu).

Một khung như vậy có hai lợi thế quan trọng. Đầu tiên, là các không gian con có thể được khai phá độc lập, chúng có thể trả đáp án cho người sử dụng mà không cần đợi tất cả các không gian con xử lý hoàn tất. Điều này có nghĩa người dùng có thể thu được đáp án trong thời gian ngắn, và không còn bị tràn ngập bởi tất cả các đáp án cùng một lúc. Thứ hai, chương trình dễ dàng làm việc song song mà không cần quá trình đồng bộ hóa - các không gian con có thể được khai phá độc lập, đồng thời song song cùng một lúc.

Dựa vào Khung này, chúng ta đề xuất hai thuật toán, C-Miner và B-Miner. Hai thuật toán này có hai sự khác biệt. Trước tiên, phương pháp phân vùng không gian khác nhau : C-Miner phân vùng không gian khai phá dựa trên việc liệt kê các dòng rút gọn trong khi B-Miner phân vùng không gian dựa trên phép chiếu các dòng cơ bản. Thứ hai, bởi vì các phương pháp phân vùng khác nhau, cho nên chiến lược rút gọn được sử dụng trong hai thuật toán cũng khác nhau.

3.1.2 Sự chuẩn bị.

Đầu tiên chúng ta phải xác định một số khái niệm mà chúng ta sẽ sử dụng trong suốt chương này, và sau đó cung cấp một số mô tả vấn đề.

Cho R = {r_1,r_2….r_n} ký hiệu tập các hàng, và C = {c_1,c_2,…c_n} ký hiệu tập các cột. Trong bộ dữ liệu, mỗi dòng r_i chứa một tập các cột, và mỗi cột c_j chứa một tập các dòng. Trong chương này, chúng ta lấy một bộ dữ liệu nhị phân O = m * n, trong đó các ô $O_{i,j}$ tương ứng với các mối quan hệ giữa dòng i và cột j; a là một giá trị đúng (giá trị bằng "1") nếu i và j có mối quan hệ, và ngược lại a là một giá trị sai(giá trị bằng "0").

Trong **Bảng 3.1**, r_3 chứa c_2 và c_6, ký hiệu là C (r_3) ={c_2; c_6}; và c_7 được chứa r_5 và r_6, ký hiệu là R (c_7) ={r_5; r_6}.

Bảng 3.1: Ví dụ tập dữ liệu (ma trận O).

R/C	c_1	c_2	c_3	c_4	c_5	c_6	c_7
r_1	1	0	0	0	1	1	0
r_2	1	1	0	0	0	1	0
r_3	0	1	0	0	0	1	0
r_4	1	1	1	1	1	1	0
r_5	1	1	0	1	0	0	1
r_6	1	0	1	1	0	0	1

✓ **Định nghĩa 3.1: Độ hỗ trợ cột R(C'):** Cho một tập hợp các cột C' ϵ C, tập dòng lớn nhất mà chứa C' được định nghĩa là độ hỗ trợ cột R (C') ϵ R.

Bảng 3.1, cho C'={c_1, c_4}, khi đó R(C') = {r_4, r_5, r_6} , {r_4, r_5, r_6} có chứa c1 và c4, và không có dòng nào khác chứa cả hai cột.

✓ **Định nghĩa 3.2: Độ hỗ trợ dòng C(R'):** Cho một tập hợp các hàng R' ϵ R, tập cột lớn nhất mà chứa R' được định nghĩa là độ hỗ trợ dòng C (R') ϵ C.

Bảng 3.1, cho R' = {r_1, r_2}, khi đó C(R') = {c_1, c_6}, {c_1, c_6} có chứa r_1 và r_6, và không có cột nào khác chứa cả hai dòng.

✓ **Định nghĩa 3.3: Độ hỗ trợ |R(C')|:** Cho tập cột C', số dòng trong bộ dữ liệu mà chứa C' được định nghĩa là độ hỗ trợ của C', ký hiệu là |R (C') |.

✓ **Định nghĩa 3.4: Tập đóng (CP):** Một tập cột C' ϵ C được gọi là tập đóng nếu không tồn tại C'' sao cho C' ϵ C'' và |R(C'')| = |R(C')|.

✓ **Định nghĩa 3.5: Tập phổ biến đóng (FCP):** Một tập cột C' ϵ C được gọi là tập phổ biến đóng nếu (1) |R(C')|, độ hỗ trợ của C', cao hơn một ngưỡng hỗ trợ tối thiểu; và (2) C' là một tập đóng.

Ví dụ, cho minsup = 1, tập cột {c_1; c_5,c_6} sẽ là một tập phổ biến đóng trong **Bảng 3.1** vì nó xảy ra hai lần nhiều hơn so với ngưỡng minsup. Tuy nhiên, {c_2; c3} không phải tập phổ biến đóng ở chỗ nó có một tập {c_1; c_2; c_3} và |R({c_1; c_2; c_3}) = |R({c_2; c_3}) |.

✓ **Định nghĩa 3.6: Mật độ dữ liệu:** Mật độ dữ liệu (ký hiệu là mật độ) được định nghĩa là tỷ lệ phần trăm của các ô có chứa giá trị 1 trong các bộ dữ liệu nhị phân.

✓ **Định nghĩa 3.7: Độ dài mẫu:** Cho một tập phổ biến đóng, số lượng các cột chứa trong tập phổ biến đóng được gọi là độ dài mẫu, ký hiệu là Len.

Ví dụ, Với tập phổ biến đóng $\{c_1; c_5; c_6\}$, chiều dài mẫu Len = 3.

✓ **Miêu tả vấn đề (khai phá FCP):** Cho một tập dữ liệu O, vấn đề của chúng ta là khai phá tất cả FCPS đối với độ hỗ trợ người sử dụng gọi là minsup và chiều dài mẫu người sử dụng gọi là minlen.

Trước khi kết thúc phần này, chúng tôi muốn chỉ ra rằng chúng ta sẽ cần để ý đến độ hỗ trợ cột của một FCP. Như vậy, để thuận tiện, chúng tôi cũng sẽ đề cập đến các ma trận con R(FCP) x FCP như FCP.

3.1.3 Tiến bộ của phương pháp khai phá tập phổ biến đóng.

Đầu tiên, chúng ta trình bày khung cơ bản cho khai phá tập phổ biến đóng. Sau đó trình bày hai thuật toán C-Miner và B-Miner, dựa trên khung này. Cuối cùng, chúng ta sẽ trình bày làm thế nào khung này có thể dễ dàng thích nghi với việc khai phá tập phổ biến đóng song song.

3.1.4 Khung cải tiến cho khai phá tập phổ biến đóng.

Cho O là bộ dữ liệu gốc (ma trận) được khai phá. Chúng ta sẽ đề cập đến O như là một không gian, trong trường hợp này O chính là không gian khai thác ban đầu. Cho MineFCP(M) ký hiệu các tập phổ biến đóng khai phá từ không gian M.

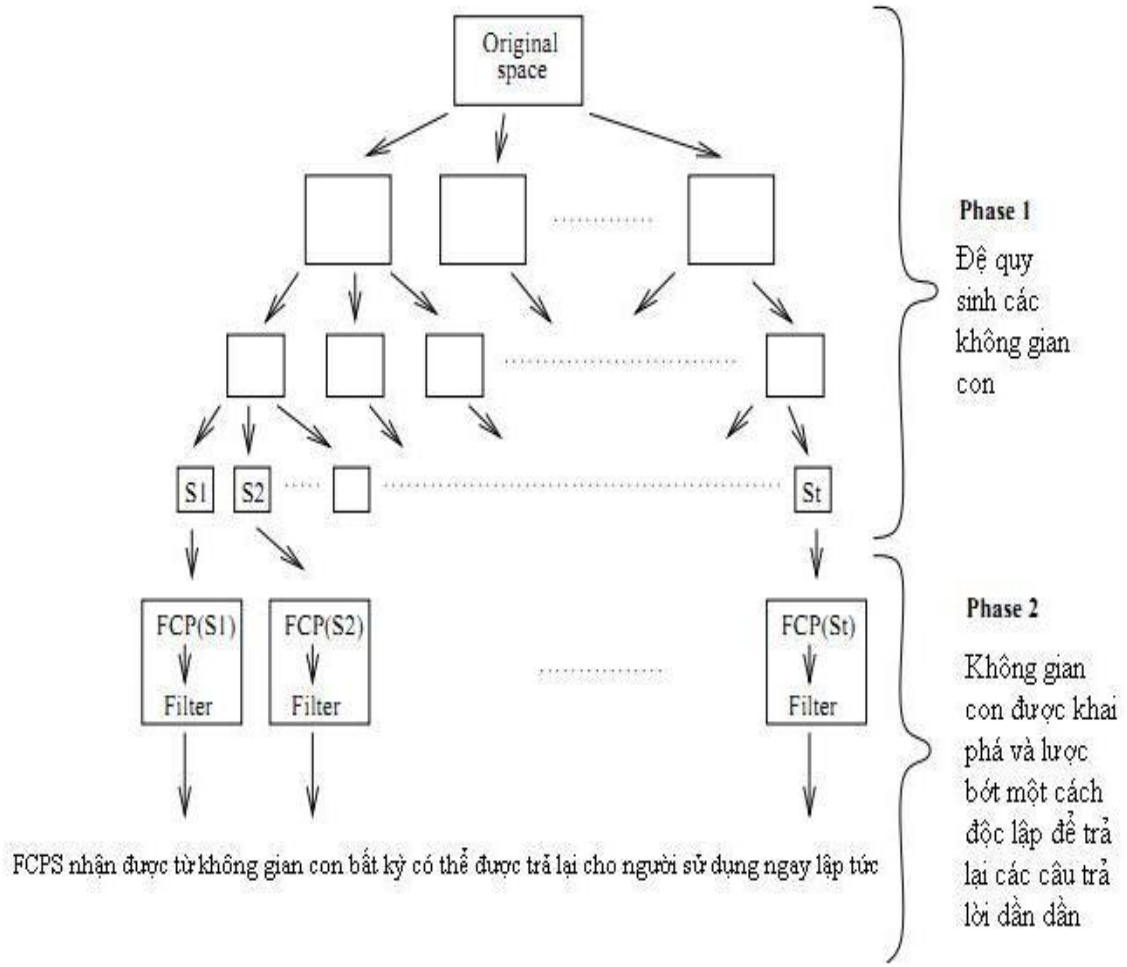
Ý tưởng cơ bản của khung này, như minh họa trong hình 3.1, bao gồm hai giai đoạn: giai đoạn hình thành các không gian con, và giai đoạn khai phá các không gian con.

Trong giai đoạn đầu, giai đoạn hình thành các không gian con, không gian O được đệ quy chia thành các không gian con S1,S2….St, trong đó $t > 1$, như vậy:

$$MineFCP(O) \in \bigcup_{i=1}^{t} MineFCP(S_i)$$

Nói cách khác, không gian ban đầu được phân chia thành nhiều không gian con, như vậy tập hợp tất cả các tập phổ biến đóng được khai phá từ tất cả không gian con có thể là câu trả lời thực tế trong không gian ban đầu. Tính chất này cho phép chúng ta khai phá các không gian con khác nhau độc lập và đồng thời.

Bằng cách này, những đáp án thu được từ không gian con có thể được trả lại ngay cho người sử dụng. Hơn nữa, vì không gian con là nhỏ hơn so với không gian ban đầu nên nó có thể được khai phá hiệu quả hơn. Như đã đề cập, khả năng khai phá không gian con là độc lập do vậy quá trình khai phá song song được thực hiện dễ dàng hơn.



Hình 3.1: Khung khai phá.

Trong giai đoạn 2, Giai đoạn khai phá các không gian con, mỗi không gian con được khai phá ta được các FCPs độc lập. Tuy nhiên, các FCPs được khai phá từ không gian con có thể không phải là đáp án. Có 2 trường hợp sảy ra:(a)FCPs khai phá từ một không gian con có thể là sai. Một tập là FCP của không gian con nhưng không phải FCP toàn cục. (b)FCPs khai phá từ không gian con là dư thừa, ví dụ: một FCP có thể khai phá từ nhiều không gian con khác.

$$\text{MineFCP}(Si) \cap \text{MineFCP}(Sj) \neq \emptyset.$$

Như vậy, cơ chế cắt tỉa sau phải được triển khai để loại bỏ các FCPS không đóng toàn cục hoặc dư thừa để chỉ có các kết quả đúng được trả lại. **Hình 3.1**, ngay sau khi kết quả được tạo ra từ các không gian con, chúng có thể được trả lại ngay cho người sử dụng.

Trong hai phần kế tiếp chúng ta sẽ trình bày hai thuật toán, C-Miner và B-Miner, dựa trên khung này. Hai phương án khác nhau trong cách phân vùng không gian ban đầu, và các chiến lược cắt tỉa.

3.1.5 Thuật toán C-Miner.

Thuật toán C-Miner**[1]** được chia làm 2 giai đoạn:

➢ **Giai đoạn phân vùng không gian khai phá**

Giai đoạn phân vùng không gian khai phá của C-Miner bao gồm 4 bước:

Bước 1: Các dòng tương đồng nhau trong dữ liệu gốc O được nhóm lại bằng một phương pháp phân cụm. Bất kỳ thuật toán phân cụm nào cũng có thể được sử dụng ở đây. Trong đề tài này chúng ta sử dụng phương pháp phân cụm CLUTO. Trong thuật toán CLUTO số lượng các cụm k là tham số do người dùng chỉ định. Giải pháp phân cụm thành k cụm mong muốn bằng cách thực hiện tuần tự việc phân chia lặp đi lặp lại của k – 1.

Trong phương pháp này, ma trận lần đầu tiên được phân chia thành 2 nhóm, sau đó một trong những nhóm này được chọn và chia tiếp. Quá trình này liên tục cho đến khi số cụm mong muốn được xây dựng. Sau mỗi bước, cụm được chia cho kết quả là 2 cụm. Dẫn đến giải pháp phân cụm tối ưu hóa được phân cụm theo tiêu chí hàm:

$$maximizes \sum_{i=1}^{k} \sqrt{\sum_{v,u \in Ri} sim(v,u)}$$

Trong đó, R_i là tập hợp các dòng được gán vào cụm thứ i. u, v đại diện cho 2 dòng và sim(u,v) là khoảng giữa 2 dòng. Việc phân cụm sẽ tiếp tục và được kiểm soát bởi các quy tắc, các quy tắc phân chia sẽ tối ưu hóa các giá trị phân cụm tổng hợp các tiêu chí chức năng.

Bước 2: Các dòng trong cùng cụm được kết hợp tạo thành dòng rút gọn mới gọi là cụm dòng. Cho G = {r_1,r_2,….,r_q} tập các dòng của cụm D. Sau đó cụm này có thể được biểu diễn như ma trận D = q * m. Cụm dòng của D, ký hiệu L = {l_1, l_2,…, l_m}, được tạo thành theo quy tắc:

$$Lj = \sum_{i=1}^{q} \bigvee di,j$$

Trong đó j = 1,2,…,m. Đó là giá trị các ô trong cụm dòng, giá trị bằng 0 chỉ khi tất cả các giá trị tạo lên là 0, ngược lại ô sẽ có giá trị bằng 1. Bằng cách trên O đã được chuyển thành ma trận nhỏ gọn O’ = l * m, trong đó l là số cụm và l ≤ n. Cho ma trận O **Hình 3.1** Giả sử các dòng {r1,r2,r3} {r5,r6} được nhóm thành 2 cụm L_1 và L_3. Khi đó ta có ma trận O (**Bảng 3.2**).

Bảng 3.2: Ma trận rút gọn O’.

	c_1	c_2	c_3	c_4	c_5	c_6	c_7
$l_1(r_1, r_2, r_3)$	1	1	0	0	1	1	0
$l_2(r_4)$	1	1	1	1	1	1	0
$l_3(r_5, r_6)$	1	1	1	1	0	0	1

Bước 3: C-Miner áp dụng chiến lược liệt kê các dòng rút gọn trên ma trận rút gọn O’ để phân chia không gian O thành các không gian con. Trong các bước trước, các dòng(hoặc cột) để liệt kê có khối lượng xử lý bằng nhau. Trong C-Miner,

khối lượng của mỗi cụm dòng là số dòng mà nó tạo ra. (ví dụ: Số dòng của các cụm tương ứng). Vì vậy, trong quá trình liệt kê các cụm dòng, độ hỗ trợ của 1 không gian con được tính bởi tổng độ lớn của các cụm dòng tương ứng. Chúng ta đang đề cập đến các không gian con O' như là các không gian rút gọn. Trong bất cứ thuật toán liệt kê dòng nào được sử dụng thì chúng phải đáp ứng được nhu cầu xử lý liệt kê lớn. Từ khi quá trình liệt kê dòng bằng với quá trình loại bỏ đệ quy từ các dòng hoặc tất cả các ô của ma trận có giá trị "0", chúng ta sử dụng chiến lược cây phân chia theo độ sâu(tương tự như D-Miner) để liệt kê dòng thu gọn, để hoạt động hiệu quả cho dự liệu dày đặc.

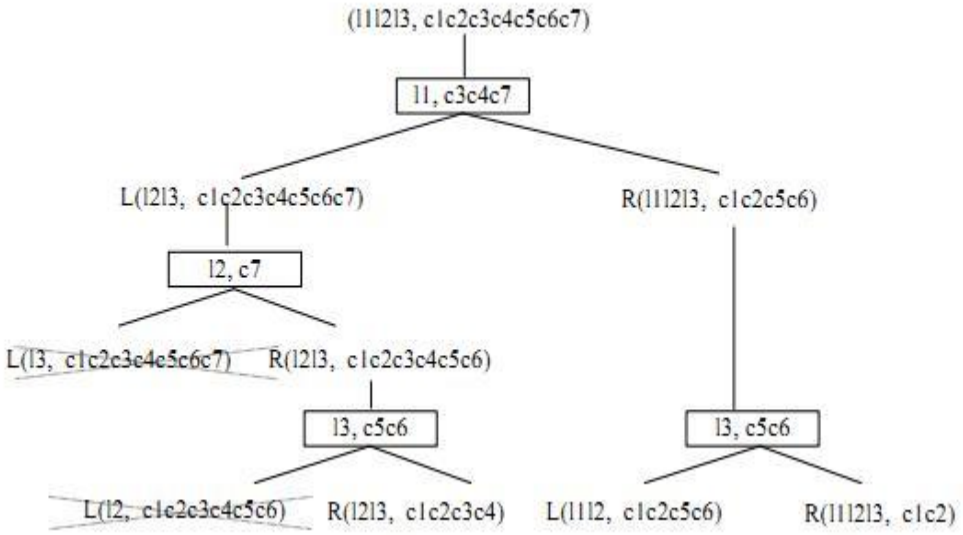
Chiến lược cây phân chia hoạt động như sau: Chúng ta nhòm tất cả các ô có giá trị "0" ở mỗi cụm dòng với nhau, và xác định từng nhóm như một lát cắt C(X,Y) trong đó X ∈ L và Y ∈ C. Như vậy số lượng các lát cắt bằng với số dòng có chứa ít nhất 1 phần từ "0". Lúc đó, lát cắt C(X,Y) phải thỏa mãn: $\forall l_i \in X$, $\forall c_i \in Y$, $O'_{i,j} = 0$ và $\forall c_k \in (C \backslash Y)$, $O'_{i,k} = 1$. **Bảng 3.3** cho thấy 3 lát cắt được sinh ra từ ví dụ ma trận O' trong **Bảng 3.2.**

Bảng 3.3: Lát cắt.

$C(X,Y)$
$C(l_1, c_3c_4c_7)$
$C(l_2, c_7)$
$C(l_3, c_5c_6)$

Cây phân chia lấy ma trận rút gọn làm gốc và phân chia nó bằng cách đệ quy sử dụng các lát cắt cho đến khi tất cả các lát cắt được sử dụng và tương ứng tất cả các ô trong mỗi không gian thu gọn được có giá trị là "1". Một lát cắt C(X,Y) được sử dụng để cắt một nút (L',C') nếu $X \cap L' = \emptyset$ và $Y \cap C' = \emptyset$. Theo quy tắc, Chúng ta định nghĩa cây con trái của nút là (L'\X,C') và cây con phải là (L',C'\Y). Kết quả các không gian rút gọn thể hiện một sự liệt kê đầy đủ của cụm dòng, không gian này thỏa mãn rằng buộc độ hỗ trợ và độ dài mẫu. Chỉ những nút không đáp ứng được rằng buộc độ hỗ trợ và độ dài mẫu thì bị lược bỏ đi. Vì vậy, các thông tin dư thừa cho khai phá tập phổ biến đóng được loại bỏ trong quá trình phân chia không gian con. Độ hỗ trợ của một nút được tính bằng tổng độ lớn cụm dòng của nút.

Cho minsup = 3 và minlen = 2. **Hình 3.2** cho thấy cây phân chia áp dụng cho ví dụ của chúng ta và các không gian rút gọn được thể hiện ở **Bảng 3.4.**



Hình 3.2: Cây phân chia sử dụng lát cắt.

Bảng 3.4: Kết quả các không gian rút gọn và không gian con.

Cluster-Row Set	Original Row Set	Original Column Set	Support	Pattern Length
l_1, l_2	r_1, r_2, r_3, r_4	c_1, c_2, c_5, c_6	4	4
l_1, l_2, l_3	$r_1, r_2, r_3, r_4, r_5, r_6$	c_1, c_2	6	2
l_2, l_3	r_4, r_5, r_6	c_1, c_2, c_3, c_4	3	4

Chúng ta lưu ý rằng thứ tự sắp xếp các lát cắt được áp dụng ảnh hưởng đến hiệu quả. Như một heuristic, lát cắt với độ ưu tiên lớn nhất thì được sử dụng đầu tiên như vậy nó sẽ cho kết quả cây ngắn hơn(do đó hiệu quả xử lý cao hơn).

- **Cuối cùng, trong bước 4,** mỗi không gian rút gọn, các cụm dòng được giải nén để tạo lại các dòng ban đầu. Quá trình giải nén này mở ra các ô mới, những ô mà có chứa giá trị 0 trong bộ dữ liệu tương ứng.

Bây giờ, mỗi bộ dữ liệu làm thành một không gian con từ đó chúng ta có thể khai phá các tập phổ biến đóng có thực (của bộ dữ liệu ban đầu). Xem xét các không gian rút gọn trong **Bảng 3.4**, chúng ta có, sau khi giải nén, kết quả các không gian con hiển thị trong **Bảng 3.4**.

❖ **Bộ đề 1:** Cho O là không gian khai phá ban đầu. Cho không gian con sinh ra bởi giai đoạn 1 của C-Miner từ không gian O được $S_1, S_2, \ldots, S_T$, $T > 1$. Lúc đó,

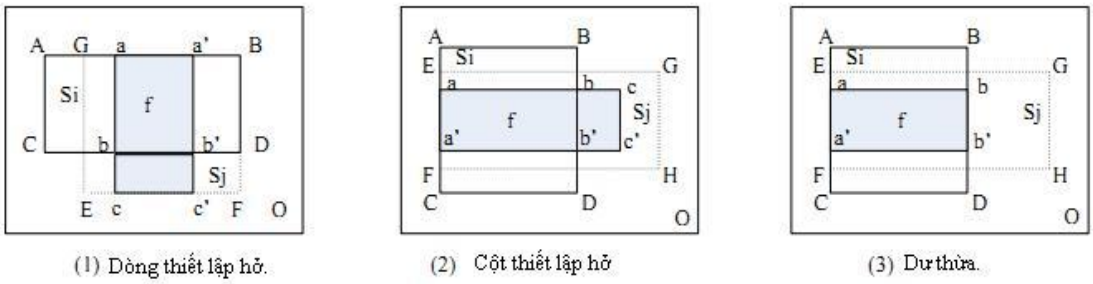
$$\text{MineFCP}(O) \in \bigcup_{i=1}^{t} \text{MineFCP}(Si).$$

➢ **Giai đoạn khai phá không gian con để tạo ra các tập phổ biến đóng.**

Để tạo ra các FCPs thực sự, thì mỗi không gian con phải được khai phá độc lập. Chúng ta sẽ sử dụng D-Miner**[1]** trong giai đoạn này.

Từ bộ đề 1, Chúng ta chú ý rằng, điều này có thể sảy ra: cho một FCP f rút ra từ một không gian con S_i (ví dụ, $f \in \text{MineFCP}(S_i)$) nó có thể là sai (ví dụ, $f \notin$

MineFCP(O)) hoặc f cũng có thể rút ra từ một không gian con khác S_j (ví dụ: f ∈ MineFCP(S_j); i ≠ j) . Có 3 trường hợp sai và dư thừa có thể xuất hiện (**Hình 3.3**).



(1) Dòng thiết lập hở. (2) Cột thiết lập hở (3) Dư thừa.

Hình 3.3: Sai sót và dư thừa.

- **Trường hợp 1:** Độ hỗ trợ tập hợp dòng của f ∈ MineFCP (S_i)) không đóng toàn cục. Trường hợp này xuất hiện khi tồn tại 1 dòng r_x ∈ R, dòng này nằm bên ngoài không gian con S_i nhưng lại chứa C_f (cột tập hợp của f). Sau đó, phải tồn tại f' ∈ MineFCP(S_i)) sao cho f ⊂ f'. Ví dụ: f = aa'bb' được khai phá từ không gian con S_i = ABCD không phải là tập hợp dòng đóng toàn cục trong đó f' = aa'cc' từ không gian con S_j = BEFG là tập cha của f'. Do đó, chúng ta có thể kết luận rằng, cho f = (R_f x C_f) ∈ MineFCP(S_i), nếu tồn tại một dòng r_x ∈ R và r_x ∉ R_i (dòng tập hợp của S_i) như vậy ∀C_y ∈ C_f, $O_{x,y}$ = 1, khi đó f phải được lược bỏ.
- **Trường hợp 2:** Cột tập hợp của f = (R_f x C_f) ∈ MineFCP(S_i)) không phải đóng toàn cục. Cho S_i = (l_{i1},l_{i2},…,l_{iu}) x C_i trong đó C_i là cột tập hợp và l_{ix} là cụm dòng đóng góp vào S_i . Cho l_{i1},l_{i2},…,l_{iu} là tập hợp dòng tương ứng(trong dữ liệu ban đầu) từ mỗi cụm dòng được sinh ra. Giả sử ∃l_{ix} ∈ (l_{i1},l_{i2},…,l_{iu}) như vậy l_f ∩ L_{ix} = ∅, đó là một số cụm dòng không đóng góp vào f. Sau đó phải tồn tại một không gian con khác không đóng góp vào cụm dòng S_j = ({l_{i1},l_{i2},…,l_{iu} }\ l_{ix}) x c_j trong đó c_i ⊂ cj. Điều này dẫn đến $\exists f' = (R'_f \times C'_f) \in MineFCP(S_j)$ Như vậy R_f = R'$_f$ và C_f ⊆ C'$_f$. Chúng ta chưa đề cập đến trường hợp C_f = C'$_f$ đó là trường hợp 3 phía dưới. Nếu C_f ⊂ C'$_f$, f phải được loại bỏ, nó không phải tập đóng toàn cục trong tập hợp cột. Ví dụ: f = aa'bb' từ S_i = ABCD không phải tập hợp cột đóng toàn cục nếu có tồn tại f' = aa'cc' từ không gian con S_j = EFGH.
- **Trường hợp 3:** f ∈ MineFCP(S_i)) là dư thừa trong đó f ∈ MineFCP(S_j)). Tiếp theo, điều kiện ở trường hợp 2 ở trên, nếu C_f = C'$_f$, sau đó f = f'. Vì vậy, f là dư thừa và có thể loại bỏ từ S_i. Ví dụ: f = aa'bb' có thể khai phá trong cả 2 không gian S_i = ABCD và S_j = EFGH.

Dựa trên những quan sát trên, chúng ta có thể đảm bảo rằng kết quả cuối cùng của chúng ta chứa tất cả và chỉ có những câu trả lời đúng. Trước khi chúng ta chứng minh kết quả này, chúng ta cung cấp các định nghĩa của tập dòng rút gọn đầu tiên.

✓ **Định nghĩa 3.8 Tập dòng rút gọn:**

Cho 1 không gian con rút gọn S_i = {l_{i1},l_{i2},…,l_{iu}} x C_i trong đó C_i là tập cột và l_x là cụm dòng cấu thành lên S_i, chúng ta định nghĩa l_{i1},l_{i2},…,l_{iu} như tập dòng rút gọn

tương ứng(trong bộ dự liệu ban đầu) từ mỗi cụm đòng được sinh ra.Với các cụm dòng l_1 trong bảng 3.2 làm ví dụ. Tập dòng rút gọn tương ứng là L1 = {r_1,r_2,r_3}.

Bây giờ, cho mỗi FCP sinh ra từ không gian con S_i, chúng ta lược bỏ đi những FCP dư thừa hoặc những lỗi cơ bản dựa trên những nguyên tắc cắt tỉa được đưa ra ở bộ đề 2: (i) điều kiện (a) nghĩa là một số tập dòng thu gọn của S_i không góp phần vào f. Lược bỏ bởi điều kiện (a) loại bỏ đi bất cứ f mà không phải là tập cột đóng toàn cục hoặc dư thừa; (ii) điều kiện (b) ngĩa là một vài dòng khác bên ngoài S_i chứa tập cột của f. Việc loại bỏ bởi điều kiện (b) lược đi bất kỳ f mà không phải tập hàng đóng toàn cục.

❖ **Bộ đề 2:** Cho O là không gian ban đầu. Cho $S_1,\ldots,S_t$ là các không gian con được sinh ra trong giai đoạn 1 của C-Miner. Cho $S_i = R_i \times C_i$ và cho $f = (R_f \times C_f) \in$ MineFCP(S_i). Sau đó, f có thể bị lược bỏ nếu (a) $\exists L_{ix} \subset R_i$ vì vậy $R_f \cap L_{ix} = \emptyset$; hoặc (b) $\exists R_y \in (R \backslash R_i)$ như vậy $\forall C_z \in C_f$, $O_{i,z} = 1$.

❖ **Bộ đề 3:** Cho O là không gian ban đầu. Cho $S_1,\ldots,S_t$ là không gian con được sinh ra trong giai đoạn 1 của C-Miner. Cho $P_1,\ldots,P_t$ là tập các FCPs mà được lược bỏ từ không gian con tương ứng trong giai đoạn 2. Khi đó,

MineFCP(Si) - Pi ∈ MineFCP(O)

Định lý 1. Cho O là không gian ban đầu. Cho $S_1,\ldots,S_t$ là các không gian con tạo ra trong giai đoạn 1 của C-Miner. Cho $P_1,\ldots,P_t$ là tập các FCPS được lược bỏ từ các không gian con tương ứng trong giai đoạn 2. Do vậy,

$$\text{MineFCP (O)} = \bigcup_{i=1}^{t} (MineFCP(S_i) - P_i).$$

Trong ví dụ (**Bảng 3.1**), sau giai đoạn 2, các FCPs thu được được biểu diễn ở **Bảng 3.5**.

Bảng 3.5: FCP(minsup = 3; minlen = 2).

support set	FCP	support	pattern length
r_1, r_2, r_4	c_1, c_6	3	2
r_2, r_3, r_4	c_2, c_6	3	2
r_2, r_4, r_5	c_1, c_2	3	2
r_4, r_5, r_6	c_1, c_4	3	2

3.1.6 Thuật toán B-Miner.

B-Miner dựa trên cơ sở đối tượng các dòng cơ bản. Thuật toán B-Miner cũng bao gồm 2 giai đoạn**[1]**.

- **Giai đoạn phân vùng không gian khai phá.**

B-Miner phân vùng không gian O = R x C trong hai bước: phân vùng tập dòng và phân vùng tập cột.

Trong bước đầu tiên, tập dòng R được phân chia thành một vài nhóm dòng khác nhau, được định nghĩa là các nhóm dòng cơ sở (BRGs). Số lượng dòng trong mỗi BRG là giống nhau, số lượng này do người dùng truyền vào bằng tham số, số lượng dòng được định nghĩa là độ dài nhóm (GL). Với GL = k, tập dòng R = $\{r_1,r_2,\ldots,r_n\}$ được chia thành q BRGs: $\{r_1,r_2,\ldots,r_k\}$, $\{r_{k+1},r_{k+2},\ldots,r_{2k}\},\ldots,\{r_{q*k+1},r_{q*k+2},\ldots,r_{q*k}\}$, trong đó $q = \frac{n}{k} + 1$. Với 1 $BRG_l = \{r_{(l-1)*k+1}, r_{(l-1)*k+2},\ldots, r_{(l-1)*k}\}$ được định nghĩa là tập dòng trước (FRS_l) của của BRG_l; và $\{r_{l*k+1},\ldots,r_n\}$ được định nghĩa là tập dòng (LRS_l) sau của BRG_l.

Trong bước thứ hai, bằng cách chiếu trên mỗi BRGs, tập cột C = $\{c_1,c_2,\ldots,c_m\}$ phân chia thành q nhóm cột, định nghĩa là nhóm cột cơ bản (BCGs). cho nhóm dòng cơ bản thứ l $BRG_l = \{r_{(l-1)*k+1},r_{(l-1)*k+2},\ldots,r_{l*k}\}$, nhóm cột cơ bản BCG_l = $\{c'_1,c'_2,\ldots,c'_m\}$ trong đó $c'_1,c'_2,\ldots,c'_m \subseteq C$ và $\forall C'_j \in \{c'_1,c'_2,\ldots,c'_m\}$, $\sum_{i'=(l-1)*k+1}^{l*k} \vee O_{i',j'} = 1$.

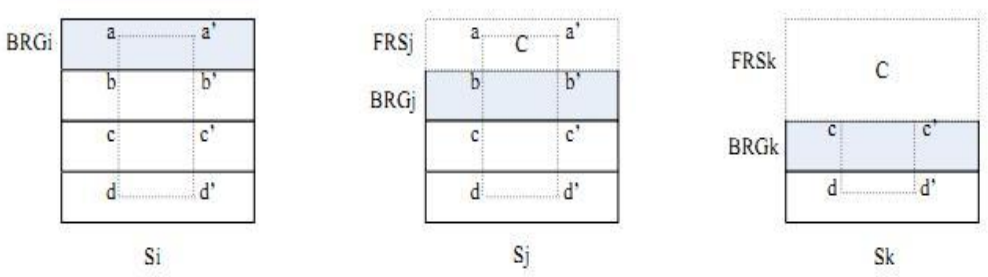
Mỗi không gian con được tạo thành từ ba yếu tố: BRG, LRS, và BCG. Do đó, các không gian con thứ i $S_i = (BRG_i \cup LRS_i)$ x BCGi, điều này cũng tương đương với $S_i = LRS_{i-1}$ x BCG_i. Cho ma trận O trong **Bảng 3.1** Ví dụ, với GL = 2, có ba không gian con tạo ra: $S_1 = \{r_1,r_2,r_3,r_4,r_5,r_6\}$ x $\{c_1,c_2,c_5,c_6\}$, $S_2 = \{r_3,r_4,r_5,r_6\}$ x $\{c_1,c_2,c_3,c_4,c_5,c_6\}$, $S_3 = \{r_5,r_6\}$ x $\{c_1,c_2,c_3,c_4,c_7\}$.

FCPS sẽ không được tạo ra trong các không gian con mà có sô dòng ít hơn minsup. Do đó, số lượng không gian con $q = \frac{(n-minsup)}{k} + 1$ thay vì $\frac{n}{k} + 1$. Nó an toàn bỏ qua những không gian con mà không có đủ số dòng. Tập cột với đủ sự hỗ trợ dòng được bao phủ bởi các không gian con trước. Đối với các ví dụ trên, nếu chúng ta đặt minsup = 3, chỉ có hai không gian con đầu tiên (S_1 và S_2) sẽ được khai phá. Không gian con cuối cùng S_3 với chỉ 2 dòng sẽ được lược bỏ an toàn.

❖ **Bổ đề 4:** Cho O là không gian khai thác ban đầu. Cho các không gian con tạo ra bởi giai đoạn 1 của B-Miner từ O được các không gian con $S_1,S_2,\ldots,S_T$, T ≥ 1. Khi đó $\text{MineFCP}(O) \subseteq \bigcup_{i=1}^{t} MineFCP(S_i)$.

➢ **Giai đoạn khai phá không gian con để tạo ra FCPS.**

Giống như C-Miner, bất kỳ thuật toán khai phá FCP nào cũng có thể được áp dụng trên các không gian con. Ở đây chúng ta tiếp tục sử dụng D-Miner để khai phá các FCP từ các không gian con. Tuy nhiên, do cách phân chia không gian, một số FCPS hở trên toàn cục hoặc dư thừa .



Hình 3.4: Ví dụ về sai sót và dư thừa.

Hình 3.4 cho thấy một số ví dụ. Hãy xem xét ba không gian con liên tiếp S_i, S_j, và S_k. Rõ ràng là một mẫu khai phá từ các không gian con trước đó có thể xuất hiện lại trong các không gian con sau. Ví dụ, mẫu bb'dd' từ S_i cũng có thể xuất hiện ở S_j, và cc'dd' từ S_i cũng có thể xuất hiện ở cả S_j và S_k. Như vậy là trường hợp dư thừa. Hơn nữa, mẫu từ các không gian con sau có thể bị hở nếu tập cột của nó chứa trong FRS của nó. Cho bb'dd' từ S_j làm ví dụ, Khi tập cột của nó cũng tồn tại trong FRS_j, nó là hở trong đó mẫu aa'dd' từ các không gian con trước S_i là tập cha và đóng toàn cục.

Để loại bỏ triệt để các FCPs hở trên toàn cục hoặc dư thừa, chúng tôi phát triển hai chiến lược cắt tỉa (xem Bổ đề 5). Điều kiện đầu tiên có nghĩa là các FCP không chứa bất kỳ dòng nào của BRG các không gian con. Các điều kiện lược bỏ do đó đảm bảo không có dư thừa, nghĩa là, FCPs từ một không gian con nhất định sẽ không xuất hiện lại trong các không gian con sau đó. Ví dụ, trong 3,4 hình, FCPs từ S_i mà không có độ hỗ trợ dòng trong BRG_i như bb'dd', cc'dd' sẽ bị lược bớt đi, trong khi FCPs như aa'dd', aa'cc' sẽ được giữ lại. Điều kiện thứ hai có nghĩa là có một dòng trong FRS các không gian con của nó mà có chứa toàn bộ tập cột của FCP. Điều kiện cắt tỉa do đó đảm bảo không có FCPS toàn cục bị hở, nghĩa là, FCPs từ một không gian con nhất định sẽ không có bất kỳ tập cha nào trong các không gian con trước của nó. Ví dụ, trong hình 3.4, FCPs từ S_j nhưng với độ hỗ trợ dòng trong FRS_j như là bb'dd' sẽ bị lược bớt đi vì nó có một tập cha aa'dd' trong không gian con trước S_i.

❖ **Bổ đề 5:** Cho O là không gian ban đầu. Cho $S_1,\ldots,S_t$ là các không gian con được tạo ra trong giai đoạn 1 của B-Miner. Cho $FCP_i = \{r_{i1},\ldots,r_{iu}\} \times \{c_{i1},\ldots,c_{iv}\}$ là các mẫu khai phá được từ các các không gian con S_i. Sau đó các FCP_i có thể bị lược bỏ

nếu (a) $\{r_{i1},\ldots,r_{iu}\} \cap BRG_i = \emptyset$ hoặc (b) $\exists r_x \in FRS_i$, như vậy là $\forall c_{iy} \in \{c_{i1},\ldots,c_{iv}\}$, $O_{x,iy} = 1$.

Chúng ta lưu ý là mỗi không gian con Si có thể được khai phá độc lập mà không cần bất kỳ kết quả của các không gian con khác. Như vậy, tất cả các nút có thể làm việc song song khi khai phá các không gian con được giao.

❖ **Bổ đề 6:** Cho O là không gian ban đầu. Cho $S_1,\ldots,S_t$ là các không gian con được tạo ra trong giai đoạn 1 của B-Miner. Cho $P_1,\ldots,P_t$ là tập hợp các FCPS bị lược bớt từ các không gian con tương ứng trong giai đoạn 2. Khi đó, MineFCP (S_i) - $P_i \subset$ MineFCP(O).

Định nghĩa 2: Cho O là không gian ban đầu. Cho $S_1,\ldots,S_t$ là các không gian con được tạo ra trong giai đoạn 1 của B-Miner. Cho $P_1,\ldots,P_t$ là tập hợp các FCPS bị lược bớt từ các không gian con tương ứng trong giai đoạn 2. Khi đó,

$$\text{MineFCP(O)} = \bigcup_{i=1}^{t} (MineFCP(S_i) - Pi).$$

3.1.7 Khai phá tập phổ biến đóng song song.

Như đã đề cập trong các phần phụ trước đó, khung khai phá FCP tiến bộ có thể dễ dàng thích nghi để xử lý song song. Trong phần này, chúng tôi sẽ trình bày khung khai phá FCP song song.

Khung khai phá FCP tiến bộ dễ dàng thích nghi để xử lý song song theo 3 giai đoạn.

- Giai đoạn biểu diễn tác vụ: Giai đoạn biểu diễn tác vụ tương ứng với giai đoạn sinh ra các không gian con của khung tiến bộ. Vì vậy, không gian ban đầu được phân chia thành các không gian con như vậy khai phá tất cả các không gian con sẽ dẫn đến một tập lớn các câu trả lời. Giai đoạn này có thể được thực hiện tại nút gốc (trong trường hợp này, nút gốc tạo ra tất cả các không gian con). Ngoài ra, chúng ta có thể làm song song giai đoạn này bằng cách khai phá nhiều nút hơn nữa để thực hiện phân vùng: (a) nút nguồn sẽ tạo ra t1 không gian con; (b) những không gian con sau đó được phân bổ thành t1 nút (bao gồm cả nút nguồn); mỗi nút t1 sẽ tiếp tục phân bố vào t2 những không gian con nhỏ hơn, sau đó các không gian con lại được tiếp tục phân phối cho các nút t2; (c) quá trình trên được lặp lại cho đến khi đủ số tác vụ / không gian con được tạo ra. Để đơn giản, trong việc tìm hiểu thực nghiệm của chúng ta, tác vụ này được thực hiện bởi các nút nguồn (nghĩa là chúng ta không làm song song giai đoạn này).
- Giai đoạn phân phối tác vụ: Trong giai đoạn hai, nút nguồn (nút mà có vai trò như một điều phối viên) sẽ chỉ định một không gian con tại mỗi nút để khai phá.
- Giai đoạn thực hiện tác vụ: Cuối cùng, trong giai đoạn thứ ba, tương tự như giai đoạn khai phá các không gian con, mỗi nút khai phá độc lập trong các không gian con phân bổ.

Chúng ta lưu ý rằng các giai đoạn thứ hai và thứ ba hoạt động lặp đi lặp lại: bất cứ khi nào một nút xử lý hoàn tất không gian con của nó, nó sẽ yêu cầu nút nguồn thay thế không gian con khác. Theo cách này, hệ thống cần tải lại.

Bây giờ, cả C-Miner và B-Miner đều có thể được thi hành song song theo khung. Tuy nhiên, có một vấn đề cần giải quyết: để cho một nút có thể được khai phá một không gian con S_i độc lập, sự lược bỏ các sai sót hoặc dư thừa các FCP_S phải được thực hiện mà không chịu bất kỳ chi phí đồng bộ giữa các nút. Để đảm bảo điều này, chúng ta cần truyền dữ liệu ban đầu O cho tất cả các nút tham gia. May mắn thay, Chi phí này là không tốn kém (về mặt thời gian đáp ứng) vì nó có thể được thực hiện đồng thời trong khi các không gian dữ liệu đang được phân vùng. Hơn nữa, chỉ có một bản sao cho mỗi một nút là cần thiết ngay cả khi nhiều không gian con được phân bổ cho một nút. Thêm vào đó, các bộ dữ liệu thực sự của chúng ta không phải là lớn.

3.1.8 Độ phức tạp thời gian.

Vấn đề của khai phá tập phổ biến tối đa là một tập hợp các vấn đề con của khai phá tập phổ biến đóng, đã được chứng minh bởi NP-hard [57]. Đối với bộ dữ liệu 2 chiều O = R x C, trong đó |R| = N, |C| = M, và η là số không gian con phân vùng trong quá trình khai phá, độ phức tạp thời gian của C-Miner và B-Miner là $O(2^{\frac{N}{\eta}} \times M)$, khi không áp dụng bất kỳ chiến lược bỏ nào. Bằng cách áp dụng minsup, minlen và liên kết chặt chẽ, hiệu quả của C-Miner và B-Miner được cải thiện rất nhiều.

3.2 Phương pháp khai phá tập phổ biến đóng trong không gian 3 chiều.

3.2.1 Tổng quan.

Mặc dù một số thuật toán khai phá FCP hiệu quả đã được tìm hiểu trong chương trước, nhưng những thuật toán này đều giới hạn ở việc phân tích tập dữ liệu 2 chiều, ví dụ như tập gen-thời gian trong việc phân tích gen, bộ dữ liệu các giao dịch-món hàng trong phân tích thị trường. Với những tiến bộ mới đây trong công nghệ microarray, tập hợp gen, tập hợp các mẫu có thể được biểu diễn trong suốt một dãy các điểm thời gian. Điều này mang lại kết quả là dữ liệu 3D microarray mẫu-gen-thời gian. Mô hình mới cung cấp mối quan hệ chắc chắn giữa mẫu-gen-thời gian có giá trị hơn trong vấn đề nghiên cứu gen. Ngay cả trong phân tích thị trường truyền thống, điều này thường để lấy một số thông tin của người tiêu dùng về một số khía cạnh, ví dụ: dữ liệu khu vực-thời gian-món hàng mà các cửa hàng bán hàng tại các địa điểm nhất định trong một thời gian nhất định. Xu hướng này thúc đẩy chúng ta mở rộng phân tích tập phổ biến đóng 2 chiều hiện có thành phân tích tập phổ biến đóng trong 3D ngữ cảnh. Chúng ta tham khảo các tập phổ biến đóng trong bối cảnh như khối lập phương phổ biến đóng 3 chiều (FCC). Thiết kế thuật toán hiệu quả để phát hiện các FCCs là chủ đề của chương này.

Phân tích sự kết hợp dựa trên các FCCs có thể cung cấp nhiều thông tin thú vị trong bối cảnh 3D. Chúng ta cho một ví dụ trong phân tích thị trường giỏ hàng. Trong khi phân tích tập phổ biến 2D cho một nhóm các mặt hàng có khả năng được mua cùng nhau trong một tập hợp các giao dịch, một FCC 3D dựa trên một tập dữ liệu bán hàng (khu vực,thời gian,món hàng) sẽ đại diện cho một nhóm các mục có

khả năng được mua cùng nhau tại một số địa điểm trong một tập hợp các khoảng thời gian. Những thông tin này sẽ cho phép các nhà cung cấp triển khai sản phẩm của họ đến các chuỗi điểm đặt hàng tại nhiều nơi khác nhau trong thời gian nhất định mà người tiêu dùng sẻ có các hành vi mua bán tương tự.

Trong chương này, chúng ta giải quyết vấn đề của khai phá các FCCs từ bộ dữ liệu 3D. Các FCCs cung cấp các mối quan hệ ba chiều đóng. Nghĩa là, chúng ta xác định các mẫu tối đa trong một bối cảnh 3D. Các mẫu 3D là tối đa trong đó sự gia tăng kích thước bất kỳ của 1 chiều sẽ làm giảm trực tiếp ít nhất một trong hai kích thước khác, nghĩa là, không có sự mở rộng nào hơn nữa trong không gian bất kỳ có thể được thực hiện trên mẫu.

Trước tiên, chúng ta giới thiệu các khái niệm của FCC và định nghĩa nó.

Thứ hai, chúng ta tìm hiểu hai thuật toán để khai phá các FCCs. Thuật toán đầu tiên là một khung ba giai đoạn, thuật toán Khai phá lát đại diện (RSM: "Representative Slice Mining"). Thuật toán RSM sử dụng các thuật toán khai phá FCP 2D để khai phá các FCCs. Ý tưởng cơ bản là chuyển đổi một bộ dữ liệu 3D thành một tập hợp các bộ dữ liệu 2D, khai phá bộ dữ liệu 2D bằng cách sử dụng một thuật toán khai phá FCP 2D đã có, và sau đó loại bỏ bất kỳ khối phổ biến mà không phải đóng. Thuật toán thứ hai được gọi là CubeMiner, nó hoạt động trực tiếp trên các dữ liệu 3D để khai phá các FCCs.

Thứ ba, chúng ta cũng cho thấy làm cách nào mà CubeMiner và RSM có thể dễ dàng mở rộng để khai phá song song.

3.2.2 Sự chuẩn bị.

Chúng ta đầu tiên phải xác định một số khái niệm mà chúng ta sẽ sử dụng trong suốt chương này, và sau đó cung cấp cho các miêu tả vấn đề.

Cho $R = \{r_1,r_2,\ldots,r_n\}$ biểu thị một tập các hàng, $C = \{c_1,c_2,\ldots,c_n\}$ biểu thị một tập hợp các cột, và $H = \{h_1,h_2,\ldots,h_n\}$ biểu thị một tập độ cao. Sau đó, một bộ dữ liệu ba chiều có thể được biểu diễn bằng ma trận nhị phân l x n x m , $O = H \times R \times C = \{O_{k,i,j}\}$. Với $k \in [1,l]$, $i \in [1,n]$, $k \in [1,m]$. Môi ô tương ứng là mối quan hệ giữa độ cao h_k, dòng r_i, cột c_j. Giá trị đúng là "1" biểu hiện cho mối quan hệ mà trong 3 chiều có ít nhất hai chiều bất kỳ đồng thời chứa giá trị 1 (S-contained).

Bảng 3.5: cho thấy một ví dụ bộ dữ liệu ba chiều kiểu Boolean. Trong **Bảng 3.5**, h_1 và r_4 S-contained trong c_3 và c_5, ký hiệu là $C(h_1 \times r_4) = \{c_3,c_5\}$; h_2 và c_5 S-contained trong r_1 và r_4, ký hiệu là $R(h_2 \times c_5) = \{r_1,r_5\}$, r_2 và c_1 S-contained trong h_1 và h_3, ký hiệu là $H(r_2,c_1) = \{h_1,h_3\}$.

Bảng 3.6: Ví dụ bộ dữ liệu ba chiều nhị phân.

$H = h_1$

R/C	c_1	c_2	c_3	c_4	c_5
r_1	1	1	1	0	1
r_2	1	1	1	0	0
r_3	1	1	1	1	1
r_4	0	0	1	0	1

$H = h_2$

R/C	c_1	c_2	c_3	c_4	c_5
r_1	1	1	1	1	1
r_2	0	1	1	1	0
r_3	1	1	1	1	0
r_4	1	1	1	0	1

$H = h_3$

R/C	c_1	c_2	c_3	c_4	c_5
r_1	1	1	1	0	0
r_2	1	1	1	0	0
r_3	1	1	1	1	0
r_4	1	1	0	1	1

❖ **Định nghĩa 4.1 Độ hỗ trợ chiều cao và độ hỗ trợ-H:** Cho một tập hợp các hàng R’ ⊆ R và một bộ cột C’ ⊆ C, tập độ cao lớn nhất mà đồng thời chứa R’ và C’ được định nghĩa là độ hỗ trợ chiều cao H(R’ x C’) ⊆ H. Chiều cao H(R’ x C’) được định nghĩa là hỗ trợ-H của (R’ x C’), ký hiệu là |H(R’ x C’)|.

Ví dụ Bảng 3.6, cho R’ = {r_1,r_2} và C’ = {c_1,c_2,c_3}, và H(R’ x C’) = {h_1,h_3} như vậy cả h1 và h3 đồng thời chứa {r_1,r_2} và {c_1,c_2,c_3}, và không có chiều cao khác đồng thời chứa chúng.

❖ **Định nghĩa 4.2 Độ hỗ trợ dòng và hỗ trợ-R:** Cho một tập hợp các cột C’ ⊆ C và một tập các chiều cao H’ ⊆ H, Tập dòng lớn nhất mà đồng thời chứa C’ và H’ được định nghĩa là tập hỗ trợ dòng R(C’ x H’) ⊆ R. Số dòng trong R(C’ x H’) được định nghĩa là hỗ trợ-R của (C’ x H’), ký hiệu là |R(C’ x H’)|.

Ví dụ Bảng 3.6, cho C’ = {c_1,c_2,c_3} và H’ = {h_1,h_3}, và R(C’ x H’) = {r_1,r_2,r_3} như vậy r_1, r_2 và r_3 đồng thời chứa {c_1,c_2,c_3} và {h_1,h_3}, và không có dòng nào khác đồng thời chứa chúng.

❖ **Định nghĩa 4.3 Độ hỗ trợ cột và hỗ trợ-C:** Cho một tập hợp các dòng R’ ⊆ R và một tập các chiều cao H’ ⊆ H, tập cột lớn nhất mà đồng thời chứa R’ và H’ được định nghĩa là tập hỗ trợ cột C(R’ x H’) ⊆ C. Số cột trong R(C’ x H’) được định nghĩa là hỗ trợ-C của (R’ x H’), ký hiệu là |C(R’ x H’)|.

Ví dụ Bảng 3.6, cho R’= {r_3,r_4} và H’ = {h_2,h_3}, và C(R’ x H’) = {c_1,c_2} như vậy c_1, c_2 và đồng thời chứa {r_3,r_4} và {h_2,h_3}, và không có cột nào khác đồng thời chứa chúng.

❖ **Định nghĩa 4.4 Khối lập phương đóng:** Cho tập dòng R' ⊆ R, tập cột C'⊆ C, và tập chiều cao H'⊆ H, một khối lập phương A = (H' x R' x C') ⊆ O được định nghĩa là một Khối lập phương đóng nếu (1) R' = R(C' x H'), (2) C' = C(R' x H') và (3) H' = H(R' x C0). Nghĩa là, A = (H' x R' x C') có thể viết là A = (H',R',C'). Hơn nữa, các điều kiện (1), (2) và (3) được gọi lần lượt là tập dòng đóng, tập chiều cao đóng và tập cột đóng tương ứng. Bằng trực giác, một khối lập phương đóng hoàn chỉnh(tất cả giá trị bên trong bằng 1) và là lớn nhất (không có khối hoàn chỉnh nào lớn hơn chứa nó).

❖ **Định nghĩa 4.5 Khối lập phương phổ biến đóng (FCC):** Một khối A = (H';R',C') ⊆ O được gọi là một khối lập phương phổ biến đóng nếu (1) hỗ trợ-H |H(R' x C')|, hỗ trợ-R |R(H' x C')|, và hỗ trợ-C |C(R' x H')| cao hơn ngưỡng hỗ trợ-H tối thiểu (minh), ngưỡng hỗ trợ-c tối thiểu (minr), và ngưỡng hỗ trợ-C tối thiểu (minc) tương ứng; và (2) A là một khối lập phương đóng.

Ví dụ, cho minh = minr = Minc = 2.

Khối lập phương A = $\{h_1,h_3\}$ x $\{r_1,r_2,r_3\}$ x $\{c_1,c_2,c_3\}$ sẽ là một khối lập phương phổ biến đóng trong **Bảng 3.6**. Tuy nhiên, A' = $\{h_1,h_3\}$ bảng $\{r_2,r_3\}$ x $\{c_1,c_2,c_3\}$không phải là một khối lập phương phổ biến đóng trong đó $\{r_2,r_3\} \neq R(\{h_1,h_3\} \text{ x } \{c_1,c_2,c_3\}) = \{r_1,r_2,r_3\}$. Để rõ ràng, khối lập phương A' = $\{h_1,h_3\}$ x $\{r_2,r_3\}$ x $\{c_1,c_2,c_3\}$ được viết là A' = $(h_1h_3,r_2r_3,c_1c_2c_3)$.

❖ **Định nghĩa Vấn đề:** Cho một bộ dữ liệu ba chiều O, vấn đề của chúng ta là để khai phá tất cả các khối phổ biến đóng đối với các ngưỡng hỗ trợ người dùng đặt ra minh, minr, và minc.

3.2.3 Thuật toán khai phá lát đại diện(RSM).

Trong phần này, chúng ta đề xuất một Khung, được gọi là khai phá lát đại diện (RSM)**[1]**, để khai phá các FCCs. Trong Khung này, bất kỳ thuật toán khai phá FCP 2D đều có thể được áp dụng để làm việc trên các bộ dữ liệu 3D. Khung này dựa trên ý tưởng là các tập dữ liệu 3D O = H x R x C được biểu diễn như là O = H x $slice_{rxc}$. Do đó, bất kỳ chiều nào H nào đều có thể được liệt kê đầu tiên. Sau đó, trên mỗi kết hợp của các lát, thuật toán FCP 2D có thể được áp dụng trên hai yếu tố khác như R và C. Cuối cùng, một chiến lược xử lý được áp dụng vào kết quả để loại bỏ các khối lập phương hở từ liệt kê khía cạnh H. Dựa trên ý tưởng này, chúng ta chia khung RSM thành ba giai đoạn như hình trong thuật toán 1.

Algorithm 1 *RSM* Framework

1: Global variables: H the set of heights, R the set of rows, C the set of columns, monotonic constraints $minH$, $minR$, and $minC$ on H, R, C respectively. α the set of height subsets, β the set of representative slices, γ the set of 2D FCPs. Let $MineFCP(\beta)$ denote the algorithm to mine the set of 2D FCPs for a slice β.
2: Input: 3D Matrix O with l heights, n rows and m columns.
3: Output: ξ the set of FCCs.
4: Phase 1: Representative Slice Generation
5: $\alpha \leftarrow \emptyset$;
6: **while** $|EnumerateSubset(H)| >= minH$ **do**
7: $\alpha \leftarrow \alpha \cup EnumerateSubset(H)$;
8: **end while**
9: $\beta \leftarrow SliceCombine(\alpha)$;
10: Phase 2: 2D FCP Generation
11: $\gamma \leftarrow MineFCP(\beta)$;
12: Phase 3: Post-Pruning
13: $\xi \leftarrow PostPruning(\gamma)$;

Trong giai đoạn 1, các lát đại diện được tạo ra dựa trên một bảng liệt kê các chiều và tổng hợp các lát. Trong giai đoạn 2, bất kỳ thuật toán khai phá tập phổ biến đóng 2D có thể được áp dụng cho khai phá FCPS 2D trên mỗi lát biểu diễn. Trong giai đoạn 3, một chiến lược loại bỏ được áp dụng để loại bỏ các FCCs hở trong bảng liệt kê chiều. Chúng ta sẽ biểu diễn chi tiết ba giai đoạn dưới đây.

3.2.3.1 Sự hình thành các lát đại diên.

Trong giai đoạn 1, đầu tiên chúng ta đưa ra độ cao H như là chiều cơ sở của chúng ta, và liệt kê tập H = $\{h_1,h_2,\ldots,h_l\}$ để có được tất cả các tập con của H (ký hiệu là H') như vậy |H'| $\geq$ minH. Cho tập dữ liệu trong **Bảng 3.6** làm ví dụ, cho minH = 2 chúng ta sẽ có được những tập hợp con $\{h_1,h_2\}$, $\{h_1,h_3\}$, $\{h_2,h_3\}$, $\{h_1,h_2,h_3\}$.

Thứ hai, những lát trong cùng một tập con được kết hợp để tạo thành lát đại diện (RS) mới. Cho một bộ dữ liệu 3D O = H x R x C = $\{O_{k,i,j}\}$ với k $\in$ [1; l], i $\in$ [1, n] và j $\in$ [1; m], và cho H0 = $\{h_1,\ldots,h_x\}$ là các tập con được kết hợp. Sau đó, RS của H' có thể được biểu diễn như một ma trận n x m như vậy $\forall O'_{i,j} \in$ RS; $O'_{i,j} = \sum_{k=1}^{x} \cap O_{k,i,j}$ trong đó i $\in$ [1, n] và j $\in$ [1; m]. Nghĩa là, giá trị ô của lát biểu diễn là 1 chỉ khi tất cả các ô tạo ra nó có giá trị là 1, ngược lại, giá trị ô là 0. Cột thứ 2 của **Bảng 3.7** cho thấy các lát đại diện của các ví dụ **Bảng 3.6**.

Bảng 3.7: Ví dụ RSM(minH = minR = minC = 2).

Height Set	Representative Slices	2D FCPs	3D FCCs
h_2, h_3	11100 01100 11110 11001	$r_1r_3 : c_1c_2c_3, 2:3$ $r_1r_3r_4 : c_1c_2, 3:2$ $r_1r_2r_3 : c_2c_3, 3:2$	$h_2h_3 : r_1r_3r_4 : c_1c_2, 2:3:2$
h_1, h_3	11100 11100 11110 00001	$r_1r_2r_3 : c_1c_2c_3, 3:3$	$h_1h_3 : r_1r_2r_3 : c_1c_2c_3, 2:3:3$
h_1, h_2	11101 01100 11110 00101	$r_1r_4 : c_3c_5, 2:2$ $r_1r_3 : c_1c_2c_3, 2:3$ $r_1r_2r_3 : c_2c_3, 3:2$	$h_1h_2 : r_1r_4 : c_3c_5, 2:2:2$
h_1, h_2, h_3	11100 01100 11110 00001	$r_1r_3 : c_1c_2c_3, 2:3$ $r_1r_2r_3 : c_2c_3, 3:2$	$h_1h_2h_3 : r_1r_3 : c_1c_2c_3, 3:2:3$ $h_1h_2h_3 : r_1r_2r_3 : c_2c_3, 3:3:2$

3.2.3.2 Sự hình thành các FCP 2D.

Trong giai đoạn 2, bất kỳ thuật toán khai phá FCP hiện có đều có thể được áp dụng trên mỗi lát đại diện để khai phá FCPs 2D dựa trên chiều R và C. Trong đồ án này của chúng ta, chúng ta áp dụng D-Miner. Sau khi khai phá, chúng ta sẽ có một bộ FCPs 2D cho chiều R và C. Ví dụ **Bảng 3.6**, các FCPS được hiển thị trong cột thứ 3 của **Bảng 3.7**

3.2.3.3 Sự hình thành các FCC 3D.

Trong giai đoạn 3, tập phổ biến 3D được tạo ra bằng cách kết hợp từng FCP 2D với chiều cao góp phần hình thành lát đại diện. Tuy nhiên, không phải tất cả những mẫu phổ biến 3D là các FCCs. Một số trong số chúng không đóng trong các tập chiều cao và cần được lược bớt đi. Ví dụ, trong **Bảng 3.7**, sau khi kết hợp FCP 2D đầu tiên "$r_1r_3 : c_1c_2c_3$, 2: 3" "với chiều cao góp phần " h_2, h_3", một mẫu phổ biến 3D " $h_2h_3 : r_1r_3 : c_1c_2c_3$; 2: 2: 3" được sinh ra. Mẫu phổ biến 3D này không phải là một FCC ở chỗ nó bị hở trong tập chiều cao và có một tập cha "$h_1h_2h_3 : r_1r_3 : c_1c_2c_3$; 2: 2: 3" (FCC thứ 4 ở cột thứ 4 của **Bảng 3.7**). Do vậy, FCP 2D không chỉ chứa trong lát h_2 và h_3, mà còn chứa trong lát h_1.

Để loại bỏ tất cả các mô hình phổ biến đóng 3D bị hở, chúng ta phát triển một chiến lược cắt tỉa sau dựa trên **Bổ đề 7**. Nếu một FCP 2D được chứa trong lát chiều cao khác ngoài lát chiều cao đóng góp của nó, nó là hở và do đó có thể được lược bỏ, nếu không, nó được giữ lại.

❖ **Bổ đề 7 Chiến lược cắt tỉa sau:** cho O' = H' x C' x R' là một tập phổ biến 3D và H là chiều cao đầy đủ. Nếu $\exists$H'' $\in$ (H \ H') như vậy $\forall h_k \in$ H'', $\forall r_i \in$ R', $\forall c_i \in$

C',$O_{k,i,j}$ = 1, O' là hở trong tập chiều cao và có thể được lược bớt đi, nếu không, O' được giữ lại.

Trong quá trình cắt tỉa sau, không phải tất cả các ô liên không góp phần cấu thành lát được kiểm tra. Như được trình bày trong thuật toán 2.

Algorithm 2 *RSM* Post Pruning

```
1: Input: 3D Pattern Set γ.
2: Output: 3D FCC Set ξ.
3: for a = 1 upto |γ| do
4:    (H', R', C') ← γ[a];
5:    flag1 ← 1;
6:    for k = 1 upto |H| do
7:       if h_k ∈ (H \ H') then
8:          flag2 ← 1;
9:          for i = 1 upto |R| do
10:            if r_i ∈ R' then
11:               for j = 1 upto |C| do
12:                  if c_j ∈ C' and O_{k,i,j} = 0 then
13:                     flag2 ← 0;
14:                     break;
15:                  end if
16:               end for
17:            end if
18:            if flag2 = 0 then
19:               break;
20:            end if
21:         end for
22:         if flag2 = 1 then
23:            flag1 ← 0;
24:            break;
25:         end if
26:      end if
27:   end for
28:   if flag1 = 0 then
29:      γ ← γ \ γ[a];
30:   end if
31: end for
32: return γ;
```

Trong mỗi quá trình kiểm tra lát, quá trình kiểm tra cột được lặp (từ dòng 12-17) bị chấm dứt bất cứ khi nào một ô có giá trị "0" bị phát hiện, điều đó trực tiếp dẫn đến việc chấm dứt vòng lặp kiểm tra dòng (từ dòng 10 đến 22). Có nghĩa là, bất kỳ một ô có giá trị "0" thì quá trình kiểm tra lát sẽ bị dừng. Và nếu chúng ta phát hiện rằng lát nào vượt qua vòng kiểm tra cột và dòng (tất cả giá trị các ô liên quan bằng "1"") mà không chấm dứt sớm, toàn bộ các vòng kiểm tra lát (từ dòng 7-28) có thể được chấm dứt lúc đó mẫu được khẳng định là hở. Chiến lược của thuật toán 2 đảm bảo rằng chúng ta hoàn tất việc kiểm tra chặt chẽ càng sớm càng tốt. Đối với ví dụ trong **Bảng 3.7**, sau quá trình cắt tỉa sau, các FCCs kết quả được hiển thị trong cột thứ 4.

3.2.3.4 Tính đúng đắn.

Định lý 3 cho thấy RSM có thể sinh ra tất cả và chỉ có tất cả các FCCs một cách chính xác.

❖ **Định lý 3.** Cho tập các khối phổ biến đóng FCCs của một bộ dữ liệu 3D. ξ ký hiệu cho kết quả thu được các khối phổ biến đóng khi chay RSM trên tập dữ liệu.

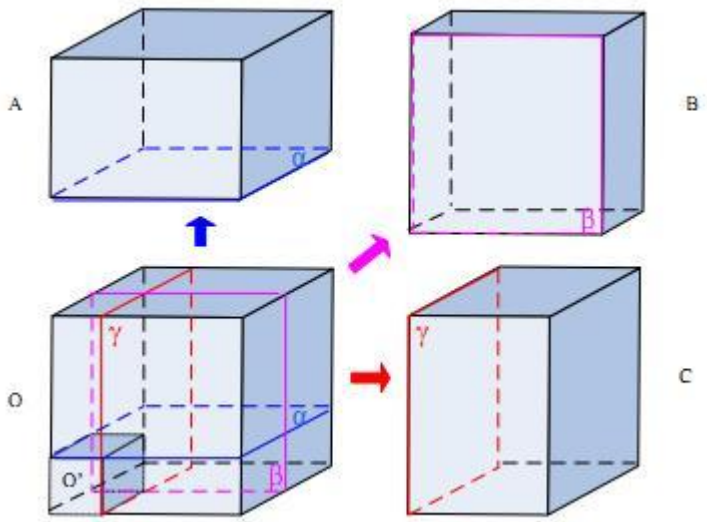
Khi đó các FCCs = ξ. Nói cách khác, RSM chính xác tạo ra tất cả và chỉ có tất cả các FCCs.

3.2.4 Thuật toán CubeMiner.

Trong khi RSM có lợi thế là nó có thể tái sử dụng các thuật toán khai phá FCP hiện tại, số lát 2D có thể là rất lớn. Trong phần này, chúng ta trình bày một phương pháp mới mà khai phá FCCs trực tiếp từ bộ dữ liệu 3D. Đầu tiên chúng ta phải trình bày các nguyên tắc đằng sau đề án CubeMiner**[1]**. Sau đó, chúng ta sẽ xem xét thuật toán, và cuối cùng chúng ta sẽ trình bày tính đúng đắn của CubeMiner.

3.2.4.1 Nguyên lý CubeMiner.

CubeMiner là một thuật toán mới để khai phá FCC (H0; R0, C0) dựa trên các ràng buộc. Nó xây dựng các tập H', R', và C' và sử dụng các ràng buộc ngưỡng hỗ trợ đơn điệu đồng thời trên H, R, và C để rút gọn không gian tìm kiếm. Một FCC cho thấy rằng tất cả các độ cao, các dòng và các cột của nó cùng chứa mối quan hệ. Từ đó, chúng ta phải xác định khối tối đa với tất cả các ô của nó có giá trị "1". Nếu chúng ta có thể loại bỏ ra giá trị 0 từ các khối dữ liệu ban đầu toàn bộ mà không thay đổi hình dạng của các khối còn lại, chúng ta sẽ thu hẹp không gian tìm kiếm rất nhiều.



Hình 3.5: CubeMiner.

Hình 3.5 minh họa các nguyên lý của CubeMiner. Cho khối lập phương O đại diện cho toàn bộ dữ liệu và ở góc bên trái là O' nằm trong O đại diện cho khu vực dư thừa (khu vực chứa giá trị "0") để được lược bỏ. Từ bề mặt của khối lập phương O', ba mặt phẳng α, β, γ. Ba mặt phẳng có thể chia khối lập phương O làm ba phần: khối phía trên A, khối phía sau B và khối bên phải C. Và phương trình A ∪ B ∪ C = O \ O' thỏa mãn. Trong bất kỳ phần A, B, C, có thể vẫn còn tồn tại khu vực chứa giá trị "0". Các nguyên lý chia tách tương tự có thể được áp dụng cho đến khi tất cả

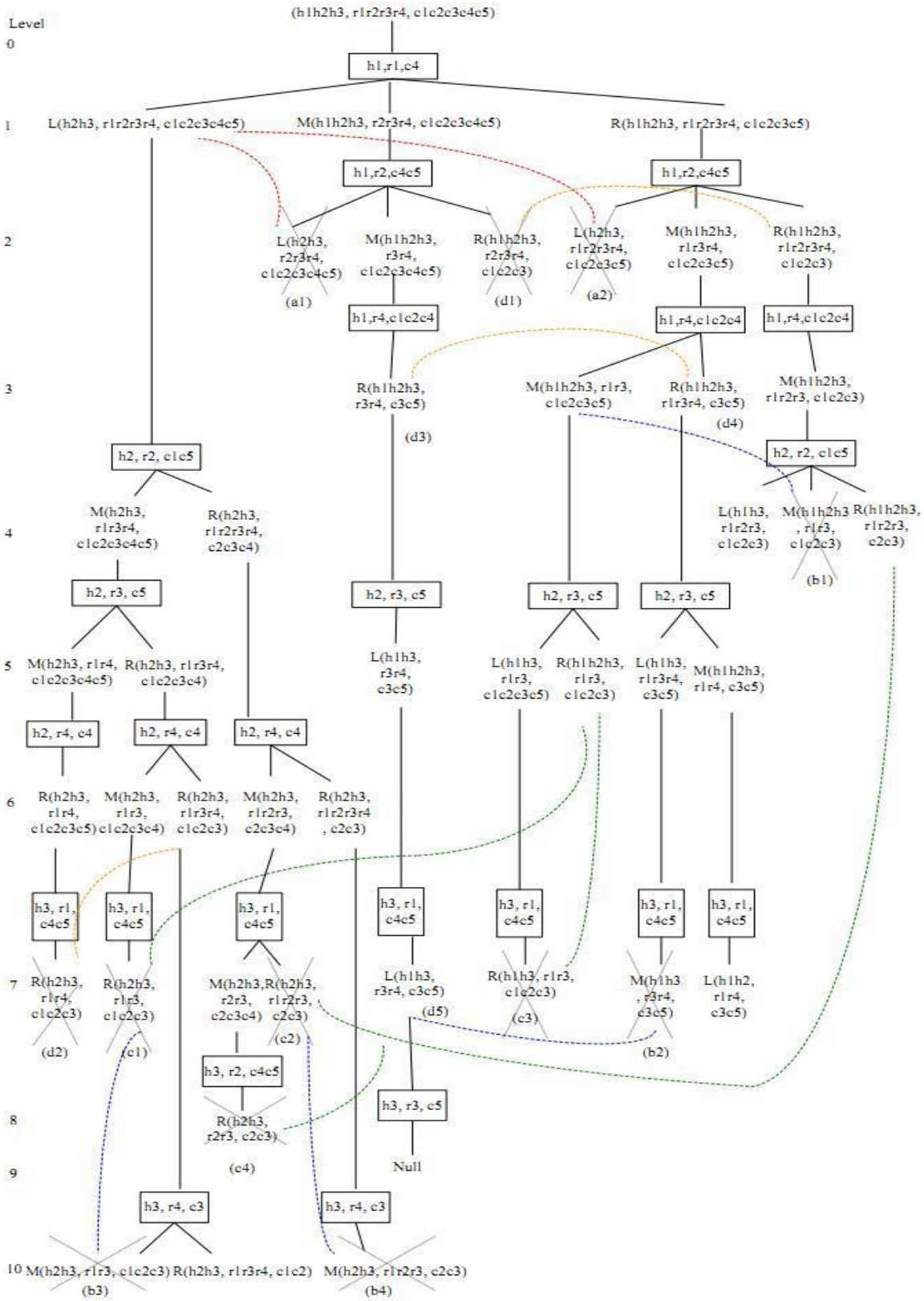
các khu vực chứa giá trị "0" được lược bỏ. Chúng ta cố gắng loại bỏ càng nhiều giá trị "0" càng tốt trong mỗi quá trình chia nhỏ. Trong quá trình quét dữ liệu, các giá trị "0" được tổng hợp với nhau trên chiều lớn nhất để đạt hiệu quả cao.

Chúng ta sử dụng Z để ký hiệu tập hợp các nhóm ô mà được phân vùng các giá trị sai (tức là "0") của ma trận Boolean. (W; X; Y) ∈ Z được gọi là một "*lát cắt*" nếu $\forall h_k \in W$, $\forall r_i \in X$; $\forall c_j \in Y$, $O_{k,i,j} = 0$ Và chúng ta gọi là W, X;. Y phần tử trái , phần tử giữa, và phần tử phải của lát cắt (W, X, Y) tương ứng. Chúng ta gộp các ô có giá trị "0" liên tiếp, do đó, Z chứa nhiều lát cắt như các dòng trong tất cả các lát chiều cao của ma trận dữ liệu 3D. Mỗi lát cắt gồm các ô có giá trị bằng 0 trong dòng. **Bảng 3.8** cho thấy 10 lát cắt của ma trận trong **Bảng 3.6**. Các lát cắt được sắp xếp theo thứ tự tăng dần theo phần từ đầu tiên bên trái và phần tử thứ hai ở giữa.

Bảng 3.8: Z (tập lát cắt).

W, X, Y
h_1, r_1, c_4
h_1, r_2, c_4c_5
$h_1, r_4, c_1c_2c_4$
h_2, r_2, c_1c_5
h_2, r_3, c_5
h_2, r_4, c_4
h_3, r_1, c_4c_5
h_3, r_2, c_4c_5
h_3, r_3, c_5
h_3, r_4, c_3

CubeMiner bắt đầu với bộ dữ liệu O(H, R, C) sau đó chia nó đệ quy bằng cách sử dụng các lát cắt của Z cho đến khi tất cả các lát cắt trong Z được sử dụng do đó tất cả các ô trong mỗi khối lập phương thu được có giá trị "1". Một lát cắt (W, X, Y) trong Z được sử dụng để lượt bớt một khối (H', R', C') nếu W ∩ H' ≠ ∅, X ∩ R' ≠ ∅, Y ∩ C' ≠ ∅. Trong trường hợp này, chúng ta nói rằng lát cắt đã được "áp dụng" cho khối. Thông thường, chúng ta xác định con trái của (H', R', C') bằng (H' \ W, R', C'), con giữa bằng (H'; R' \ X; C') và con phải bằng (H', R', C' \ Y). Đệ quy chia nhỏ dẫn đến tất cả các FCCs, nhưng cũng có một số khối hở, không tối đa. Chiến lược cắt tỉa cần phải được áp dụng để đảm bảo rằng chúng ta có được tất cả các FCCs và chỉ có FCCs. Chúng ta sẽ xem xét làm thế nào để phát triển các chiến lược cắt tỉa như vậy. **Hình 3.5** cho thấy cây được tạo ra từ ma trận 3D trong **Bảng 3.6**.



Hình 3.6: Cây khai phá FCC.

Từ **Hình 3.6**, chúng ta thấy rằng 10 lát cắt trong **Bảng3.8** phân chia tập dữ liệu ban đầu sau 10 bước (cấp). Chúng ta xác định bước từ gốc đến một nút là đường

dẫn của nút. Mỗi nút được chia thành ba nút mới trong bước tiếp theo nếu lát cắt được áp dụng. Chúng ta chỉ giữ và hiển thị các nút đáp ứng các ngưỡng hỗ trợ (cho minH = minR = Minc = 2) để hạn chế không gian. Tuy nhiên, trong mỗi bước, không phải tất cả các nút được tạo ra là hữu ích cho việc chia nhỏ hơn nữa Có bốn loại nút vô ích:

a. Con trái từ nhánh giữa / phải được cắt bởi lát cắt vì phần tử trái đã được cắt ở đường dẫn của nút trước. Ví dụ, phần tử trái h_1 của lát cắt (h_1, r_2, c_4c_5) đã cắt đường dẫn của con trái L(h_2h_3, $r_2r_3r_4$, $c_1c_2c_3c_4c_5$) (a_1 ở mức 2) và L(h_2h_3, $r_1r_2r_3r_4$, $c_1c_2c_3c_5$) (a_2 tại mức 2. a_1 từ nhánh giữa là hở trong tập dòng và a_2 từ nhánh phải là hở trong tập cột. Chúng được lược bớt đi vì là các tập con của nút L(h_2h_3, $r_1r_2r_3r_4$, $c_1c_2c_3c_4c_5$) (nút1 trong mức 1).

b. Con giữa từ nhánh phải được cắt bởi lát cắt mà phần tử giữa đã được cắt ở đường dẫn của nút trước. Ví dụ, phần tử giữa r_2 của lát cắt (h_2, r_2, c_1c_5) đã cắt đường dẫn của con giữa M($h_1h_2h_3$, r_1r_3, $c_1c_2c_3$) (b_1 ở mức 4). Con giữa này bị hở trong tập cột và cần được lượt bớt đi vì là tập con của nút M($h_1h_2h_3$, r_1r_3, $c_1c_2c_3c_5$) (nút 2 ở mức 3). Con giữa b_2, b_3 và b_4 là tất cả các trường hợp tương tự: chúng là bản sao hoặc tập hợp con của các nút khác.

c. Các nút mà hở trong tập chiều cao. Ví dụ, nút R(h_2h_3, r_1r_3, $c_1c_2c_3$) (c_1 trong mức 7) là hở trong tập chiều cao bởi vì tồn tại nút cha của nó R($h_1h_2h_3$, r_1r_3, $c_1c_2c_3$) (5 nút ở mức 5). Các nút như vậy cần phải lược bớt đi để đảm bảo tính đóng trong tập chiều cao. Các nút c_2, c_3, c_4 là những ví dụ như vậy.

d. Các nút mà hở trong tập dòng. Ví dụ, nút R($h_1h_2h_3$, $r_2r_3r_4$, $c_1c_2c_3$) (d_1 ở mức 2) là hở trong tập dòng vì có nút cha của nó R(h1h2h3, $r_1r_2r_3r_4$, $c_1c_2c_3$) (nút 6 trong mức 2). Các nút như vậy cần lược bớt đi để đảm bảo tính đóng trong tập dòng. Nút R(h_2h_3, r_1r_4, $c_1c_2c_3$) (d_2 tại mức 7) cũng là một trong những ví dụ bị lược bớt đi vì nó không đóng trong dòng r3. Lưu ý rằng có thể tồn tại một số nút đóng trong tập dòng mặc dù chúng có thể đã có một nút cha tạm thời trong quá trình xử lý. Ví dụ, nút R($h_1h_2h_3$, r_3r_4, c_3c_5) (d_3 ở mức 3) có một nút cha tạm thời R($h_1h_2h_3$, $r_1r_3r_4$, c_3c_5) (d_4 ở mức 3). Mặc dù nút d_3 xuất hiện tạm thời hở do dòng r_1, chúng ta phát hiện rằng sau khi áp dụng lát cắt sau (h_3, r_1, c_4c_5) ở mức độ 7, nút d4 bị mất đi tập cha của nó, và nút con của d3 L(h_1h_3, r_3r_4, c_3c_5) (d_5 trong mức 7) chỉ phục vụ như là một lý do để loại bỏ các con giữa M(h_1h_3, r_3r_4, c_3c_5) (b_2 con đẻ của d_4) một cách an toàn. Do đó, chẳng hạn tập dòng mà là tạm các nút thời hở trong khi xử lý được giữ lại trong đó chúng là những tập dòng đóng trong toàn bộ quá trình.

- Để loại bỏ các nút vô ích loại (a) và (b), chúng ta giữ hai tập TL = {W_1, W_2, W_p}; TM = {X_1, X_2,…, X_q} tại mỗi nút để theo dõi phần tử bên trái và giữa của lát cắt đã sử dụng để cắt đường dẫn của nút đó. Và dựa trên hai tập, chúng ta phát triển *Left Track Checking* trong **Bổ đề 8** và *Middle Track Checking* trong **bổ đề 9**. Trạng thái khởi tạo TL = TM = rỗng. Chỉ khi con trái từ một nhánh giữa / phải cần được kiểm tra, tập TL chỉ được cập nhật trên con giữa / phải mới được tạo ra. Tương tự như vậy, chỉ khi con giữa từ một nhánh bên phải cần được kiểm tra, thiết lập TM

chỉ được cập nhật trên con phải mới được tạo ra. Chúng ta sẽ ký hiệu tập TL (và TM) của nút O là TL_O (và TM_O).

❖ **Bộ đề 8 Left Track Checking:** Cho L=(H'\W,R',C') là con trái của nút O'(H',R',C') bởi lát cắt z = (W,X,Y). Nếu W ∩ $TL_{o'} \neq \emptyset$, L có thể được lược bỏ.

Ví dụ: Trong **Hình 3.6**, Con trái L(h_2h_3, $r_2r_3r_4$, $c_1c_2c_3c_4c_5$) (a1 ở mức 2) có cha là P($h_1h_2h_3$, $r_2r_3r_4$, $c_1c_2c_3c_4c_5$) (nút thứ 2 ở mức 1) bởi lát cắt (h_1, r_2, c_4c_5) bị lược bỏ vì W ∩ $TL_P = \{h1\} \neq \emptyset$.

❖ **Bộ đề 9 Middle Track Checking:** Cho M = (H',R'\X,C') là con giữa của nút O'(H',R',C') bởi lát cắt z = (W,X,Y). Nếu X ∩ $TM_{O'} \neq \emptyset$, M có thể được lược bỏ.

Ví dụ: Trong **Hình 3.6**, Con giữa M($h_1h_2h_3$, r_1r_3, $c_1c_2c_3$) (b1 ở mức 4) có cha là P($h_1h_2h_3$, $r_1r_2r_3$, $c_1c_2c_3$) (nút thứ 4 ở mức 3) bởi lát cắt (h_2, r_2, c_1c_5) bị lược bỏ vì X ∩ $TM_P = \{r2\} \neq \emptyset$.

Để loại bỏ các nút vô ích loại (c) và (d), chúng ta phát triển *Close Height Set Checking* trong **Bổ đề 10** và *Close Row Set Checking* trong **bổ đề 11**.

❖ **Bổ đề 10 Close Height Set Checking:** Cho O'' = (H'',R'',C'') là con giữa / phải của nút O' và Z là toàn bộ tập lát cắt. Nếu ∃H_w ∈ (H\H'')(H là tập chiều cao đầy đủ của O) như vậy ∀({h_w}, {r_x}, C_y) ∈ Z trong đó r_x ∈ R'', C'' ∩ Cy = ∅, khi đó O'' là hở trong tập chiều cao và có thể được lượt bỏ đi. Từ đó con trái không bao giờ thỏa mãn điều kiện, chỉ con giữa và phải cần kiểm tra này.

Ví dụ: Trong **Hình 3.6**, nút R(h_2h_3, $r_1r_2r_3$, c_2c_3) (c2 trong mức 7) không đóng trong tập chiều cao vì có h_1 ∈ (H\ {h_2h_3}), như vậy lát cắt (h_1, r_1, c_4) và (h_1,r_2,c_4c_5), {c_2,c_3} ∩ {c_4} = ∅ và {c_2,c_3} ∩ {c_4c_5} = ∅. Và chúng ta tìm thấy tập cha của c2 trong nút R($h_1h_2h_3$, $r_1r_2r_3$, c_2c_3)(nút thứ 5 ở mức 4).

❖ **Bổ đề 11 Close Row Set Checking:** Cho O'' = (H'',R'',C'') là con trái / phải của nút O' và Z là toàn bộ tập lát cắt. Nếu ∃r_x ∈ (R\R'')(R là tập dòng đầy đủ của O) như vậy ∀({h_w}, {r_x}, C_y) ∈ Z trong đó h_w ∈ H'', C'' ∩ Cy = ∅, khi đó O'' là hở trong tập dòng và có thể được lượt bỏ đi. Từ đó con phải không bao giờ thỏa mãn điều kiện, chỉ con trái và phải cần kiểm tra này.

Ví dụ: Trong **Hình 3.6**, nút R(h_2h_3, r_1r_4, $c_1c_2c_3$) (d2 trong mức 7) không đóng trong tập dòng vì có r_3 ∈ (R\ {r_1r_4}), như vậy lát cắt (h_2, r_3, c_5) và (h_3,r_3,c_5), {c_1, c_2,c_3} ∩ {c_5} = ∅. Và chúng ta tìm thấy tập cha của d2 trong nút R(h_2h_3, $r_1r_3r_4$, $c_1c_2c_3$)(nút thứ 3 ở mức 6).

3.2.4.2 Thuật toán CubeMiner.

CubeMiner sử dụng phương thức depth-first để khai phá FCCs. Algorithm 3 chứa code-pseudo của CubeMiner. Đầu tiên, kiểm tra TL, TM đã được khởi tạo bằng rỗng hay chưa và tập lát cắt Z đã được tính toán, sau đó sử dụng đệ quy hàm *cut()* trong Algorithm 6 được gọi.

Algorithm 3 *CubeMiner*

1: ***CubeMiner()***
2: Global variables: H the set of heights, R the set of rows, C the set of columns, monotonic constraints $minH$, $minR$, and $minC$ on H, R, C respectively.
3: Input: 3D Matrix O with l heights, n rows and m columns.
4: Output: ξ the set of FCCs.
5: $TL \leftarrow empty(), TM \leftarrow empty()$;
6: Z and $|Z|$ are computed from O;
7: $\xi \leftarrow cut((H, R, C), Z, 0, |Z|, TL, TM)$;

Algorithm 6 Cutting

1: **cut**$((H', R', C'), Z, 0, |Z|, TL, TM)$
2: Input: Node (H', R', C'), cutters list Z, iteration number i, $|Z|$ the size of Z, left and right atoms tracks TL and TM.
3: Output: ξ the set of FCCs.
4: $(W, X, Y) \leftarrow Z[i]$;
5: **if** $i \leq |Z| - 1$ **then**
6: **if** $W \cap H' = \emptyset$ or $X \cap R' = \emptyset$ or $Y \cap C' = \emptyset$ **then**
7: $\xi \leftarrow \xi \cup cut((H', R', C'), Z, i + 1, |Z|, TL, TM)$;
8: **else**
9: **if** $minH(H' \setminus W)$ satisfied and $W \cap TL = \emptyset$ **then**
10: $\beta \leftarrow Rcheck((H' \setminus W, R', C'), Z)$;
11: **if** $\beta = 1$ **then**
12: $\xi \leftarrow \xi \cup cut((H' \setminus W, R', C'), Z, i + 1, |Z|, TL, TM)$;
13: **end if**
14: **end if**
15: **if** $minR(R' \setminus X)$ satisfied and $X \cap TM = \emptyset$ **then**
16: $\alpha \leftarrow Hcheck((H', R' \setminus X, C'), Z)$;
17: **if** $\alpha = 1$ **then**
18: $\xi \leftarrow \xi \cup cut((H', R' \setminus X, C'), Z, i + 1, |Z|, TL \cup W, TM)$;
19: **end if**
20: **end if**
21: **if** $minC(C' \setminus Y)$ satisfied **then**
22: $\alpha \leftarrow Hcheck((H', R', C' \setminus Y), Z)$;
23: **if** $\alpha = 1$ **then**
24: $\beta \leftarrow Rcheck((H', R', C' \setminus Y), Z)$;
25: **if** $\beta = 1$ **then**
26: $\xi \leftarrow \xi \cup cut((H', R', C' \setminus Y), Z, i + 1, |Z|, TL \cup W, TM \cup X)$;
27: **end if**
28: **end if**
29: **end if**
30: **end if**
31: **else**
32: $\xi \leftarrow (H', R', C')$;
33: **end if**
34: **return** ξ;

Hàm *cut()* cắt nút O' = (H', R', C') với lát cắt đầu tiên Z[i] = (W, X, Y) điều đó thỏa mãn những ràng buộc sau đây. Đầu tiên, (H', R', C') phải giao với Z[i] khác rỗng, nếu không thỏa mãn trường hợp này, hàm *cut()* sẽ được gọi với lát cắt tiếp theo.

Để xây dựng con trái L = (H' \ W, R', C') (dòng 9-14), đòi hỏi 3 kiểm tra: kiểm tra minH(H'\W), *Left Track Check*, và *Close Row Set Check* (hàm Rcheck() trong **Algorithm 4**). Nếu L không bị lược bỏ bởi ba kiểm tra, hàm *cut()* được gọi để xử lý L, và không cập nhật tập TL và TM cho L.

Algorithm 4 Close Row Set Check

1: **Rcheck**$((H', R', C'), Z)$
2: Input: node (H', R', C') and cutters list Z.
3: Output: flag β.
4: **if** $\exists r_x \in (R \setminus R')$ such that $\forall(\{h_w\}, \{r_x\}, C_y) \in Z$ where $h_w \in H'$, $C' \cap C_y = \emptyset$ **then**
5: $\beta \leftarrow 0$;
6: **else**
7: $\beta \leftarrow 1$;
8: **end if**
9: **return** β;

Để xây dựng con giữa M = (H', R'\X, C')(dòng 15-20) đòi hỏi 3 kiểm tra: kiểm tra minR(R'\X), *Middle Track Check*, và *Close Height Set Check* (hàm Hcheck() trong **Algorithm 5**). Nếu M không bị lược bỏ bởi ba kiểm tra, hàm *cut()* được gọi để xử lý M, và tập TL cho L được cập nhật TL ∪ W.

Algorithm 5 Close Height Set Check

1: **Hcheck**$((H', R', C'), Z)$
2: Input: node (H', R', C') and cutters list Z.
3: Output: flag α.
4: **if** $\exists h_w \in (H \setminus H')$ such that $\forall(\{h_w\}, \{r_x\}, C_y) \in Z$ where $r_x \in R'$, $C' \cap C_y = \emptyset$ **then**
5: $\alpha \leftarrow 0$;
6: **else**
7: $\alpha \leftarrow 1$;
8: **end if**
9: **return** α;

Để xây dựng con phải R = (H', R', C'\Y)(dòng 21-29) đòi hỏi 3 kiểm tra: kiểm tra minC(C'\Y), *Close Row Set Check*, và *Close Height Set Check*. Nếu R không bị lược bỏ bởi ba kiểm tra, hàm *cut()* được gọi để xử lý R, và tập TL, TM cho L được cập nhật TL ∪ W, TM ∪ X.

Khi đó kích thước của Z và thứ tự sắp xếp các lát cắt trong Z quan trọng tới hiệu suất, thuật toán có thể được tối ưu hóa bằng tiền xử lý các tập dữ liệu 3D. Chúng ta áp dụng hai heuristics. Đầu tiên, chúng ta chuyển vị dữ liệu ma trận 3D để làm cho |H| < |C| và |R| < |C|, điều này giúp giảm thiểu kích thước của |Z|. Thứ hai, chúng ta sắp xếp những lát chiều cao có chứa nhiều giá trị 0 luôn luôn đứng trước những lát chiều cao có ít giá trị 0, điều này giúp đẩy nhanh quá trình khai phá bởi lược bớt không gian tìm kiếm càng sớm càng tốt.

3.2.4.3 Tính đúng đắn.

CubeMiner xây dựng gốc (H, R, C) và sau đó giảm đồng thời H, R, C để thu được tập các lá có nguồn gốc từ (H, R, C). **Định lý 4** cho thấy rằng CubeMiner có thể chính xác tạo ra tất cả và chỉ có tất cả các FCCs.

Định lý 4: Cho FCCs là tập khối thường xuyên đóng của bộ dữ liệu 3D. Cho LV là tập các nút là có nguồn gốc từ áp dụng CubeMiner trên tập dữ liệu. Khi đó, FCCs = LV. Nói cách khác, CubeMiner có thể chính xác tạo ra tất cả và chỉ có tất cả FCCs.

3.2.3 Khai phá FCC song song.

Cho rằng khai phá FCC tính toán mất nhiều thời gian, một giải pháp để giảm thời gian đáp ứng là khai thác song song. Trong phần này, chúng ta sẽ cho thấy làm thế nào RSM và CubeMiner có thể được dễ dàng khai phá song song.

Nói chung, một thuật toán song song thường bao gồm ba giai đoạn: (a) giai đoạn tạo ra các tác vụ con bằng cách tách tác vụ ban đầu thành các tác vụ nhỏ hơn, (b) giai đoạn phân bổ tác vụ đó là chỉ định các tác vụ con cho các bộ xử lý; (c) giai đoạn thi hành các tác vụ trong đó mỗi bộ xử lý có tác động trên tác vụ con được chỉ định. Một yếu tố quan trọng trong việc khai phá song song là để giảm thiểu sự trở ngại trong giai đoạn thi hành, để tất cả các bộ vi xử lý có thể hoạt động độc lập, đồng thời mà không cần phải giao tiếp với nhau.

Điều đó chỉ ra rằng cả hai thuật toán RSM và CubeMiner phù hợp một cách tuyệt vời với khung ở trên: các tác vụ có thể được tạo ra và giao cho các bộ xử lý để thực hiện độc lập.

- **Khai phá song song RSM:** Trong RSM, khai phá của mỗi lát đại diện tương ứng với một tác vụ, nói cách khác, số lượng tối đa các tác vụ là số bản liệt kê của các chiều cơ sở (những liệt kê mà không đáp ứng ngưỡng yêu cầu tối thiểu được loại bỏ). Mỗi tác vụ có thể được giao cho các bộ xử lý, và có thể được xử lý độc lập.
- **Khai phá song song CubeMiner:** Trong CubeMiner, mỗi nhánh của quá trình tách cây có thể được xử lý độc lập, và do đó, mỗi nhánh tương ứng với một tác vụ. Nói cách khác, chúng ta có thể phân bổ một nhánh của quá trình tách cây tới một bộ xử lý.

Đối với cả hai RSM và CubeMiner, để đảm bảo rằng các tác vụ có thể được xử lý độc lập, mỗi bộ xử lý đòi hỏi phải có một bản sao đầy đủ của bộ dữ liệu. Điều này là cần thiết để các giai đoạn cắt tỉa sau có thể được thực hiện độc lập. May mắn thay, việc đồng bộ trên (để truyền tải các bộ dữ liệu cho tất cả các bộ vi xử lý) không quan trọng: **(a)** bộ dữ liệu có thể được truyền đi trong khi tác vụ đang được tạo ra, vì vậy thời gian đáp ứng là không bị ảnh hưởng nhiều; **(b)** chi phí đồng bộ tương đối nhỏ so với chi phí khai phá.

3.2.4 Độ phức tạp thời gian.

Độ phức tạp thời gian của khai phá FCCs tính theo hàm số mũ số lượng các mẫu. Cho bộ dữ liệu 3D O = H x R x C, trong đó |H| = L, |R| = N, |C| = M, độ phức tạp thời gian của RSM và CubeMiner lần lượt là $O(2^{L+N} + N^2 + M)$ và $O(2^{LN}$ x M)(không áp dụng bất kỳ chiến lược cắt tỉa sau nào). Bằng cách áp dụng minH, minR, minC, các ràng buộc chặt chẽ và các chiến lược cắt tỉa sau, hiệu quả của RSM và CubeMiner có thể được cải thiện đáng kể.

3.3 Tóm tắt.

Trong chương này, chúng ta đã đề xuất một Khung mới cho khai phá FCP_S trên bộ dữ liệu dày đặc. Ý tưởng khung này là phân vùng tập dữ liệu ban đầu thành những không gian con như vậy việc khai phá những không gian con sẽ tạo ra những đáp án giống như khai phá từ không gian ban đầu. Căn cứ vào Khung này, chúng ta đề xuất hai thuật toán C-Miner, B-Miner khai phá cho bộ dữ liệu 2D và hai thuật toán RSM, CubeMiner cho bộ dữ liệu 3D.

Hai thuật toán C-Miner và B-Miner bao gồm 2 giai đoạn: phân vùng không gian khai phá và khai phá FCP từ các không gian con. Hai thuật toán áp dụng các cách phân vùng và chiến lược lược bỏ khác nhau.

Thuật toán RSM dựa trên ý tưởng cơ bản là chuyển đổi bộ dữ liệu 3D thành tập các bộ dữ liệu 2D, sau đó áp dụng một thuật toán khai phá FCP bất kỳ để khai phá các bộ dữ liệu 2D, cuối cùng tổng hợp các FCP 2D để thu được các FCC 3D.

Thuật toán CubeMiner khai phá FCC trực tiếp trên bộ dữ liệu 3D. Ý tưởng cơ bản là sử dụng các lát cắt và cây phân chia để thu được các FCC.

Ngoài ra chúng ta cũng cho thấy được làm thế nào khung có thể khai phá song song các FCP,FCC một cách đơn giản và hiệu quả

CHƯƠNG 4: CÀI ĐẶT THUẬT TOÁN THỬ NGHIỆM.

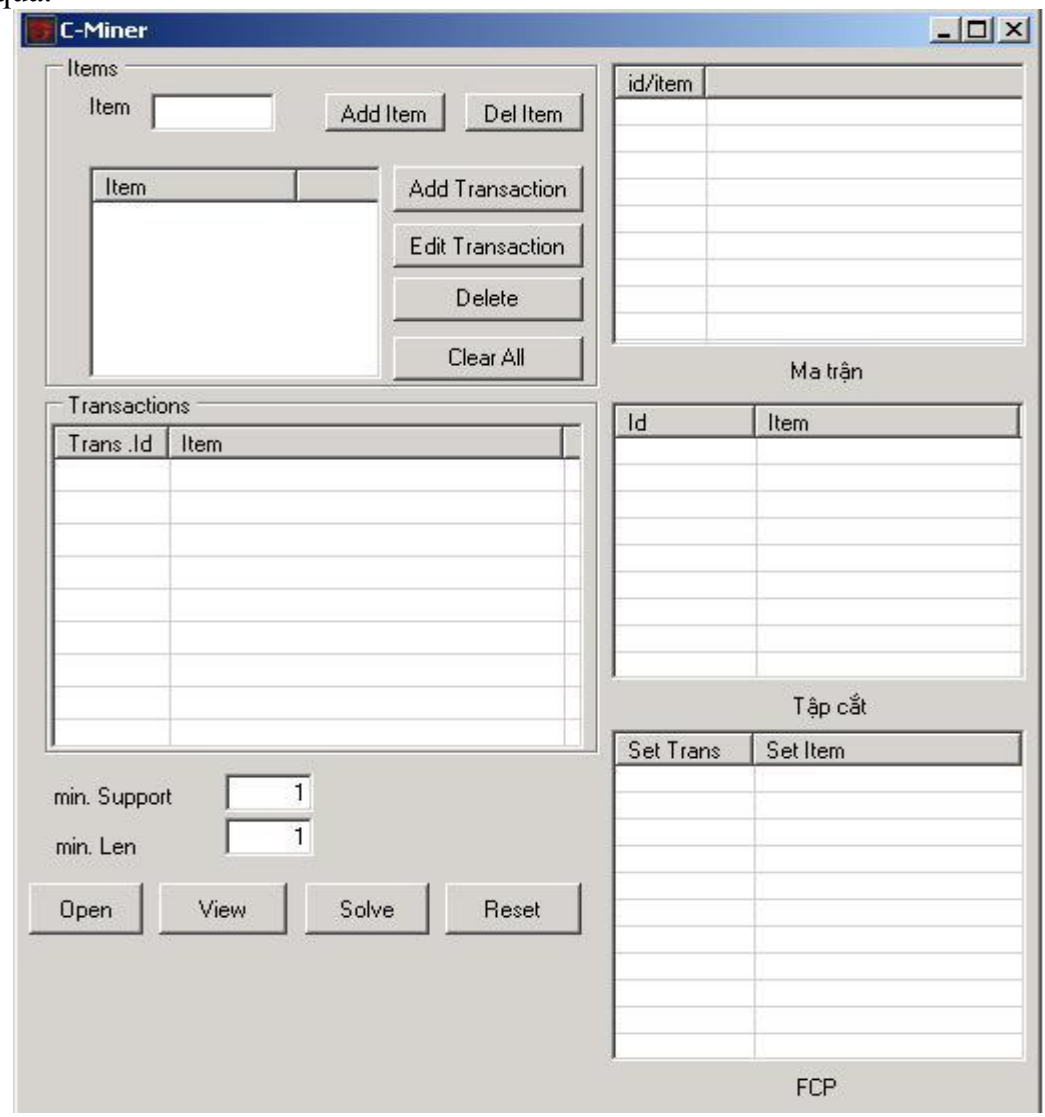
Vì thời gian tìm hiểu không nhiều, để minh họa cho các thuật toán đã được nêu ở trên. Em xin giới thiệu chương trình khai phá tập phổ biến đóng trong không gian 2 chiều áp dụng thuật toán C-Miner.

4.1 Giới thiệu về chương trình.

- Chương trình được xây dựng bằng ngôn ngữ VB.NET.
- Đầu vào là một bộ dữ liệu 2 chiều do người dùng thiết lập.
- Đầu ra sẽ là các tập phổ biến đóng trong không gian 2 chiều đã cho.

4.2 Giao diện chương trình.

Chương trình bao gồm một form chính: dùng để nhập dữ liệu , xử lý và hiển thị kết quả.



4.3 Các thành phần và chức năng trong chương trình.

- Group Items.

+ Ô textbox để nhập tên mục.
+ Nút Add Item để thêm một mục cho bộ dữ liệu.
+ Nút Del Item để xóa một mục trong bộ dữ liệu.
+ Một ListView để biểu diễn các tập mục trong bộ dữ liệu.
+ Lần lượt các nút add transaction, edit transaction, delete, clear all có chức năng thêm , sửa, xóa và clear các dòng trong bộ dữ liệu.

- Group Trấnctions gồm 1 ListView để biểu diễn các dòng trong bộ dữ liệu.
- 2 textbox để nhập giá trị min_support và min_len.
- Nút Open đề mở dữ có sẵn từ file txt.
- Nút View có chưc năng hiển thị Ma trận của bộ dữ liệu.
- Nút Solve Có chức năng thực hiện tính toán để tìm kiếm các FCP.
- Nút Reset có chức năng khỏi tạo lại bộ dữ liệu mới.
- 3 ListView bên phải lần lượt biểu diễn cho Ma trận dữ liệu, Tập cắt và Tập FCP khai phá được.

4.4 Kết quả thực nghiệm.

Sau khi chạy một số ví dụ trên các bộ dữ liệu khác nhau với các ngưỡng min_sup và min_len khác nhau. Ta thấy thuật toán C-Miner đa khai phá được chính xác tất cả các FCP và tất cả đều là FCP. Với bộ dữ liệu dạng lớn thì hiệu quả của C-Miner được cải thiện rất nhiều.

KẾT LUẬN.

Hiện nay, con người đang chìm ngập trong tri thức nhưng lại rất thiếu thốn thông tin, với lượng dữ liệu lớn và phức tạp như hiện nay thì nhu cầu khai phá tri thức trở nên rất thiết yếu với con người.

Đồ án đã giới thiệu được tổng quát về KPTT và KPDL, các hướng tiếp cận chính trong KPTT, các lĩnh vực ứng dụng KPTT trong thực tế. Ngoài ra đồ án còn đề cập đến một số phương pháp khai phá dữ liệu dạng đóng được ứng dụng trong nhiều lĩnh vực thực tế hiện nay (phân tích thị trường, phân tính sinh học,…). Cụ thể là các thuật toán C-Miner và B-Miner trong khai phá bộ dữ liệu 2 chiều, và RSM và CubeMiner trong khai phá bộ dữ liệu 3 chiều.

Hạn chễ: Vì thời gian tìm hiểu chưa được nhiều nên em mới chỉ xây dựng được một chương trình khai phá dữ liệu dựa trên thuật toán C-Miner với đầu vào là bộ dữ liệu dạng text.

Hướng đi tiếp theo:

Xây dựng chương trình thực nghiệm đối với các thuật toán B-Miner, RSM và CubeMiner đối với dữ liệu là các cơ sở dữ liệu thực tế.

TÀI LIỆU THAM KHẢO.

[1] Ji Liping (Bachelor of Management, Nanjing University, China): *Mining Localized*
co-expressed gene patterns from microarray data, A dissertation submitted for the degree of philosophy at national university of Singapore school of computing june 2006.
[2] Andrew Kusiak Intelligent Systems Laboratory 2139 Seamans Center The University of Iowa Iowa City, Iowa 52242 – 1527: *Association Rules The Apriori Algorithm.*
[3] Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U. and Hsu, M: *Frequent pattern-projected sequential pattern mining.*
[4] Nguyễn Đức Cường – Khoa Công Nghệ Thông Tin – Đại học Bách Khoa Thành Phố Hồ Chí Minh: *Tổng quan về khai phá dữ liệu*.