Chương 1

1. Tìm hiểu về HOIM (High Occupancy Itemset Mining)
   1. **Khái niệm**

**HOIM (High Occupancy Itemset Mining)** là một phương pháp khai phá dữ liệu nhằm tìm ra các **tập mục (itemset)** có mức độ **"chiếm dụng cao" (occupancy)** trong cơ sở dữ liệu.  
Không giống các thuật toán truyền thống như **Apriori** hay **FP-Growth** chỉ dựa vào tần suất xuất hiện (support), HOIM đánh giá **mức độ bao phủ** của một tập mục trong toàn bộ dữ liệu.

Mục tiêu: Khai thác các tập mục không chỉ xuất hiện thường xuyên mà còn **có mặt trong nhiều phần của các giao dịch lớn hoặc quan trọng.**

* 1. **Định nghĩa độ chiếm dụng (Itemset Occupancy – IO)**

Độ chiếm dụng đo mức độ "phủ sóng" của một tập mục X trong cơ sở dữ liệu:

Trong đó:

* : là tập giao dịch
* : là một giao dịch
* : số lượng mục trong tập X
* : số lượng mục trong giao dịch T
* : tức là tập mục X xuất hiện trong T

**Ý nghĩa**: IO càng cao → tập mục X càng “chiếm dụng” nhiều trong CSDL → có giá trị khai thác cao hơn.

* 1. Điều kiện chọn tập mục

Một tập mục X được xem là tập mục chiếm dụng cao (High-Occupancy Itemset – HOI) nếu:

Trong đó **MinIO** là ngưỡng do người dùng đặt.

* 1. **Ưu điểm của HOIM**
* Xem xét **mức độ quan trọng thực tế** của tập mục trong dữ liệu thay vì chỉ đếm số lần xuất hiện.
* Hiệu quả hơn trong các trường hợp dữ liệu **không đồng đều**, nhiều giao dịch ngắn/dài khác nhau.
* Tập trung vào các mẫu **có ý nghĩa thực tiễn hơn**, nhất là trong hệ thống thương mại hoặc giám sát.
  1. Nhược điểm của HOIM
* **Không phân biệt các giao dịch** – mọi giao dịch đều được đối xử như nhau.
* **Không tối ưu tìm kiếm**: Không có giới hạn trên (upper bound) để loại bỏ sớm các tập mục không tiềm năng.
* **Chi phí tính toán cao** với tập dữ liệu lớn do phải duyệt tất cả các giao dịch.
  1. **Ví dụ minh họa (giản lược)**

Giả sử có 3 giao dịch:

|  |  |
| --- | --- |
| **Giao dịch** | **Mục trong giao dịch** |
| T1 | A, B, C |
| T2 | A, B |
| T3 | A, D |

Tính IO của tập mục **{A, B}**:

* Xuất hiện trong T1 và T2 → 2 giao dịch
* Tổng số mục trong T1 và T2 = 3 + 2 = 5
* Tổng số mục toàn CSDL = 3 + 2 + 2 = 7

Nếu MinIO = 0.5 thì {A, B} là tập mục chiếm dụng cao.

1. **Tìm hiểu về HOIMTO (High Occupancy Itemset Mining with Transaction Occupancy)**
   1. **Khái niệm**

**HOIMTO** là phiên bản cải tiến của thuật toán **HOIM**, được đề xuất nhằm khắc phục một số hạn chế của HOIM. Cụ thể, HOIMTO không chỉ dựa trên **độ chiếm dụng tập mục (IO)** mà còn kết hợp thêm **độ chiếm dụng giao dịch (TO)** để đánh giá chính xác hơn mức độ quan trọng của các tập mục trong cơ sở dữ liệu.

Mục tiêu: Ưu tiên các tập mục xuất hiện trong **giao dịch lớn/quan trọng**, từ đó lọc ra các mẫu có ý nghĩa thực tiễn cao hơn.

* 1. **Thành phần chính**

1. **Transaction Occupancy (TO)**

TO đo độ quan trọng của từng giao dịch dựa trên số lượng mục bên trong nó:

Trong đó:

* : số lượng mục trong giao dịch TT
* Mẫu số là tổng số mục trong toàn bộ tập dữ liệu

Giao dịch dài → TO cao → giao dịch quan trọng hơn → được ưu tiên khi đánh giá tập mục.

1. **Itemset Occupancy (IO) (có điều chỉnh)**

Tức là thay vì đếm cứng như trong HOIM, HOIMTO **cộng dồn TO** của các giao dịch chứa tập mục X.

1. **IOUB (Itemset Occupancy Upper Bound)**

Giới hạn trên giúp loại bỏ sớm các tập mục không tiềm năng:

Nếu ⇒ Không cần mở rộng tập X, vì không thể đạt yêu cầu.

* Cơ chế này giúp **cắt tỉa cây tìm kiếm** rất hiệu quả.
  1. **Điều kiện chọn tập mục**

Tương tự như HOIM, một tập mục XX được chọn nếu:

Tuy nhiên, IO ở đây là tổng các TO giao dịch chứa XX.

* 1. **Ưu điểm của HOIMTO**
* **Chính xác hơn**: Giao dịch quan trọng được xem xét ưu tiên.
* **Loại bỏ tập mục yếu sớm** nhờ IOUB.
* **Hiệu suất cải thiện rõ rệt** trên tập dữ liệu thực tế.
  1. **Nhược điểm**
* **Không xử lý trọng số từng mục (chỉ trọng số giao dịch).**
* **Chi phí tính toán vẫn còn cao** khi làm việc với dữ liệu lớn.
* IOUB vẫn có thể bị lỏng → cần tối ưu them.
  1. **Ví dụ minh họa đơn giản**

Giả sử:

* Dữ liệu có 3 giao dịch:
  + T1 = {A, B, C} → TO(T1) = 3/7
  + T2 = {A, B} → TO(T2) = 2/7
  + T3 = {A, D} → TO(T3) = 2/7

→ Tổng TO = 1

Xét tập mục {A, B}:

* Xuất hiện trong T1 và T2
* IO({A, B}) = TO(T1) + TO(T2) = 3/7 + 2/7 = **5/7 ≈ 0.714**

Nếu đặt MinIO = 0.6 → {A, B} được chọn là tập mục chiếm dụng cao.

1. Tìm hiểu về Cơ sở dữ liệu có trọng số (Weighted Database - WD)
   1. **Khái niệm**

**Cơ sở dữ liệu có trọng số (Weighted Database)** là dạng cơ sở dữ liệu trong đó **mỗi mục (item)** hoặc **mỗi giao dịch (transaction)** được gán một **trọng số (weight)** phản ánh **mức độ quan trọng**, **giá trị**, hoặc **độ ưu tiên** của chúng.

Trọng số có thể đại diện cho:

* **Giá trị tiền tệ** (ví dụ: giá sản phẩm)
* **Mức độ ưu tiên** trong phân tích
* **Tầm quan trọng** của mục đối với người dùng hoặc hệ thống
  1. **Các loại trọng số**

|  |  |
| --- | --- |
| **Loại** | **Mô tả** |
| **Trọng số mục (item weight)** | Gán trọng số cho từng mặt hàng hoặc mục dữ liệu (ví dụ: A = 5$, B = 10$) |
| **Trọng số giao dịch (transaction weight)** | Giao dịch tổng thể có trọng số riêng (ví dụ: giao dịch của khách VIP) |

* 1. **Ứng dụng thực tiễn**
* **Phân tích giỏ hàng bán lẻ**: Ưu tiên các mục có giá trị cao để tối ưu hóa lợi nhuận.
* **Khai phá dữ liệu giao dịch**: Phát hiện các tổ hợp mục có "giá trị" thực sự, thay vì chỉ xuất hiện nhiều.
* **Thương mại điện tử**: Gợi ý sản phẩm không chỉ dựa vào tần suất, mà còn dựa trên trọng số (lợi nhuận, xu hướng...)
  1. **Thuật toán liên quan**

Một số thuật toán đã được đề xuất để khai thác dữ liệu có trọng số:

|  |  |
| --- | --- |
| **Thuật toán** | **Mô tả** |
| **FWI (Frequent Weighted Itemset)** | Dựa trên ngưỡng **weighted support** để lọc tập mục |
| **HUIM (High Utility Itemset Mining)** | Khai phá tập mục có **lợi ích cao**, tính theo trọng số và số lượng |
| **NFWI (Weighted N-list)** | Tối ưu hóa FWI bằng cách dùng cấu trúc N-list có trọng số |

* 1. **Cách tính “weighted support”**

Ví dụ: Nếu mục A có trọng số 5, xuất hiện 3 lần trong các giao dịch:

→ Có thể đặt một ngưỡng để quyết định tập mục có đủ quan trọng hay không.

* 1. **Thách thức**
* **Trọng số không đồng đều** → cần phương pháp lọc hiệu quả để tránh nhiễu.
* **Không thể áp dụng trực tiếp các thuật toán phổ biến như Apriori** nếu không cải tiến chúng.
* **Cần thiết kế lại cách tính độ quan trọng của tập mục** (utility/occupancy) sao cho phù hợp với dữ liệu có trọng số.
  1. **Liên hệ với đề tài**

HOIMTO (High-Occupancy Itemset Mining with Transaction Occupancy) hiện tại **chưa xét trọng số**, tức là mặc định mọi mục trong giao dịch đều có "trọng lượng bằng nhau".  
=> Đây là **khoảng trống lý thuyết** để đề xuất thuật toán mới **HOWI-MTO (High-Occupancy Weighted Itemset Mining with Transaction Occupancy)** nhằm:

* Tích hợp trọng số vào TO và IO
* Đánh giá chính xác hơn vai trò của từng mục trong khai thác

Chương 2: **TRIỂN KHAI THUẬT TOÁN HOIM VÀ HOIMTO**

**1. Giới thiệu**

Khai thác tập mục chiếm dụng cao (High-Occupancy Itemset Mining) là một kỹ thuật khai thác dữ liệu tập trung vào mức độ bao phủ của các tập mục trong cơ sở dữ liệu giao dịch, thay vì chỉ dựa vào tần suất xuất hiện như trong khai thác tập mục phổ biến (Frequent Itemset Mining). Hai thuật toán được triển khai trong nghiên cứu này là:

* **HOIM** (High-Occupancy Itemset Mining): Tìm các tập mục có độ chiếm dụng (Itemset Occupancy - IO) cao dựa trên kích thước giao dịch.
* **HOIMTO** (High-Occupancy Itemset Mining with Transaction Occupancy): Cải tiến HOIM bằng cách sử dụng độ chiếm dụng giao dịch (Transaction Occupancy - TO) và giới hạn trên của độ chiếm dụng (Itemset Occupancy Upper Bound - IOUB) để tăng hiệu suất và độ chính xác.

Mục tiêu của các thuật toán là xác định các tập mục có IO lớn hơn hoặc bằng ngưỡng tối thiểu (MinIO), giúp khám phá các mẫu quan trọng trong dữ liệu, ví dụ: các sản phẩm thường được mua cùng nhau trong giỏ hàng.

**2. Cơ sở lý thuyết**

**2.1. Cơ sở dữ liệu giao dịch**

Cơ sở dữ liệu giao dịch là tập hợp các giao dịch, mỗi giao dịch là một tập hợp các mục (items). Ví dụ:

T1: a b c d

T2: a c

T3: b c d

Dữ liệu được lưu trong file transaction\_database.txt và được đọc vào chương trình dưới dạng tập hợp (set).

**2.2. Độ chiếm dụng tập mục (IO)**

* **HOIM**: , với là số mục trong giao dịch T .
* **HOIMTO**: , với : là độ chiếm dụng giao dịch.

**2.3. Độ chiếm dụng giao dịch (TO)**

Chỉ sử dụng trong HOIMTO, TO đo mức độ quan trọng của một giao dịch dựa trên số mục: Giao dịch có nhiều mục hơn sẽ có TO cao hơn, được ưu tiên trong tính toán IO.

**2.4. Giới hạn trên của độ chiếm dụng (IOUB)**

Chỉ sử dụng trong HOIMTO, IOUB dự đoán giá trị tối đa của IO. Nếu , tập mục ( X ) và các tập con mở rộng sẽ bị loại bỏ, giúp giảm không gian tìm kiếm.

**2.5. Ngưỡng tối thiểu (MinIO)**

Giá trị IO của tập mục phải đạt hoặc vượt MinIO để được coi là tập mục chiếm dụng cao (HOI). Trong triển khai, MinIO = 0.1 (HOIM) và 0.3 (HOIMTO).

**3. Triển khai thuật toán**

Thuật toán được triển khai bằng Python, sử dụng các thư viện chuẩn (collections, time, os) và psutil để đo hiệu suất. Dữ liệu đầu vào được đọc từ file transaction\_database.txt. Các bước triển khai được trình bày dưới đây, kèm mã nguồn chính.

**3.1. Thuật toán HOIM**

HOIM tìm các tập mục có IO cao dựa trên nghịch đảo kích thước giao dịch. Các bước triển khai:

**Bước 1: Đọc dữ liệu**

Hàm load\_database đọc file transaction\_database.txt, chuyển mỗi giao dịch thành tập hợp (set):

def load\_database(file\_path):

if not os.path.exists(file\_path):

print(f"Error: File {file\_path} not found!")

return []

database = []

with open(file\_path, 'r') as f:

for line in f:

items = line.strip().split()[1:] # Bỏ T1, T2,...

database.append(set(items))

print("Loaded database:", database)

return database

**Bước 2: Khởi tạo 1-itemsets**

Quét cơ sở dữ liệu để xây dựng item\_support, lưu danh sách giao dịch chứa từng mục:

item\_support = defaultdict(list)

for tid, t in enumerate(database):

for item in t:

item\_support[item].append(tid)

candidates = [[item] for item in item\_support]

**Bước 3: Tính IO và kiểm tra**

Với mỗi tập mục, tính IO và kiểm tra ngưỡng MinIO:

tids = [tid for tid, t in enumerate(database) if set(itemset).issubset(t)]

IO = sum(1 / len(database[tid]) for tid in tids)

if IO >= MinIO:

HOI.append((itemset, IO))

**Bước 4: Sinh k-itemsets**

Sinh các tập mục lớn hơn (k+1 itemsets) theo phương pháp kiểu Apriori:

if k == 1:

next\_candidates.extend([[itemset[0], new\_item] for new\_item in item\_support if new\_item > itemset[0]])

else:

for other in candidates:

if other[:k-1] == itemset[:k-1] and other[k-1] > itemset[k-1]:

next\_candidates.append(itemset + [other[k-1]])

**Bước 5: Lặp và trả kết quả**

Lặp lại bước 3-4 cho đến khi không còn tập mục mới, trả về danh sách HOI.

**3.2. Thuật toán HOIMTO**

HOIMTO cải tiến HOIM bằng cách sử dụng TO và IOUB. Các bước triển khai:

**Bước 1: Đọc dữ liệu**

Tương tự HOIM, sử dụng hàm load\_database (như trên).

**Bước 2: Tính TO**

Hàm calculate\_TO tính độ chiếm dụng giao dịch:

def calculate\_TO(database):

total\_items = sum(len(t) for t in database)

return [len(t) / total\_items for t in database]

**Bước 3: Khởi tạo 1-itemsets**

Tương tự HOIM, xây dựng item\_support và candidates.

**Bước 4: Tính IO và IOUB**

Hàm calculate\_IO\_IOUB tính IO và IOUB, cắt tỉa nếu IOUB nhỏ hơn MinIO:

def calculate\_IO\_IOUB(itemset, database, TO):

tids = [tid for tid, t in enumerate(database) if set(itemset).issubset(t)]

IO = sum(TO[tid] for tid in tids)

IOUB = sum(TO[tid] for tid, t in enumerate(database) if any(item in t for item in itemset))

return IO, IOUB, tids

**Bước 5: Kiểm tra và sinh k-itemsets**

Kiểm tra IOUB và IO, sinh k+1 itemsets tương tự HOIM:

IO, IOUB, tids = calculate\_IO\_IOUB(itemset, database, TO)

if IOUB >= MinIO:

if IO >= MinIO:

HOI.append((itemset, IO))

**Bước 6: Lặp và trả kết quả**

Trả về danh sách HOI với các tập mục và IO.

**3.3. Đo hiệu suất**

Cả hai thuật toán đo thời gian và bộ nhớ:

def get\_memory\_usage():

process = psutil.Process(os.getpid())

return process.memory\_info().rss / 1024 / 1024 # MB

start\_time = time.time()

start\_memory = get\_memory\_usage()

results = HOIM(database, MinIO) # hoặc HOIMTO

end\_time = time.time()

end\_memory = get\_memory\_usage()

print(f"Time: {end\_time - start\_time:.3f} seconds")

print(f"Memory: {end\_memory - start\_memory:.3f} MB")

**3.4. So sánh HOIM và HOIMTO**

File main.py chạy cả hai thuật toán và so sánh:

database = load\_database(file\_path)

MinIO = 0.3

print("HOIM results:")

hoim\_results = HOIM(database, MinIO)

for itemset, IO in hoim\_results:

print(f"Itemset: {itemset}, IO: {IO:.3f}")

print("\nHOIMTO results:")

hoimto\_results = HOIMTO(database, MinIO)

for itemset, IO in hoimto\_results:

print(f"Itemset: {itemset}, IO: {IO:.3f}")

**4. Kết quả**

Dữ liệu mẫu gồm 6 giao dịch với 4 mục (a, b, c, d):

T1: a b c d

T2: a c

T3: b c d

T4: a b c

T5: b d

T6: a c d

Tính các tập mục phổ biến nhất (tập 1 mục và vài tập 2 mục):

Ta có

* **HOIM** (MinIO = 0.1):

{a}: T1, T2, T4, T6 🡺 0.25+0.5+0.333+0.333≈1.416

{b}: T1, T3, T4, T5 🡺 0.25+0.333+0.333+0.5=1.416

{c}: T1, T2, T3, T4, T6 🡺 0.25+0.5+0.333+0.333+0.333​=1.75

{d}: T1, T3, T5, T6 🡺 0.25+0.333+0.5+0.333=1.416

{a,c}: T1, T2, T4, T6 🡺 0.25+0.5+0.333+0.333=1416

{b,c}: T1, T3, T4 🡺 0.25+0.333+0.333=0.916

{c,d}: T1, T3, T6 🡺 0.25+0.333+0.333=0.916

{a,d}: T1, T6 🡺 0.25+0.333=0.583

{b,d}: T1, T3, T5 🡺 0.25+0.333+0.5=1.083

{a,b}: T1, T4 🡺 0.25+0.333=0.583

Các tập mục thỏa mãn IO(X)≥0.3: ​{a}, {b}, {c}, {d},{a,c}, {b,c}, {c,d}, {a,d}, {b,d}, {a,b}​

* **HOIMTO** (MinIO = 0.3):

TO: ., ,

{a}: T1, T2, T4, T6 🡺 IO({a})=0.235+0.118+0.176+0.176=0.705

{b}: T1, T3, T4, T5 🡺 IO({b})=0.235+0.176+0.176+0.118=0.705

[c}: T1, T2, T3, T4, T6 🡺 IO({c})=0.235+0.118+0.176+0.176+0.176=0.881

{d}: T1, T3, T5, T6 🡺 IO({d})=0.235+0.176+0.118+0.176=0.705

{a,c}: T1, T2, T4, T6 🡺 IO({a,c})=0.235+0.118+0.176+0.176=0.705

{b,c}: T1, T3, T4 🡺 IO({b,c})=0.235+0.176+0.176=0.587

{c,d}: T1, T3, T6 🡺 IO({c,d})=0.235+0.176+0.176=0.587

{a,d}: T1, T6 🡺 IO({a,d})=0.235+0.176=0.411

{b,d}: T1, T3, T5 🡺 IO({b,d})=0.235+0.176+0.118=0.529

{a,b}: T1, T4 🡺 IO({a,b})=0.235+0.176=0.411

Các tập mục thỏa mãn IO(X)≥0.3: ​{a}, {b}, {c}, {d},{a,c}, {b,c}, {c,d}, {a,d}, {b,d}, {a,b}​

* **Hiệu suất**:
  + Thời gian: < 0.01 giây (dữ liệu nhỏ).
  + Bộ nhớ: < 1 MB.
  + HOIMTO nhanh hơn nhờ cắt tỉa bằng IOUB.

**5. So sánh HOIM và HOIMTO**

* **Thời gian**: HOIMTO nhanh hơn HOIM do cắt tỉa sớm bằng IOUB, giảm số lượng ứng viên.
* **Bộ nhớ**: HOIMTO tiêu tốn ít bộ nhớ hơn vì xử lý ít tập mục hơn.
* **Kết quả**: HOIMTO trả về tập hợp con của kết quả HOIM, vì IO trong HOIMTO dựa trên TO (tỷ lệ độ dài giao dịch), trong khi HOIM dựa trên nghịch đảo độ dài.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Cơ chế | Điểm mạnh | Điểm yếu |
| **HOIM** | * Tạo tập ứng viên 1-itemset, sau đó mở rộng dần sang k-itemset. * Kiểm tra IO của từng tập mục so với ngưỡng MinIO. * Sử dụng chiến lược cắt tỉa đơn giản dựa trên tập giao dịch (tids) chứa tập mục. | * Đơn giản, dễ triển khai. * Phù hợp với cơ sở dữ liệu nhỏ. | * Không sử dụng trọng số của các mục, dẫn đến việc không phản ánh đầy đủ mức độ quan trọng của các mục trong cơ sở dữ liệu có trọng số. * IO dựa trên nghịch đảo độ dài giao dịch có thể không tối ưu khi các giao dịch có độ dài chênh lệch lớn. |
| **HOIMTO** | * Tính TO cho từng giao dịch. * Sử dụng IOUB để cắt tỉa các tập mục không thỏa mãn ngưỡng MinIO trước khi tính IO. * Sinh tập ứng viên tương tự HOIM nhưng hiệu quả hơn nhờ cắt tỉa sớm. | * Cắt tỉa hiệu quả hơn HOIM nhờ IOUB, giảm số lượng ứng viên cần kiểm tra. * Phù hợp hơn với các cơ sở dữ liệu lớn. | * Vẫn không tích hợp trọng số của các mục, điều cần thiết trong cơ sở dữ liệu có trọng số. * IOUB có thể quá lỏng, dẫn đến việc giữ lại nhiều ứng viên không cần thiết. * Độ phức tạp tính toán IO và IOUB tăng khi cơ sở dữ liệu lớn. |

**6. Kết luận**

HOIM và HOIMTO là các công cụ mạnh mẽ để khai thác tập mục chiếm dụng cao. HOIMTO vượt trội về hiệu suất và độ chính xác, là nền tảng để phát triển thuật toán HOWI-MTO, kết hợp với cơ sở dữ liệu có trọng số trong các bước tiếp theo của nghiên cứu.

Chương 3: Hạn chế của HOIMTO và đề xuất ý tưởng cải tiến thuật toán HOWI-MTO

* 1. **Phân tích hạn chế của HOIMTO**

Dựa trên mã nguồn HOIMTO.py và mục tiêu nghiên cứu, dưới đây là các hạn chế chính của HOIMTO, được phân tích chi tiết hơn so với trao đổi trước:

* 1. **Không hỗ trợ trọng số của mục**

HOIMTO sử dụng Transaction Occupancy (TO) dựa trên độ dài giao dịch nhưng không tích hợp trọng số của các mục. Trong cơ sở dữ liệu có trọng số, các mục có mức độ quan trọng khác nhau (ví dụ: mục aaa có thể quan trọng hơn bbb), nhưng HOIMTO xử lý tất cả các mục như nhau.

Hạn chế này làm giảm khả năng phát hiện các tập mục quan trọng trong các ứng dụng thực tế, như phân tích giỏ hàng (nơi sản phẩm có giá trị khác nhau) hoặc khai thác dữ liệu cảm biến (nơi các sự kiện có mức độ ưu tiên).

* 1. **IOUB quá lỏng, gây lãng phí tài nguyên**:

IOUB được tính là tổng TO của các giao dịch chứa ít nhất một mục trong tập mục . Điều này dẫn đến việc giữ lại nhiều tập ứng viên không thỏa mãn ngưỡng MinIO, đặc biệt khi cơ sở dữ liệu có nhiều giao dịch hoặc mục.

Ví dụ: Trong cơ sở dữ liệu mẫu, IOUB của tập mục ab bao gồm tất cả giao dịch chứa a hoặc b, ngay cả khi ab chỉ xuất hiện trong một vài giao dịch, làm tăng số lượng ứng viên cần kiểm tra.

* 1. **Hiệu suất kém với cơ sở dữ liệu lớn**:

Việc tính IO và IOUB yêu cầu duyệt qua tất cả giao dịch cho mỗi tập mục, dẫn đến độ phức tạp cao (, với là số giao dịch và là số tập ứng viên).

Trong cơ sở dữ liệu mẫu (6 giao dịch), điều này không đáng kể, nhưng với hàng nghìn giao dịch, thời gian và bộ nhớ tăng đáng kể, như đã đề cập trong trao đổi trước khi bạn yêu cầu đo thời gian và bộ nhớ.

* 1. **Thiếu cơ chế tối ưu hóa cấu trúc dữ liệu**:

HOIMTO sử dụng danh sách tập giao dịch (tids) và duyệt tuần tự để kiểm tra tập mục con ***(set(itemset).issubset(t))***. Cách tiếp cận này không hiệu quả khi số lượng giao dịch hoặc kích thước tập mục tăng.

Không có cấu trúc dữ liệu chuyên biệt (như FP-Tree hoặc bitmap) để giảm thời gian truy vấn.

* 1. **Khả năng tùy chỉnh thấp**:

HOIMTO không cho phép điều chỉnh cách tính TO hoặc IO dựa trên đặc điểm của cơ sở dữ liệu. Ví dụ, trong một số ứng dụng, bạn có thể muốn ưu tiên các giao dịch có tổng trọng số cao hơn hoặc có số lượng mục đặc biệt.

* 1. **Không tận dụng thông tin trọng số trong cắt tỉa**:

Trọng số của các mục có thể được sử dụng để cắt tỉa sớm các tập mục không tiềm năng, nhưng HOIMTO không khai thác điều này, dẫn đến việc kiểm tra nhiều tập mục không cần thiết.

* 1. Đề xuất ý tưởng cải tiến cho thuật toán HOWI-MTO
  2. Ý tưởng 1: Tích hợp trọng số vào Occupancy

**Mô tả**: Thay vì chỉ sử dụng TO, tích hợp trọng số của các mục vào công thức tính Weighted Itemset Occupancy (WIO) để phản ánh mức độ quan trọng của tập mục.

**Công thức đề xuất**:

Trong đó:

* : Giữ nguyên như HOIMTO.
* ​: Trọng số của mục x (được cung cấp trong cơ sở dữ liệu hoặc tính toán dựa trên ứng dụng).
* : Trọng số trung bình của tập mục trong giao dịch.

**Lợi ích**:

* Phản ánh chính xác tầm quan trọng của tập mục trong cơ sở dữ liệu có trọng số.
* Phù hợp với các ứng dụng như phân tích thị trường (ưu tiên sản phẩm có giá trị cao) hoặc khai thác dữ liệu y tế (ưu tiên triệu chứng nghiêm trọng).
  1. Cải tiến IOUB thành WIOUB chặt chẽ hơn

**Mô tả**: Thiết kế Weighted IOUB (WIOUB) để cắt tỉa hiệu quả hơn, giảm số lượng ứng viên không tiềm năng.

**Công thức đề xuất**:

* Chỉ lấy trọng số lớn nhất của các mục trong tập mục tại mỗi giao dịch, thay vì tổng TO như IOUB.

**Lợi ích**:

* WIOUB chặt chẽ hơn IOUB, giảm số lượng tập ứng viên cần kiểm tra.
* Tận dụng thông tin trọng số để ưu tiên các tập mục có mục quan trọng.
  1. Sử dụng cấu trúc dữ liệu tối ưu

**Mô tả**: Thay vì duyệt tuần tự, sử dụng **FP-Tree** hoặc **bitmap** để lưu trữ cơ sở dữ liệu và tập giao dịch, giảm thời gian kiểm tra tập mục con.

**Lợi ích**:

* FP-Tree nén dữ liệu giao dịch, giảm bộ nhớ và thời gian truy vấn.
* Bitmap cho phép kiểm tra tập mục con nhanh chóng bằng phép toán bit.
  1. Cắt tỉa dựa trên trọng số và tần suất

**Mô tả**: Loại bỏ sớm các mục hoặc tập mục có trọng số hoặc tần suất thấp trước khi sinh ứng viên.

**Cách thực hiện**:

* Trước khi chạy thuật toán, tính WeightedSupport cho mỗi mục:
* Loại bỏ các mục có WS(x)<MinWIOWS.
* Trong quá trình sinh ứng viên, chỉ kết hợp các tập mục có tổng trọng số trung bình vượt ngưỡng nhất định.

**Lợi ích**:

* Giảm số lượng 1-itemset ngay từ đầu, tiết kiệm thời gian cho các bước sau.
* Tăng độ chính xác bằng cách tập trung vào các mục quan trọng.
  1. Tùy chỉnh TO dựa trên trọng số giao dịch

**Mô tả**: Thay vì chỉ dựa trên độ dài giao dịch, tính TO dựa trên tổng trọng số của các mục trong giao dịch.

**Công thức đề xuất**:

**Lợi ích**:

* Phản ánh mức độ quan trọng của giao dịch dựa trên trọng số của các mục.
* Phù hợp với các ứng dụng cần ưu tiên giao dịch có giá trị cao (ví dụ: giao dịch mua sắm lớn).
  1. Tối ưu hóa hiệu suất bằng xử lý song song

**Mô tả**: Sử dụng multiprocessing hoặc joblib để tính WIO/WIOUB cho các tập ứng viên song song.

**Lợi ích**:

* Giảm thời gian chạy trên cơ sở dữ liệu lớn.
* Tận dụng tài nguyên CPU đa lõi.