ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA TOÁN - CƠ - TIN HỌC

Fast Algorithms for Mining Association Rules

Chuyên đề 4

Ngành: **Khoa học dữ liệu**

(Chương trình đào tạo thạc sĩ)

Giảng viên hướng dẫn: PGS.TS Vũ Tiến Dũng

Nguyễn Thị Ngọc Uyên - 23007930
Lý Đức Trung - 23007933

Hà Nội - 2025

LỜI CẢM ƠN

Bài tiểu luận cuối kỳ này được hoàn thành dưới sự hướng dẫn của thầy PGS.TS. Vũ Tiến Dũng. Chúng em xin bày tỏ lòng biết ơn sâu sắc tới thầy vì sự định hướng và gợi ý đề tài trong quá trình học tập môn học này.

Trong quá trình làm bài tiểu luận này, do kiến thức còn hạn chế nên chúng em không tránh khỏi những thiếu sót. Chúng em mong nhận được những ý kiến đóng góp của thầy và các bạn để có thể rút ra kinh nghiệm, hoàn thiện và phát triển đề tài hơn.

Chúng em xin chân thành cảm ơn!

Mục lục

Giới thiệu

Trong bối cảnh các hệ thống thông tin hiện đại đang tạo ra khối lượng dữ liệu khổng lồ mỗi ngày, việc khai thác tri thức từ dữ liệu (data mining) đã trở thành một lĩnh vực nghiên cứu quan trọng và có nhiều ứng dụng thực tiễn. Trong số các kỹ thuật khai phá dữ liệu, khai phá luật kết hợp (association rule mining) là một trong những hướng tiếp cận được quan tâm nhiều nhất bởi khả năng phát hiện các mối quan hệ tiềm ẩn giữa các đối tượng dữ liệu. Những luật kết hợp này không chỉ hỗ trợ mô tả hành vi của người dùng và khách hàng, mà còn mang lại giá trị lớn trong các lĩnh vực như thương mại điện tử, tối ưu hoá bán lẻ, marketing, phân tích rủi ro tài chính và các hệ thống gợi ý thông minh.

Một bài toán trung tâm trong khai phá luật kết hợp là bài toán tìm tập mục thường xuyên (frequent itemset mining). Đây là bước tiền đề quan trọng để từ đó sinh ra các luật kết hợp có ý nghĩa. Tuy nhiên, do không gian tìm kiếm tăng theo cấp số mũ với số lượng mặt hàng và giao dịch, việc khai thác các tập mục thường xuyên đòi hỏi những thuật toán hiệu quả về thời gian và tài nguyên xử lý. Trong lĩnh vực này, ba thuật toán kinh điển và nền tảng được sử dụng rộng rãi nhất là Apriori, AprioriTid và AprioriHybrid. Các thuật toán này không chỉ đặt nền móng cho các phương pháp hiện đại mà còn thể hiện những chiến lược tối ưu hóa quan trọng nhằm giảm số lần quét cơ sở dữ liệu, giảm số lượng ứng viên cần xét và tận dụng tốt hơn tài nguyên bộ nhớ.

Thuật toán **Apriori** giới thiệu một thuộc tính quan trọng cho phép giảm mạnh số lượng tập ứng viên, dựa trên nguyên tắc rằng một tập mục chỉ có thể là phổ biến khi tất cả các tập con của nó cũng phổ biến. Thuật toán **AprioriTid** cải thiện hiệu năng ở các vòng lặp sau bằng cách sử dụng một cấu trúc dữ liệu trung gian, tránh việc phải quét lại toàn bộ cơ sở dữ liệu. Trong khi đó, **AprioriHybrid** kết hợp ưu điểm của cả Apriori và AprioriTid, nhờ đó đạt được hiệu suất tối ưu hơn trong các tình huống thực tế.

Việc nghiên cứu ba thuật toán này không chỉ giúp làm rõ các nguyên lý nền tảng của khai

phá luật kết hợp, mà còn cho thấy cách những chiến lược tối ưu hóa khác nhau ảnh hưởng trực tiếp đến hiệu năng thực thi của thuật toán.

Tiểu luận này được thực hiện nhằm mục tiêu:

- Trình bày tổng quan về khai phá dữ liệu và khai phá luật kết hợp.
- Phân tích chi tiết ba thuật toán Apriori, AprioriTid và AprioriHybrid.
- Minh họa hoạt động của các thuật toán thông qua ví dụ cụ thể.
- Đánh giá và so sánh hiệu năng của các thuật toán trên các tiêu chí lý thuyết.
- Rút ra những nhận xét và định hướng áp dụng phù hợp trong thực tế.

Phạm vi tiểu luận tập trung vào khía cạnh thuật toán và hiệu năng, không đi sâu vào các mở rộng hoặc biến thể hiện đại như FP-Growth, ECLAT hoặc các phương pháp dựa trên cây tiền tố và đồ thị. Tuy nhiên, những thuật toán cổ điển được phân tích trong báo cáo này vẫn giữ vai trò quan trọng trong việc hiểu rõ tư duy thiết kế thuật toán khai phá dữ liệu.

Bố cục của tiểu luận được tổ chức như sau: Chương 1 giới thiệu tổng quan về khai phá dữ liệu và luật kết hợp. Chương 2, Chương 3 và Chương 4 lần lượt phân tích ba thuật toán Apriori, AprioriTid và AprioriHybrid. Chương 5 so sánh hiệu năng các thuật toán và thảo luận các yếu tố ảnh hưởng. Cuối cùng là phần kết luận tổng hợp nội dung và đề xuất hướng phát triển tiếp theo.

Chương 1

Tổng quan về khai phá dữ liệu và luật kết hợp

1.1 Tổng quan về khai phá dữ liệu

1.1.1 Khái niệm khai phá dữ liệu

Khai phá dữ liệu (Data Mining) là quá trình phát hiện các mẫu, quy luật hoặc tri thức tiềm ẩn trong các tập dữ liệu lớn, phức tạp và thường khó phân tích bằng các phương pháp thống kê truyền thống. Đây là một giai đoạn quan trọng trong quy trình khám phá tri thức từ dữ liệu (Knowledge Discovery in Databases – KDD), bao gồm các bước như làm sạch dữ liệu, tích hợp, lựa chọn dữ liệu, khai phá và đánh giá tri thức. Khai phá dữ liệu kết hợp nhiều kỹ thuật từ thống kê, học máy, hệ cơ sở dữ liệu và trí tuệ nhân tạo nhằm rút ra thông tin hữu ích phục vụ cho việc ra quyết định.

1.1.2 Vai trò của khai phá dữ liệu

Trong thời đại dữ liệu lớn (Big Data), các tổ chức và doanh nghiệp thu thập dữ liệu từ nhiều nguồn khác nhau như giao dịch thương mại điện tử, mạng xã hội, hệ thống cảm biến hay dữ liệu tài chính. Việc phân tích dữ liệu này mang lại nhiều giá trị:

- Hỗ trợ ra quyết định chiến lược dựa trên bằng chứng.
- Dự đoán hành vi khách hàng hoặc xu hướng thị trường.
- Tối ưu hóa hoạt động kinh doanh và giảm thiểu rủi ro.

- Phát hiện các bất thường, gian lận hoặc rủi ro tiềm ẩn.

Nhờ những lợi ích này, khai phá dữ liệu trở thành một trong những lĩnh vực cốt lõi của khoa học dữ liệu và công nghệ thông tin.

1.1.3 Các nhiệm vụ chính trong khai phá dữ liệu

Khai phá dữ liệu bao gồm nhiều nhóm nhiệm vụ khác nhau, trong đó phổ biến nhất là:

- **Phân lớp (Classification):** Dự đoán nhãn lớp cho dữ liệu mới dựa trên mô hình học từ dữ liệu quá khứ.
- **Phân cụm (Clustering):** Nhóm các đối tượng tương đồng lại với nhau dựa trên khoảng cách hoặc đặc trưng.
- **Phát hiện bất thường (Anomaly Detection):** Tìm ra các điểm dữ liệu bất thường trong tập hợp lớn.
- **Khai phá luật kết hợp (Association Rule Mining):** Tìm các mẫu quan hệ dạng "nếu–thì" giữa các mục xuất hiện trong dữ liệu.

Trong tiểu luận này, chúng ta tập trung vào nhiệm vụ khai phá luật kết hợp, một phương pháp quan trọng và có nhiều ứng dụng trong thực tế.

1.2 Khai phá luật kết hợp

1.2.1 Mục đích của luật kết hợp

Khai phá luật kết hợp nhằm tìm ra những quan hệ thường xuyên xuất hiện giữa các đối tượng trong dữ liệu. Một luật kết hợp có dạng:

$$X \Rightarrow Y$$

trong đó X và Y là hai tập mục không giao nhau. Luật được hiểu là: khi các mục trong X xuất hiện trong một giao dịch, thì các mục trong Y cũng có khả năng xuất hiện theo.

Mục tiêu chính của khai phá luật kết hợp là phát hiện các mẫu có ý nghĩa, giúp mô tả hành vi người dùng hoặc xu hướng đồng xuất hiện trong dữ liệu.

1.2.2 Ứng dụng trong thực tế

Luật kết hợp được sử dụng rộng rãi trong nhiều lĩnh vực:

- **Bán lẻ và thương mại điện tử:** gợi ý sản phẩm, phân tích giỏ hàng, bố trí hàng hóa.
- **Ngân hàng và tài chính:** phát hiện hành vi gian lận hoặc các mẫu giao dịch bất thường.
- **Marketing:** phân tích chiến dịch quảng cáo, xác định nhóm khách hàng tiềm năng.
- **Y tế:** phát hiện mối liên hệ giữa triệu chứng và bệnh lý.

Nhờ khả năng diễn giải trực quan, luật kết hợp đặc biệt hữu ích trong các hệ thống gợi ý và phân tích hành vi tiêu dùng.

1.2.3 Ý nghĩa trong hệ thống gợi ý

Trong thương mại điện tử, luật kết hợp đóng vai trò quan trọng trong việc xây dựng các mô hình gợi ý kiểu "Frequently Bought Together". Các luật như:

$$\text{(Laptop)} \Rightarrow \text{(Chuột không dây)}$$

giúp hệ thống tự động đề xuất sản phẩm phù hợp, gia tăng doanh thu và cải thiện trải nghiệm người dùng.

1.3 Mô hình dữ liệu giỏ hàng (Basket Data)

1.3.1 Khái niệm và cấu trúc dữ liệu

Dữ liệu giỏ hàng (Basket Data) là mô hình dữ liệu phổ biến trong phân tích giao dịch, trong đó mỗi giao dịch là một tập các mục được mua cùng nhau. Ký hiệu:

$$D = \{T_1, T_2, ..., T_m\}, \quad T_i \subseteq I$$

với I là tập tất cả các mặt hàng.

1.3.2 Đặc trưng của dữ liệu giỏ hàng

Dữ liệu giỏ hàng có các đặc điểm:

- Dữ liệu thường rất lớn và thưa.
- Mỗi giao dịch chứa ít mục so với kích thước toàn bộ tập mục.
- Các mục có xu hướng đồng xuất hiện theo nhóm.

Những đặc điểm này ảnh hưởng trực tiếp đến thiết kế thuật toán tìm tập mục thường xuyên.

1.3.3 Ví dụ minh hoạ dữ liệu giao dịch

Một ví dụ đơn giản về dữ liệu giỏ hàng:

TID	Mục hàng
1	Bread, Milk, Butter
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Cola
4	Bread, Milk, Diaper, Beer

Ví dụ này cho thấy các mục như *Bread*, *Milk*, *Diaper* thường xuất hiện cùng nhau, là cơ sở để sinh ra các luật kết hợp.

1.4 Các khái niệm cơ bản

1.4.1 Độ hỗ trợ (Support)

Độ hỗ trợ phản ánh mức độ phổ biến của luật trong toàn bộ cơ sở dữ liệu giao dịch. Nó được định nghĩa là tỷ lệ phần trăm các giao dịch chứa đồng thời cả X và Y:

$$\text{support}(X \Rightarrow Y) = \frac{\text{count}(X \cup Y)}{|D|},$$

với $|D|$ là tổng số giao dịch.

Độ hỗ trợ càng cao nghĩa là luật xuất hiện thường xuyên trong dữ liệu, do đó có giá trị thực tiễn lớn hơn. Với các tập dữ liệu rất lớn, tiêu chí này giúp loại bỏ những luật chỉ xuất hiện hiếm hoi hoặc mang tính bất thường.

1.4.2 Độ tin cậy (Confidence)

Độ tin cậy là thước đo phản ánh mức độ chính xác của luật kết hợp. Nó cho biết trong số các giao dịch có chứa toàn bộ các mặt hàng trong X, có bao nhiêu phần trăm giao dịch đồng

thời chứa các mặt hàng trong Y. Độ tin cậy của luật $X \Rightarrow Y$ được định nghĩa:

$$\text{confidence}(X \Rightarrow Y) = \frac{\text{count}(X \cup Y)}{\text{count}(X)},$$

trong đó $\text{count}(Z)$ là số lượng giao dịch chứa tập Z.

Độ tin cậy cao thể hiện rằng mối quan hệ giữa X và Y là mạnh và đáng tin cậy. Đây là một trong hai tiêu chí quan trọng nhất để đánh giá chất lượng của một luật kết hợp.

1.4.3 Mục tiêu của khai phá luật kết hợp

Khai phá luật kết hợp hướng đến việc tìm ra tất cả các luật có ý nghĩa, đồng thời đảm bảo luật đủ phổ biến và đủ chính xác. Vì vậy, hai ngưỡng quan trọng được đặt ra trong quá trình này là:

- `minsup`: độ hỗ trợ tối thiểu,
- `minconf`: độ tin cậy tối thiểu.

Một luật chỉ được giữ lại nếu thỏa mãn cả hai ngưỡng trên. Điều này giúp giảm đáng kể số lượng luật không thực sự hữu ích, đồng thời tập trung vào những luật phản ánh hành vi phổ biến của người dùng.

1.5 Các bài toán con trong khai phá luật kết hợp

Bài toán khai phá luật kết hợp thường được chia thành hai phần độc lập:

1. **Tìm tất cả các tập mặt hàng có độ hỗ trợ không nhỏ hơn minsup** Các tập này được gọi là *tập mục lớn* (large itemsets) hay *tập mục thường xuyên*. Việc tìm được các tập này là bước quan trọng nhất, bởi toàn bộ luật kết hợp về sau đều được suy ra từ chúng.
2. **Sinh luật kết hợp từ các tập mục lớn** Sau khi có các tập mục lớn, ta xét các tập con của từng tập mục để tạo thành các luật ứng viên, sau đó tính độ tin cậy để lọc ra các luật đạt minconf.

Trong nghiên cứu và trong bài tiểu luận này, trọng tâm được đặt vào bài toán thứ nhất—tìm tập mục thường xuyên—vì đây là phần tốn nhiều chi phí tính toán và là động lực cho sự ra đời của các thuật toán nổi tiếng như Apriori, AprioriTid và AprioriHybrid. 1

Chương 2

Thuật toán Apriori

2.1 Giới thiệu

Thuật toán Apriori là một trong những phương pháp quan trọng nhất trong khai phá dữ liệu để tìm các tập mục thường xuyên (frequent itemsets) và sinh luật kết hợp. Mặc dù ra đời từ rất sớm, Apriori vẫn là nền tảng cho nhiều thuật toán hiện đại nhờ vào cơ chế sinh ứng viên theo từng mức và khả năng loại bỏ mạnh mẽ các tổ hợp không cần thiết. Ý tưởng cốt lõi của Apriori dựa trên một tính chất đặc biệt liên quan đến mối quan hệ giữa một tập và các tập con của nó. Tính chất này giúp giảm đáng kể số lượng ứng viên, khiến cho bài toán trở nên khả thi ngay cả với tập dữ liệu có kích thước lớn.

Chương này trình bày chi tiết nguyên lý, quy trình hoạt động, cách sinh ứng viên, cơ chế loại bỏ (prune), đồng thời cung cấp một ví dụ minh họa cụ thể dựa trên nội dung các slide đã cho.

2.2 Nguyên lý cốt lõi

2.2.1 Tính chất cơ bản của Apriori

Tính chất quan trọng nhất của thuật toán Apriori được phát biểu bằng lời như sau:

> Nếu một tập mục kích thước k là một tập mục phổ biến (large itemset), thì mọi tập con có kích thước k-1 của nó cũng phải là tập mục phổ biến.

Tính chất này còn được gọi là nguyên lý "đóng xuống" (downward closure). Điều này có

một hệ quả quan trọng:

- Nếu một tập con bất kỳ có kích thước k-1 không phải là tập phổ biến, thì mọi tập có kích thước k chứa tập con đó chắc chắn không phải là tập phổ biến.

Nhờ đó, thuật toán có thể loại bỏ một lượng rất lớn các ứng viên không cần thiết ngay từ đầu, thay vì phải tốn công đếm độ hỗ trợ trong cơ sở dữ liệu.

2.2.2 Ý nghĩa của nguyên lý

Dựa trên nguyên lý ở trên, Apriori có thể:

- giảm số ứng viên cần xét trong mỗi vòng lặp,
- tránh xét các tập mục có kích thước lớn khi tập con của chúng đã không đạt ngưỡng hỗ trợ tối thiểu,
- giúp việc sinh frequent itemsets theo mức (level-wise) trở nên khả thi.

Nhờ đó, mặc dù cần quét cơ sở dữ liệu nhiều lần, Apriori vẫn đạt hiệu quả tốt nhờ giảm mạnh không gian tìm kiếm.

2.3 Quy trình hoạt động của Apriori

Thuật toán Apriori hoạt động theo phương pháp tìm kiếm theo từng mức. Cụ thể:

1. Quét cơ sở dữ liệu để tìm tất cả các tập 1-itemset có độ hỗ trợ không nhỏ hơn ngưỡng tối thiểu. Đây là tập L1.
2. Ở mỗi vòng lặp tiếp theo, từ tập L(k-1), thuật toán sinh ra tập ứng viên Ck bằng cách kết hợp các phần tử trong L(k-1).
3. Quét cơ sở dữ liệu và đếm độ hỗ trợ cho từng ứng viên trong Ck.
4. Tập Lk gồm tất cả các ứng viên trong Ck có độ hỗ trợ đạt ngưỡng tối thiểu.
5. Thuật toán dừng khi Lk rỗng.

Kết quả cuối cùng là hợp của tất cả Lk được sinh ra trong quá trình thực thi.

2.4 Giả mã thuật toán

Dưới đây là giả mã tương đương với slide, được viết hoàn toàn bằng văn bản thuần:

```
1.  L1 = tập các 1-itemset phổ biến.
2.  Với k = 2; khi L(k-1) không rỗng; tăng k lên:
3.      Sinh tập ứng viên Ck từ L(k-1) bằng hàm apriori-gen.
4.      Với mỗi giao dịch t trong cơ sở dữ liệu:
5.          Xác định Ct là tập các ứng viên trong Ck xuất hiện trong t.
6.          Với mỗi ứng viên c trong Ct:
7.              Tăng bộ đếm số lần xuất hiện của c.
8.      Lk = tập các ứng viên trong Ck có độ hỗ trợ không nhỏ hơn minsup.
9.  Hợp tất cả các Lk là kết quả cần tìm.
```

2.5 Sinh tập ứng viên

Quá trình sinh Ck từ L(k-1) gồm hai bước chính: bước nối (join) và bước loại bỏ (prune).

2.5.1 Bước Join

Hai phần tử p và q trong L(k-1) được nối với nhau để tạo thành một ứng viên mới c thuộc Ck nếu:

- p và q trùng nhau ở k-2 phần tử đầu tiên,
- phần tử cuối của p có giá trị nhỏ hơn phần tử cuối của q (để tránh trùng lặp).

Mô tả join trong slide:

```
Chèn vào Ck:
Chọn p.item1, p.item2, ..., p.item(k-1), q.item(k-1)
từ L(k-1) p và L(k-1) q
với điều kiện:
  p.item1 = q.item1, ..., p.item(k-2) = q.item(k-2)
  và p.item(k-1) < q.item(k-1)
```

2.5.2 Bước Prune

Tập ứng viên được tạo ra sẽ tiếp tục được lọc nhờ tính chất Apriori. Một ứng viên c trong Ck sẽ bị loại bỏ nếu tồn tại bất kỳ tập con có kích thước k-1 của c không nằm trong L(k-1).

Mô tả prune:

```
Với mỗi itemset c trong Ck:
    Với mỗi tập con s của c có kích thước k-1:
        Nếu s không nằm trong L(k-1), loại c khỏi Ck.
```

Đây là một trong những bước giúp Apriori vận hành hiệu quả.

2.6 Ví dụ minh họa

Xét cơ sở dữ liệu giao dịch sau:

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Giả sử ngưỡng hỗ trợ tối thiểu minsup là 0.5, tương đương cần xuất hiện ít nhất 2 lần trong 4 giao dịch.

2.6.1 Bước 1: Tìm frequent 1-itemsets

Đếm số lần xuất hiện:

- Item 1: 2 lần
- Item 2: 3 lần
- Item 3: 3 lần
- Item 4: 1 lần (loại)
- Item 5: 3 lần

Do đó L1 gồm: {1}, {2}, {3}, {5}.

2.6.2 Bước 2: Sinh frequent 2-itemsets

Join L1 để tạo các ứng viên 2-itemset:

- {1,2}, {1,3}, {1,5}
- {2,3}, {2,5}
- {3,5}

Đếm số lần xuất hiện:

- {1,2}: 1 lần (loại)
- {1,3}: 2 lần
- {1,5}: 1 lần (loại)
- {2,3}: 2 lần
- {2,5}: 3 lần
- {3,5}: 2 lần

Do đó L2 gồm: {1,3}, {2,3}, {2,5}, {3,5}.

2.6.3 Bước 3: Sinh frequent 3-itemsets

Join L2 tạo ứng viên:

- {1,2,3}
- {1,2,5}
- {1,3,5}
- {2,3,5}

Prune:

- {1,2,3}: loại vì {1,2} không thuộc L2
- {1,2,5}: loại vì {1,2} không thuộc L2
- {1,3,5}: loại vì {1,5} không thuộc L2
- {2,3,5}: hợp lệ

Đếm:

{2,3,5}: xuất hiện 2 lần nên thuộc L3.

2.6.4 Kết quả

Frequent itemsets thu được:

- Kích thước 1: {1}, {2}, {3}, {5}
- Kích thước 2: {1,3}, {2,3}, {2,5}, {3,5}
- Kích thước 3: {2,3,5}

2.7 Kết luận

Thuật toán Apriori là một phương pháp kinh điển trong khai phá dữ liệu nhờ dựa trên tính chất mạnh mẽ về quan hệ giữa tập và tập con. Mặc dù phải quét dữ liệu nhiều lần, Apriori vẫn hoạt động hiệu quả trong nhiều trường hợp nhờ cơ chế sinh ứng viên theo từng mức và khả năng loại bỏ các ứng viên không cần thiết ngay từ khi sinh ra. Các thuật toán cải tiến như AprioriTid và AprioriHybrid được xây dựng dựa trên Apriori nhằm giảm số lần quét cơ sở dữ liệu và rút ngắn thời gian xử lý.

Chương 3

Thuật toán AprioriTid

3.1 Giới thiệu

AprioriTid là một biến thể quan trọng của thuật toán Apriori truyền thống, được thiết kế nhằm giảm chi phí đọc lại toàn bộ cơ sở dữ liệu trong các vòng lặp sau mức 1. Điểm khác biệt chính giữa hai thuật toán nằm ở cách tính độ hỗ trợ (support) của các ứng viên. Trong khi Apriori phải truy cập lại toàn bộ tập giao dịch D ở mỗi mức k để xác định xem các ứng viên có xuất hiện trong giao dịch hay không, AprioriTid sử dụng một cấu trúc dữ liệu trung gian để đại diện cho thông tin cần thiết. Nhờ đó, khi mức k tăng lên (tức các ứng viên lớn dần), AprioriTid thường có hiệu năng tốt hơn.

3.2 Ý tưởng và nguyên lý hoạt động

AprioriTid vẫn sử dụng hàm sinh ứng viên apriori-gen giống Apriori. Tuy nhiên, thay vì quét cơ sở dữ liệu gốc D để đếm support, thuật toán duy trì một bảng trung gian Ck_bar, trong đó mỗi dòng ứng với một giao dịch và chứa danh sách các ứng viên có khả năng xuất hiện trong giao dịch đó.

Một phần tử của Ck_bar có dạng:

```
< TID , list_of_candidate_itemsets >
```

Trong đó:

- TID là mã giao dịch.

- list_of_candidate_itemsets là danh sách các ứng viên k-itemset có thể tồn tại trong giao dịch tương ứng.

Điểm mạnh của cấu trúc này là:

- Không cần đọc lại toàn bộ dữ liệu D khi k >= 2.
- Khi k tăng, số lượng ứng viên trong mỗi giao dịch giảm mạnh, khiến kích thước Ck_bar ngày càng nhỏ.
- Thời gian đếm support giảm đáng kể.

3.3 Giả mã thuật toán AprioriTid

Dựa trên slide, giả mã thuật toán được viết lại như sau:

```
1.  L1 = tập các frequent 1-itemset.
2.  C1_bar = cơ sở dữ liệu D ban đầu.
3.  For k = 2; khi L(k-1) không rỗng; tăng k lên:
4.      Ck = apriori-gen( L(k-1) ).   // Sinh ứng viên mới
5.      Ck_bar = rỗng.
6.      Với mỗi dòng t trong C(k-1)_bar:
7.          Xác định tập Cprime gồm các ứng viên trong Ck xuất hiện
                trong t (dựa trên tập các item cấp thấp hơn).
8.          Với mỗi c trong Cprime:
9.              Tăng bộ đếm c.count lên 1.
10.             Nếu Cprime không rỗng, đưa
                < TID , Cprime > vào Ck_bar.
11.     Lk = các ứng viên trong Ck có c.count >= minsup.
12. Trả về hợp của tất cả Lk.
```

Khác với Apriori:

- Ở mức k = 1, AprioriTid giống hệt Apriori.
- Từ mức k = 2 trở đi, AprioriTid không sử dụng tập giao dịch D nữa mà chỉ dùng C(k-1)_bar.

3.4 Ví dụ minh họa

Xét tập giao dịch sau:

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Giả sử ngưỡng hỗ trợ tối thiểu minsup = 2.

3.4.1 Bước 1: Tìm L1 và C1_bar

Đếm support từng item:

- {1}: xuất hiện 2 lần
- {2}: 3 lần
- {3}: 3 lần
- {4}: 1 lần (loại)
- {5}: 3 lần

Do đó:

- L1 = {1}, {2}, {3}, {5}

C1_bar chính là dữ liệu gốc, nhưng chuyển thành dạng:

```
< TID , set-of-1-itemsets >
```

Ví dụ:

- TID 100: {1}, {3}, {4}
- TID 200: {2}, {3}, {5}
- TID 300: {1}, {2}, {3}, {5}
- TID 400: {2}, {5}

3.4.2 Bước 2: Sinh C2 và tính L2

Từ L1, sinh C2 gồm các tập 2-itemset:

- {1 2}, {1 3}, {1 5}
- {2 3}, {2 5}
- {3 5}

Dựa trên C1_bar, ta xác định các ứng viên xuất hiện trong từng giao dịch:

- TID 100: {1 3}
- TID 200: {2 3}, {2 5}, {3 5}
- TID 300: tất cả trừ {1 2} và {1 5}
- TID 400: {2 5}

Đếm support:

- {1 3}: 2 lần
- {2 3}: 2 lần
- {2 5}: 3 lần
- {3 5}: 2 lần
- Các ứng viên còn lại: 1 lần hoặc 0

Do đó:

- L2 = {1 3}, {2 3}, {2 5}, {3 5}

C2_bar được xây dựng tương tự, ví dụ:

- TID 100: {1 3}
- TID 200: {2 3}, {2 5}, {3 5}
- TID 300: {1 3}, {2 3}, {2 5}, {3 5}
- TID 400: {2 5}

3.4.3 Bước 3: Sinh C3 và tính L3

Join L2 để sinh các ứng viên 3-itemset:

- {2 3 5}

Sử dụng C2_bar:

- TID 200: có {2 3} và {2 5} và {3 5}, nên chứa {2 3 5}
- TID 300: tương tự
- TID 100 và 400: không chứa đủ bộ 2-itemset cần thiết

Đếm support:

- {2 3 5}: 2 lần

Suy ra:

- L3 = {2 3 5}

C3_bar chỉ còn:

- TID 200: {2 3 5}
- TID 300: {2 3 5}

3.4.4 Kết quả cuối cùng

Các frequent itemsets của AprioriTid:

- L1: {1}, {2}, {3}, {5}
- L2: {1 3}, {2 3}, {2 5}, {3 5}
- L3: {2 3 5}

Tập kết quả trùng hoàn toàn với Apriori truyền thống, nhưng số lần truy cập cơ sở dữ liệu giảm đáng kể.

3.5 Nhận xét về thuật toán AprioriTid

Ưu điểm:

- Không cần đọc lại tập giao dịch gốc khi k >= 2.
- Kích thước bảng Ck_bar giảm dần khi k tăng, do các ứng viên lớn thường hiếm.
- Xử lý nhanh hơn Apriori trên các tập dữ liệu lớn hoặc thưa.

Hạn chế:

- Ở mức k = 1, AprioriTid giống Apriori nên không có lợi thế.
- Nếu dữ liệu có nhiều giao dịch rất lớn, kích thước Ck_bar có thể lớn ở các mức đầu.

3.6 Kết luận

AprioriTid là một cải tiến quan trọng của Apriori, đặc biệt hữu ích khi dữ liệu lớn và các tập ứng viên kích thước cao trở nên hiếm. Việc lưu trữ và sử dụng bảng trung gian Ck_bar giúp giảm đáng kể chi phí quét dữ liệu và mang lại hiệu quả cao hơn so với Apriori truyền thống ở các mức k lớn.

Chương 4

Thuật toán AprioriHybrid

4.1 Giới thiệu

AprioriHybrid là sự kết hợp giữa hai thuật toán Apriori và AprioriTid nhằm tận dụng ưu điểm của cả hai. Các quan sát thực nghiệm cho thấy rằng Apriori hoạt động tốt trong các vòng lặp đầu tiên, trong khi AprioriTid trở nên hiệu quả hơn khi kích thước ứng viên tăng và bảng trung gian Ck_bar trở nên nhỏ hơn so với cơ sở dữ liệu ban đầu. AprioriHybrid được thiết kế để tự động chuyển đổi từ Apriori sang AprioriTid tại thời điểm thích hợp, từ đó cải thiện hiệu năng tổng thể của quá trình khai phá tập mục thường xuyên.

4.2 Ý tưởng của AprioriHybrid

AprioriHybrid kết hợp hai thuật toán theo chiến lược sau:

- Sử dụng Apriori trong các vòng lặp đầu tiên.
- Khi dự đoán rằng bảng Ck_bar (nếu sử dụng AprioriTid) có thể nằm vừa trong bộ nhớ, thuật toán chuyển sang AprioriTid.

Chiến lược này tận dụng ưu điểm:

- Apriori hiệu quả khi số lượng ứng viên còn lớn.
- AprioriTid hiệu quả khi ứng viên ít và có thể lưu trữ trong bộ nhớ.

4.3 Quy trình tổng quát của AprioriHybrid

Mặc dù không có giả mã chính thức tuyệt đối, quy trình kết hợp được mô tả như sau:

1. Chạy Apriori từ mức 1 và tăng dần k.
2. Ở mỗi mức k, ước lượng kích thước của bảng Ck_bar nếu áp dụng AprioriTid.
3. Nếu kích thước dự kiến của Ck_bar nhỏ hơn hoặc bằng lượng bộ nhớ cho phép:
    - Chuyển sang AprioriTid từ mức hiện tại.
    - Tiếp tục chạy AprioriTid đến hết thuật toán.
4. Ngược lại, tiếp tục sử dụng Apriori.

Không cần thiết phải chuyển ngược lại thành Apriori. Một khi đã chuyển sang AprioriTid, hiệu năng về sau luôn tốt hơn.

4.4 Lợi ích của việc kết hợp

AprioriHybrid thường vượt trội hơn Apriori trong hầu hết các trường hợp nhờ:

- Tận dụng khả năng xử lý nhanh của Apriori ở mức thấp.
- Giảm mạnh chi phí quét dữ liệu khi chuyển sang AprioriTid.
- Khi kích thước Ck_bar giảm dần theo k, thuật toán hưởng lợi từ hiệu năng tăng dần theo thời gian.

Theo các kết quả thực nghiệm, hiệu năng cải thiện nhiều nhất khi kích thước Ck_bar giảm từ từ theo từng mức. Khi điều này xảy ra:

- AprioriTid được kích hoạt sớm.
- Chi phí đọc đĩa giảm mạnh.
- Thời gian thực thi tổng thể giảm đáng kể.

Ngược lại, nếu kích thước Ck_bar giảm đột ngột hoặc không giảm đều, thời điểm chuyển đổi có thể đến muộn hơn, nhưng hiệu năng tổng thể vẫn tốt hơn Apriori.

4.5 Phân tích ưu điểm và hạn chế

4.5.1 Ưu điểm

- Tối ưu hóa tự động theo từng mức, phù hợp cho dữ liệu lớn.
- Tiết kiệm thời gian quét dữ liệu khi mức k cao.
- Giảm chi phí truy cập đĩa khi Ck_bar vừa bộ nhớ.
- Thường nhanh hơn cả Apriori và AprioriTid trong thực tế.

4.5.2 Hạn chế

- Cần ước lượng kích thước Ck_bar, yêu cầu thêm logic trong cài đặt.
- Trong trường hợp dữ liệu rất nhỏ, lợi ích có thể không rõ ràng.
- Khi dữ liệu có nhiều giao dịch rất lớn, Ck_bar ban đầu có thể quá lớn để việc chuyển đổi diễn ra sớm.

Chương 5

Đánh giá hiệu năng và so sánh các thuật toán

5.1 Các yếu tố ảnh hưởng đến hiệu năng

Hiệu năng của các thuật toán khai thác tập mục thường xuyên phụ thuộc mạnh vào đặc trưng dữ liệu và một số tham số điều khiển thuật toán. Dưới đây là những yếu tố quan trọng nhất:

1. Số lượng giao dịch (|D|)

Tập giao dịch càng lớn thì thời gian quét CSDL càng tăng. Đối với Apriori, số lần quét CSDL theo từng vòng lặp làm chi phí tăng tuyến tính theo |D|.

2. Số lượng mặt hàng (|I|)

Khi |I| lớn, không gian ứng viên có thể bùng nổ (combinatorial explosion), làm số lượng ứng viên tăng theo cấp số nhân. Điều này ảnh hưởng trực tiếp đến Apriori và AprioriTid.

3. Độ dài trung bình giao dịch (T)

Các giao dịch dài hơn làm tăng khả năng tạo ra nhiều tập k-mục tiềm năng $\rightarrow$ nhiều ứng viên hơn $\rightarrow$ thời gian kiểm tra hỗ trợ (support counting) lâu hơn.

4. Ngưỡng hỗ trợ tối thiểu (minsup)

minsup cao $\rightarrow$ ít tập thường xuyên $\rightarrow$ ít vòng lặp $\rightarrow$ chạy nhanh

minsup thấp $\rightarrow$ sinh ra rất nhiều ứng viên $\rightarrow$ thuật toán có thể chậm mạnh hoặc tốn nhiều bộ nhớ

5. Số lượng ứng viên sinh ra trong mỗi vòng lặp

Đây là yếu tố quyết định tốc độ: càng nhiều ứng viên, thuật toán càng chậm.

6. Chiến lược lưu trữ và truy xuất dữ liệu

Apriori dùng giao dịch gốc. AprioriTid và Hybrid lưu bảng ID-list, giúp giảm kích thước dữ liệu về sau nhưng tốn bộ nhớ lúc đầu.

5.2 Thiết kế thực nghiệm

5.2.1 Môi trường thực nghiệm

(Điền thông tin hệ thống tại đây) Ví dụ:

CPU: Intel i7 - 12700H

RAM: 16 GB

Hệ điều hành: Windows 11

Ngôn ngữ cài đặt thuật toán: Python

Phiên bản thư viện hỗ trợ (nếu có): efficient_apriori

5.2.2 Tập dữ liệu và cấu hình tham số

Trong thực nghiệm, sử dụng 6 tập dữ liệu tổng hợp với các thông số được mô tả trong Bảng **??**.

Bảng 5.1: Cấu hình các tập dữ liệu sử dụng trong thực nghiệm

Dataset	**|D|**	**|I|**	**T**	**Minsup thử nghiệm**
T5.I2.D100K	100000	2	5	0.25%, 0.33%, 0.5%, 0.75%, 1%, 1.5%, 2%
T10.I2.D100K	100000	2	10	0.25%, 0.33%, 0.5%, 0.75%, 1%, 1.5%, 2%
T10.I4.D100K	100000	4	10	0.25%, 0.33%, 0.5%, 0.75%, 1%, 1.5%, 2%
T20.I2.D100K	100000	2	20	0.25%, 0.33%, 0.5%, 0.75%, 1%, 1.5%, 2%
T20.I4.D100K	100000	4	20	0.25%, 0.33%, 0.5%, 0.75%, 1%, 1.5%, 2%
T20.I6.D100K	100000	6	20	0.25%, 0.33%, 0.5%, 0.75%, 1%, 1.5%, 2%

5.2.3 Tiêu chí đánh giá

- Thời gian thực thi

- Số lần quét CSDL
- Số lượng và kích thước tập ứng viên
- Mức sử dụng bộ nhớ

5.3 So sánh Apriori, AprioriTid và AprioriHybrid

5.3.1 Số lần quét CSDL

Apriori: mỗi vòng lặp cần 1 lần quét lại toàn bộ CSDL → rất tốn chi phí khi số vòng lặp lớn.

AprioriTid: sau vòng đầu tiên, không cần quét CSDL nữa. Thuật toán chỉ làm việc trên bảng ID-list, giúp giảm chi phí I/O.

AprioriHybrid: kết hợp hai phương pháp.

Giai đoạn đầu dùng Apriori (khi ID-list còn lớn).

Khi kích thước ID-list nhỏ hơn CSDL gốc thì chuyển sang AprioriTid.

→ Về lý thuyết: AprioriTid Hybrid Apriori (về số lần đọc CSDL).

5.3.2 Chi phí sinh và kiểm tra ứng viên

Apriori: tốn chi phí kiểm tra ứng viên trên toàn bộ giao dịch.

AprioriTid: sử dụng danh sách ID-list nên việc kiểm tra hỗ trợ nhanh hơn ở các vòng sau.

Hybrid: giảm chi phí ở những vòng sâu (vòng 3 trở đi).

Nhược điểm AprioriTid: ID-list ban đầu rất lớn → tốn bộ nhớ.

5.3.3 Tốc độ thực thi tổng thể

Tốc độ phụ thuộc vào:

- Số vòng lặp
- Số lượng ứng viên sinh ra
- Chi phí đọc/ghi dữ liệu
- Kết luận thường gặp:
- minsup cao → cả ba thuật toán đều nhanh

- minsup thấp → Apriori chậm nhất, AprioriTid hoặc Hybrid tốt hơn tùy dataset

5.3.4 Mức sử dụng bộ nhớ

Apriori: tiết kiệm bộ nhớ, chỉ giữ dữ liệu gốc + ứng viên hiện tại.

AprioriTid: chiếm bộ nhớ lớn vì phải lưu bảng ID-list cho từng ứng viên.

Hybrid: dùng nhiều bộ nhớ ở giai đoạn chuyển tiếp nhưng nhìn chung tiết kiệm hơn AprioriTid.

5.3.5 Ảnh hưởng của minsup (thấp / cao)

minsup cao → ít ứng viên → cả 3 thuật toán chạy nhanh → Apriori có thể ngang bằng hoặc nhanh hơn do không tốn ID-list.

minsup thấp → nhiều ứng viên → Apriori chậm nhất vì phải quét lại CSDL. → AprioriTid và Hybrid vượt trội hơn, đặc biệt khi số vòng lặp lớn.

5.3.6 Kịch bản phù hợp cho từng thuật toán

Apriori: dataset nhỏ hoặc minsup cao.

AprioriTid: dataset lớn, minsup thấp, số vòng lặp nhiều.

Hybrid: trường hợp tổng quát – thường tốt nhất nếu không biết trước đặc trưng dữ liệu.

5.4 Kết quả thực nghiệm

Trong phần này, chúng tôi trình bày các kết quả thu được từ quá trình đánh giá hiệu năng của ba thuật toán Apriori, AprioriTid và AprioriHybrid. Kết quả bao gồm các bảng tổng hợp số liệu và các biểu đồ trực quan giúp so sánh tốc độ thực thi, số lượng ứng viên và mức sử dụng bộ nhớ.

5.4.1 Bảng tổng hợp số lượng ứng viên và thời gian chạy theo thuật toán (bảng 1)

Thuật toán	Độ hỗ trợ tối thiểu (%)						
	2	1.5	1	0.75	0.5	0.33	0.25
	Thời gian chạy - Dataset T5.I2.D100K						
setm	0.25	0.27	0.33	0.88	3.49	4.92	6.26
apriori	0.09	0.09	0.16	0.70	2.81	4.02	4.62
apriori_tid	0.19	0.21	0.34	0.81	3.21	4.54	5.19
apriori_hybrid	0.11	0.12	0.19	0.68	2.86	3.99	4.61
Số lượng ứng viên	**0**	**4**	**75**	**256**	**627**	**785**	**933**
	Thời gian chạy - Dataset T10.I2.D100K						
setm	0.46	1.01	4.69	7.55	9.77	10.07	10.79
apriori	0.17	0.69	3.64	5.99	7.76	8.00	8.51
apriori_tid	0.35	0.91	4.29	6.98	8.98	9.23	9.85
apriori_hybrid	0.21	0.72	3.70	6.04	7.90	7.91	8.63
Số lượng ứng viên	**32**	**169**	**541**	**752**	**920**	**947**	**2705**
	Thời gian chạy - Dataset T10.I4.D100K						
setm	0.52	1.43	7.20	10.08	11.71	12.17	12.36
apriori	0.23	1.05	5.94	7.92	9.22	9.64	9.64
apriori_tid	0.40	1.31	6.71	9.23	10.74	11.16	11.24
apriori_hybrid	0.23	1.06	5.80	8.00	9.32	9.52	9.78
Số lượng ứng viên	**40**	**217**	**677**	**849**	**955**	**995**	**1269**

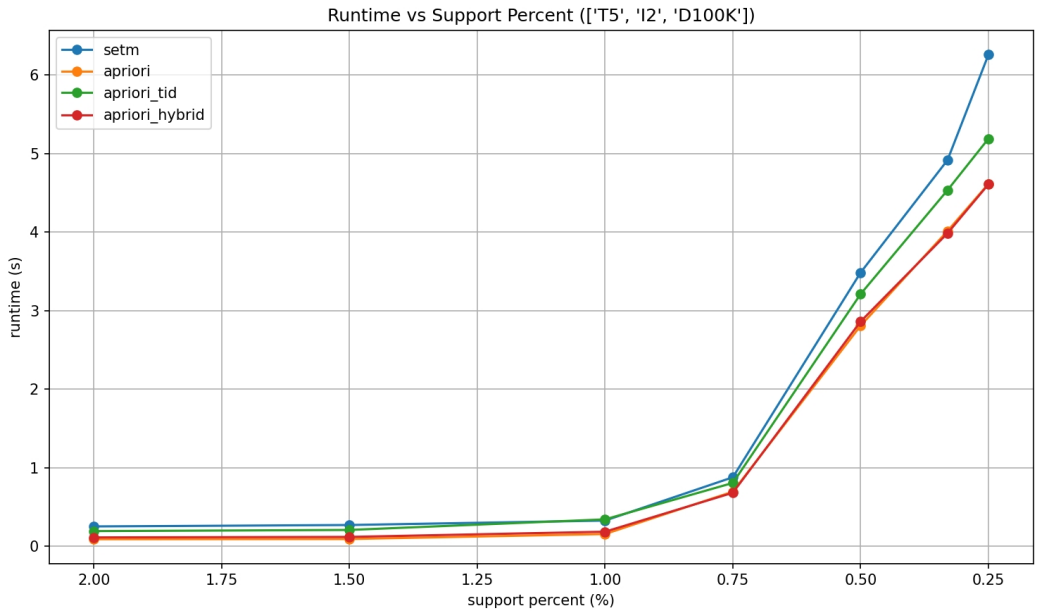
Hình 5.1: Bảng số lượng ứng viên theo mỗi vòng lặp của ba thuật toán

5.4.2 Bảng tổng hợp số lượng ứng viên và thời gian chạy theo thuật toán (bảng 2)

Thuật toán	Độ hỗ trợ tối thiểu (%)						
	2	1.5	1	0.75	0.5	0.33	0.25
	Thời gian chạy - Dataset T20.I2.D100K						
setm	5.91	15.15	19.36	20.77	21.01	21.62	22.39
apriori	4.89	12.50	16.19	17.30	17.35	17.78	18.55
apriori_tid	5.73	14.12	18.58	20.04	20.13	19.60	20.27
apriori_hybrid	4.89	12.54	16.11	17.38	17.38	17.78	18.60
Số lượng ứng viên	**431**	**777**	**932**	**982**	**1017**	**3678**	**8639**
	Thời gian chạy - Dataset T20.I4.D100K						
setm	8.73	17.24	21.72	22.34	22.68	22.73	55.00
apriori	7.35	14.55	18.11	18.68	18.73	18.64	32.99
apriori_tid	8.20	16.47	20.70	21.22	21.05	21.37	23.42
apriori_hybrid	7.40	14.58	18.24	18.68	18.67	18.97	32.70
Số lượng ứng viên	**534**	**819**	**972**	**996**	**1235**	**1993**	**56735**
	Thời gian chạy - Dataset T20.I6.D100K						
setm	11.53	20.38	22.83	23.85	24.11	24.20	26.49
apriori	9.81	17.32	20.50	20.46	20.33	20.21	21.16
apriori_tid	10.89	19.01	22.40	23.06	22.73	22.34	22.89
apriori_hybrid	10.39	17.24	19.98	20.64	19.88	20.41	21.18
Số lượng ứng viên	**613**	**877**	**977**	**997**	**1034**	**2240**	**6710**

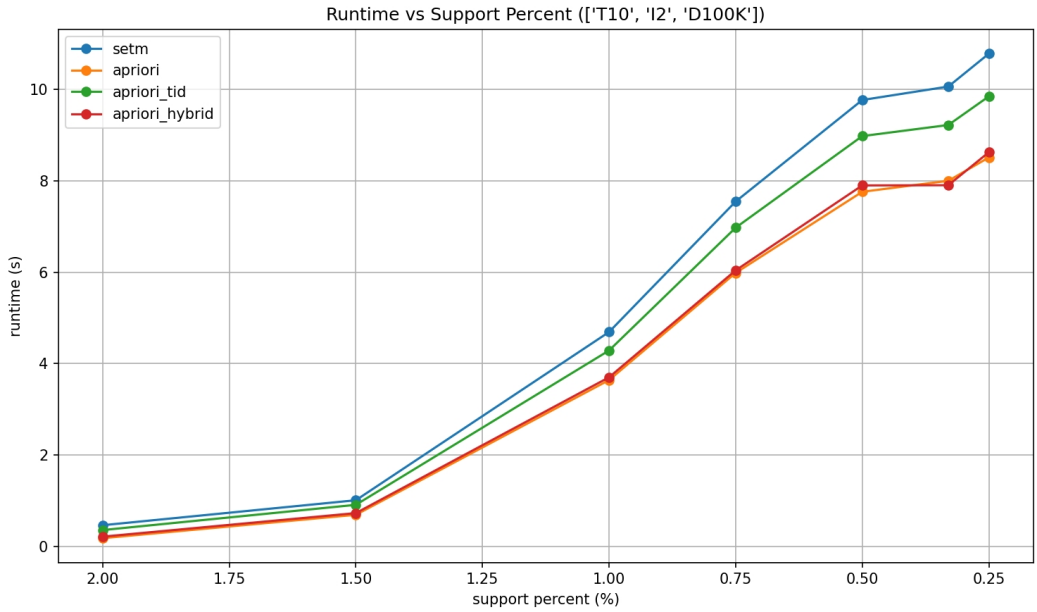
Hình 5.2: Bảng thời gian thực thi của Apriori, AprioriTid và AprioriHybrid

5.4.3 Biểu đồ số lượng ứng viên theo từng vòng

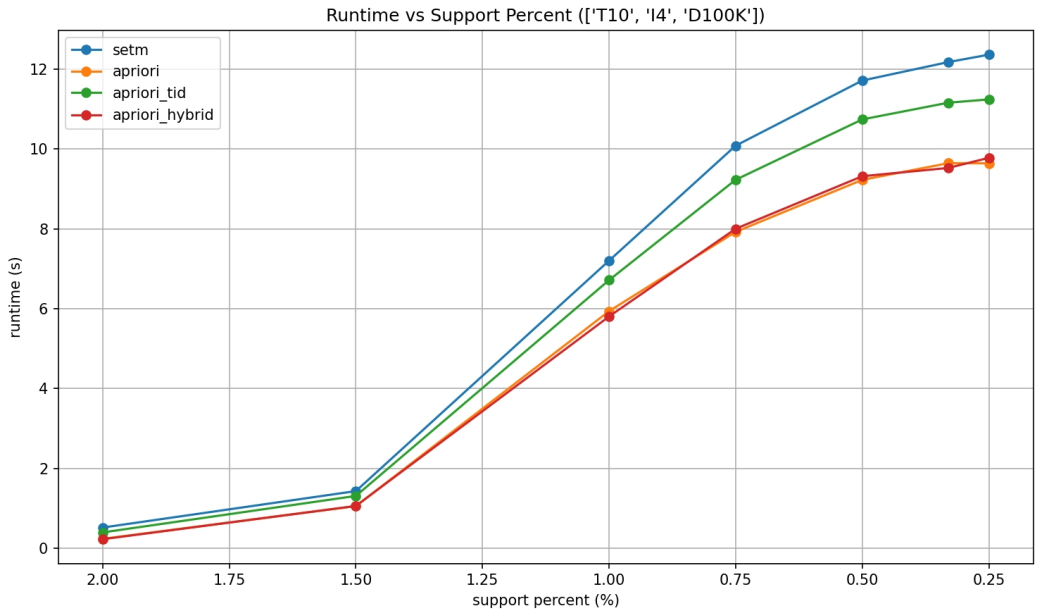


Hình 5.3: Biểu đồ số thời gian chạy cho Dataset T5.I2.D100K

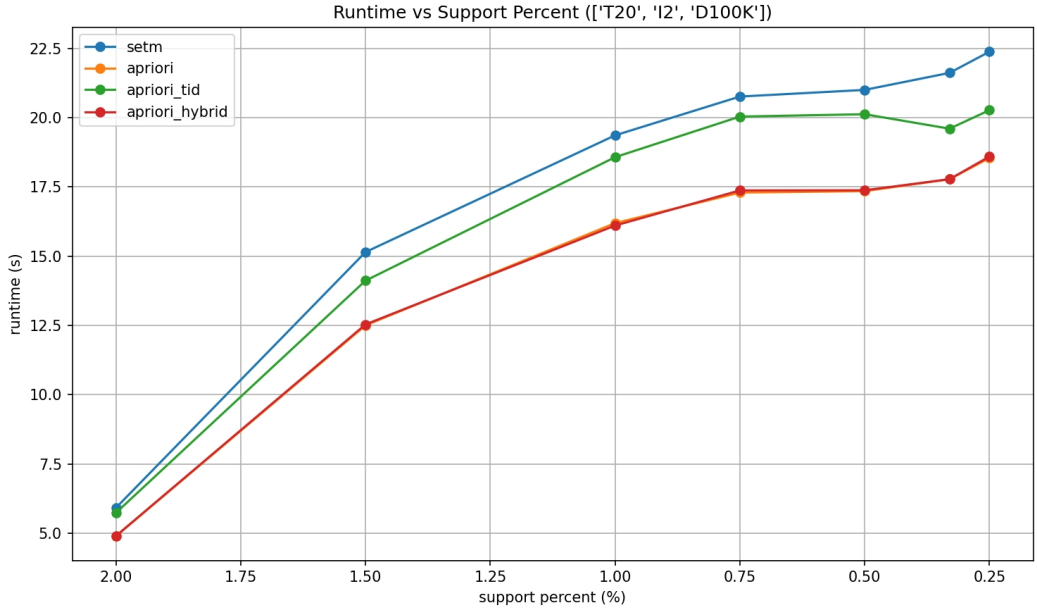
5.4.4 Biểu đồ thời gian thực thi của ba thuật toán



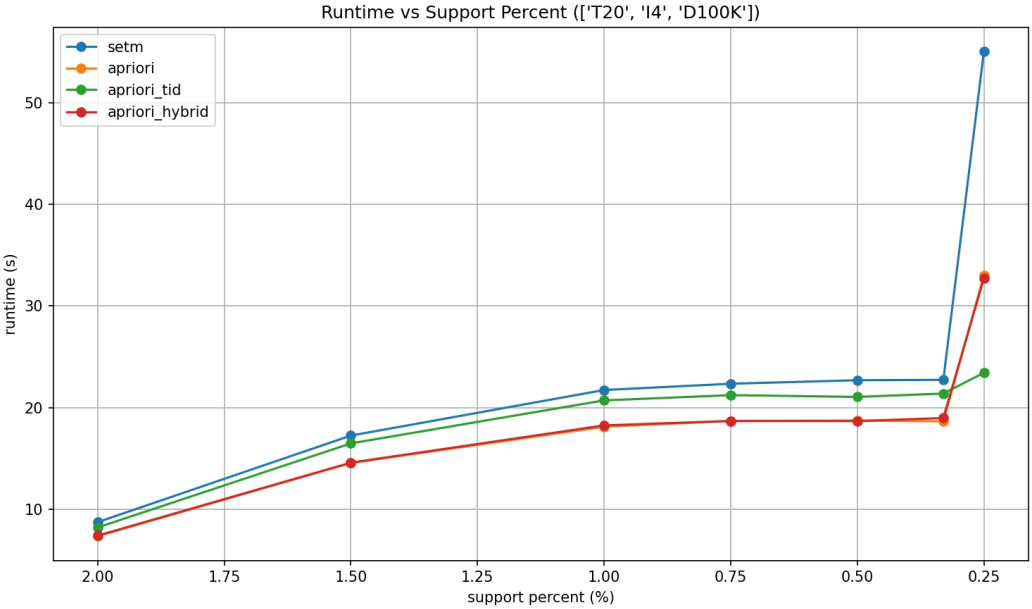
Hình 5.4: Biểu đồ số thời gian chạy cho Dataset T10.I2.D100K



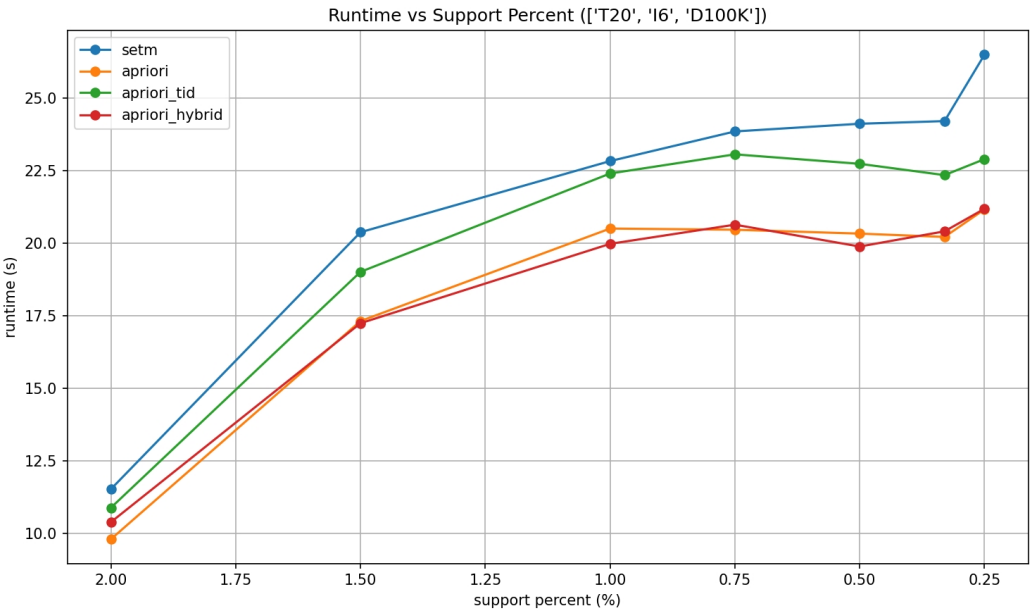
Hình 5.5: Biểu đồ số thời gian chạy cho Dataset T10.I4.D100K



Hình 5.6: Biểu đồ số thời gian chạy cho Dataset T20.I2.D100K



Hình 5.7: Biểu đồ số thời gian chạy cho Dataset T20.I4.D100K



Hình 5.8: Biểu đồ số thời gian chạy cho Dataset T20.I6.D100K

5.5 Nhận xét và thảo luận

5.5.1 Ưu và nhược điểm từng thuật toán

Apriori:

Ưu: đơn giản, dễ cài đặt, ít tốn RAM

Nhược: nhiều lần quét CSDL, không hiệu quả khi minsup thấp

AprioriTid:

Ưu: không quét lại CSDL sau vòng đầu

Nhược: danh sách ID-list lớn, dễ tràn bộ nhớ

Hybrid:

Ưu: cân bằng giữa hai thuật toán, thường nhanh nhất

Nhược: logic phức tạp hơn, cần điều kiện chuyển chế độ

5.5.2 Giải thích sự khác biệt giữa ba phương pháp

Lý do chủ yếu: cách lưu trữ và cách tính hỗ trợ khác nhau.

- Apriori dùng dữ liệu gốc
- AprioriTid dùng ID-list
- Hybrid chuyển đổi giữa hai giai đoạn

5.5.3 Các trường hợp sử dụng thích hợp

Dựa trên các đặc điểm về cấu trúc dữ liệu, chi phí xử lý và mức độ tiêu tốn tài nguyên, mỗi thuật toán trong ba thuật toán Apriori, AprioriTid và AprioriHybrid phù hợp với các tình huống ứng dụng khác nhau. Phần này trình bày các kịch bản sử dụng điển hình cho từng thuật toán.

Thuật toán Apriori

Thuật toán Apriori phù hợp trong các trường hợp:

- Tập dữ liệu có kích thước nhỏ hoặc trung bình, với số vòng lặp không quá lớn.
- Ngưỡng hỗ trợ tối thiểu (*minsup*) tương đối cao, giúp giảm số lượng tập ứng viên.
- Hệ thống ưu tiên tính đơn giản, dễ triển khai và dễ kiểm soát.
- Các bài toán phân tích giao dịch ở quy mô vừa, chẳng hạn như phân tích giỏ hàng trong các cửa hàng bán lẻ nhỏ hoặc tập dữ liệu hành vi người dùng có quy mô hạn chế (<ví

dụ: dữ liệu giỏ hàng từ một chuỗi cửa hàng nhỏ, dữ liệu clickstream của một website quy mô nhỏ>).

Thuật toán AprioriTid

AprioriTid thích hợp với các tình huống:

- Ngưỡng hỗ trợ thấp, khiến số lượng ứng viên tăng mạnh qua từng vòng lặp.
- Tập dữ liệu có số lượng giao dịch lớn nhưng độ dài mỗi giao dịch không quá lớn.
- Hệ thống có đủ bộ nhớ để lưu bảng ID-list trong quá trình thực thi.
- Các ứng dụng khai thác luật kết hợp yêu cầu xử lý khối lượng lớn dữ liệu, ví dụ: phân tích giỏ hàng trong các hệ thống bán lẻ quy mô lớn hoặc phân tích log hành vi người dùng trong các nền tảng thương mại điện tử (<ví dụ: dataset giao dịch của siêu thị lớn, dữ liệu clickstream của sàn thương mại điện tử>).

Thuật toán AprioriHybrid

AprioriHybrid kết hợp ưu điểm của hai thuật toán trên và phù hợp với phần lớn các tình huống thực tế:

- Các tập dữ liệu lớn và đa dạng, với đặc trưng không ổn định hoặc khó dự đoán trước.
- Các trường hợp yêu cầu cân bằng giữa tốc độ thực thi và mức sử dụng bộ nhớ.
- Khi hệ thống cần xử lý nhiều loại dữ liệu khác nhau, tránh phụ thuộc vào một thuật toán cố định.
- Các ứng dụng phân tích nâng cao trong thương mại điện tử, marketing và tài chính, chẳng hạn như phân tích thị trường, xây dựng hệ thống gợi ý sản phẩm hoặc khai thác mẫu chi tiêu của khách hàng (<ví dụ: dữ liệu giao dịch ngân hàng, dữ liệu hành vi người dùng của nền tảng streaming>).

5.5.4 Hạn chế và hướng cải thiện

Mặc dù các thuật toán Apriori, AprioriTid và AprioriHybrid được sử dụng rộng rãi trong khai thác luật kết hợp, chúng vẫn tồn tại một số hạn chế nhất định. Phần này trình bày các hạn chế chính và gợi ý một số hướng cải thiện tiềm năng trong tương lai.

Hạn chế

- **Chi phí sinh ứng viên lớn:** Tất cả các thuật toán dựa trên Apriori đều phụ thuộc vào chiến lược sinh–kiểm tra (*generate-and-test*), dẫn đến số lượng ứng viên tăng theo cấp số nhân khi *minsup* thấp hoặc khi tập mặt hàng lớn.
- **Nhiều lần quét cơ sở dữ liệu:** Apriori yêu cầu quét toàn bộ cơ sở dữ liệu nhiều lần, gây tiêu tốn thời gian I/O đáng kể đối với các tập dữ liệu lớn.
- **Tốn bộ nhớ (AprioriTid và Hybrid):** Việc duy trì các bảng ID-list trong AprioriTid và AprioriHybrid làm tăng đáng kể mức sử dụng bộ nhớ, đặc biệt ở các vòng lặp đầu.
- **Không phù hợp với dữ liệu rất lớn:** Các thuật toán Apriori truyền thống không hiệu quả trong môi trường dữ liệu lớn (*big data*) do chi phí xử lý cao và không tận dụng được khả năng tính toán song song hiện đại.
- **Không tối ưu cho dữ liệu có cấu trúc phức tạp:** Các thuật toán này chủ yếu phù hợp với dữ liệu dạng giao dịch; chúng không hoạt động tốt với dữ liệu có cấu trúc phức tạp như chuỗi thời gian, dữ liệu phân cấp hoặc dữ liệu đồ thị.

Hướng cải thiện

- **Sử dụng các thuật toán thay thế không sinh ứng viên:** Các thuật toán như FP-Growth hoặc ECLAT giúp loại bỏ chi phí sinh ứng viên và có khả năng mở rộng tốt hơn đối với tập dữ liệu lớn.
- **Tối ưu hóa truy xuất dữ liệu:** Áp dụng các kỹ thuật như nén giao dịch, đánh chỉ mục hoặc sử dụng cấu trúc dữ liệu hiệu quả hơn để giảm chi phí quét CSDL.
- **Khai thác tính song song và phân tán:** Việc triển khai trên các nền tảng như Hadoop MapReduce hoặc Spark cho phép mở rộng thuật toán tới quy mô dữ liệu rất lớn.
- **Điều chỉnh động ngưỡng *minsup*:** Một số nghiên cứu đề xuất thích ứng ngưỡng hỗ trợ theo cấu trúc dữ liệu giúp giảm số lượng ứng viên và tăng hiệu quả khai thác.
- **Ứng dụng học máy để giảm không gian tìm kiếm:** Các mô hình học máy hoặc học sâu có thể hỗ trợ dự đoán các tập mục tiềm năng, qua đó giảm số lượng ứng viên phải xét trong từng vòng lặp.

- **Phát triển phiên bản tối ưu theo đặc trưng dữ liệu:** Tuỳ chỉnh thuật toán cho từng loại dữ liệu cụ thể (<ví dụ: dữ liệu giao dịch thời gian thực, dữ liệu tuần tự, dữ liệu đồ thị>) giúp tăng hiệu quả rõ rệt.

Kết luận

Trong luận văn này, chúng em đã tiến hành nghiên cứu, phân tích và đánh giá hiệu năng của ba thuật toán khai thác tập mục thường xuyên dựa trên Apriori, bao gồm *Apriori*, *AprioriTid* và *AprioriHybrid*. Trên cơ sở khảo sát lý thuyết và thực nghiệm trên nhiều tập dữ liệu tổng hợp với các đặc trưng khác nhau, một số kết luận quan trọng đã được rút ra như sau.

- Thứ nhất, thuật toán Apriori mặc dù đơn giản và dễ triển khai nhưng gặp hạn chế lớn về số lần quét cơ sở dữ liệu và chi phí sinh ứng viên, đặc biệt khi giá trị *minsup* thấp hoặc số lượng mặt hàng lớn. Tuy nhiên, Apriori vẫn là lựa chọn phù hợp cho các bài toán quy mô vừa hoặc môi trường tài nguyên hạn chế.
- Thứ hai, AprioriTid thể hiện ưu điểm rõ rệt trong các vòng lặp sau, nhờ loại bỏ việc truy xuất trực tiếp vào cơ sở dữ liệu và thay thế bằng cấu trúc bảng ID-list. Điều này giúp cải thiện tốc độ khi số lượng ứng viên tăng mạnh, nhưng đi kèm mức sử dụng bộ nhớ cao hơn ở các vòng đầu.
- Thứ ba, thuật toán AprioriHybrid cho thấy sự cân bằng hiệu quả giữa hai phương pháp trên. Việc chuyển đổi từ Apriori sang AprioriTid tại thời điểm thích hợp giúp tối ưu hoá cả thời gian xử lý lẫn mức tiêu thụ bộ nhớ. Nhờ đó, AprioriHybrid đạt hiệu năng tốt nhất trong hầu hết các kịch bản thực nghiệm của đề tài.

Ngoài việc so sánh hiệu năng, luận văn cũng chỉ ra các hạn chế vốn có của họ thuật toán Apriori, đặc biệt trong bối cảnh dữ liệu lớn và yêu cầu tính toán theo thời gian thực. Từ đó, một số hướng cải thiện đã được đề xuất, bao gồm sử dụng các thuật toán không sinh ứng viên, khai thác tính song song, tối ưu hoá cấu trúc dữ liệu và tích hợp các phương pháp học máy để giảm không gian tìm kiếm. Trong tương lai, các nghiên cứu có thể mở rộng theo các hướng: đánh giá trên dữ liệu thực tế quy mô lớn, áp dụng trên môi trường phân tán như Spark hoặc Hadoop, hoặc kết hợp với các mô hình dự đoán nhằm nâng cao chất lượng khai thác

tập mục và luật kết hợp. Những cải tiến này hứa hẹn mang lại hiệu quả cao hơn, đáp ứng tốt hơn nhu cầu phân tích dữ liệu trong các hệ thống hiện đại. Tóm lại, kết quả nghiên cứu trong luận văn đã cung cấp cái nhìn toàn diện về đặc điểm, ưu điểm, hạn chế và hiệu năng của ba thuật toán Apriori, AprioriTid và AprioriHybrid. Các kết luận thu được không chỉ giúp lựa chọn thuật toán phù hợp cho từng loại bài toán mà còn là cơ sở cho các hướng nghiên cứu tiếp theo trong lĩnh vực khai phá dữ liệu.

Tài liệu tham khảo