ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Nguyễn Ngọc Đức

BẢO VỆ TÍNH RIÊNG TƯ TRONG KHAI THÁC MẪU HỮU ÍCH

LUẬN VĂN THẠC SĨ KHOA HỌC MÁY TÍNH

TP. Hồ Chí Minh, Năm 2020

ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Nguyễn Ngọc Đức

BẢO VỆ TÍNH RIÊNG TƯ TRONG KHAI THÁC MẪU HỮU ÍCH

Chuyên ngành: **KHOA HỌC MÁY TÍNH**
Mã số: **8480101**

LUẬN VĂN THẠC SĨ KHOA HỌC MÁY TÍNH

Người hướng dẫn khoa học: **GS.TS LÊ HOÀI BẮC**

TP. Hồ Chí Minh, Năm 2020

LỜI CAM ĐOAN

Tôi xin cam đoan đây là công trình nghiên cứu của tôi dưới sự hướng dẫn của GS.TS Lê Hoài Bắc. Các số liệu và kết quả nghiên cứu được tôi tự tìm hiểu và phân tích một cách khách quan, trung thực. Kết quả thực nghiệm chưa được công bố ở bất kỳ công trình nghiên cứu nào khác. Các tài liệu tham khảo đều được trích dẫn rõ ràng, đúng quy định.

TP.Hồ Chí Minh, ngày 01 tháng 09 năm 2020

Học viên

Nguyễn Ngọc Đức

LỜI CẢM ƠN

Trước tiên, tôi xin chân thành cảm ơn tới các thầy (cô) đang giảng dạy chương trình sau Đại học tại khoa Công nghệ thông tin - trường Đại học Khoa học Tự nhiên đã tạo điều kiện tốt nhất, giúp đỡ tôi trong suốt quá trình học tập và nghiên cứu tại trường. Tôi xin bày tỏ lòng biết ơn sâu sắc đến GS.TS Lê Hoài Bắc người đã tận tình chỉ dẫn, giúp đỡ cũng như động viên tôi trong suốt thời gian thực hiện luận văn.

Tôi xin cảm ơn toàn thể học viên cao học khoa Công nghệ Thông tin khóa 28 - trường Đại học Khoa học Tự nhiên đã ủng hộ, động viên cũng như khuyến khích tôi trong quá trình nghiên cứu và thực hiện luận văn này.

Mặc dù đã cố gắng hoàn thành luận văn trong phạm vi và khả năng cho phép nhưng chắc chắn sẽ không tránh khỏi những thiếu sót, kính mong nhận được sự thông cảm cũng như lời góp ý của quý thầy, cô và các bạn.

Học viên Nguyễn Ngọc Đức

MỤC LỤC

DANH MỤC CÁC KÝ HIỆU, CHỮ VIẾT TẮT

δ	Ngưỡng lợi ích tối thiểu
CPU	Bộ xử lý trung tâm (Central processing unit)
CSP	Bài toán thỏa mãn ràng buộc (Constraints satisfaction problem)
GPU	Bộ xử lý đồ họa (Graphic processing unit)
HTWUI	Itemset TWU cao (High-TWU itemset)
HUI	Itemset lợi ích cao (High-utility itemset)
HUPM	Khai thác mẫu lợi ích cao (High-utility pattern mining)
ILP	Quy hoạch số nguyên tuyến tính (Integer linear programming)
IP	Quy hoạch số nguyên (Integer programming)
LP	Quy hoạch tuyến tính (Linear programming)
LP relaxation	Lời giải nới lỏng của bài toán quy hoạch số nguyên
LUI	Itemset lợi ích thấp (Low-utility itemset)
SHUI	Itemset lợi ích cao nhạy cảm (Sensitive high-utility itemset)

NSHUI	Itemset lợi ích cao không nhạy cảm (Non-sensitive high-utility itemset)
PPFIM	Bảo vệ tính riêng tư trong khai thác tập phổ biến Privacy-preserving frequent itemset mining
PPUM	Bảo vệ tính riêng tư trong khai thác mẫu hữu ích (Privacy-preserving utility mining)
SIP	Tỷ lệ thông tin nhạy cảm (Sensitive information percentage)

DANH MỤC CÁC BẢNG

DANH MỤC CÁC HÌNH VẼ, ĐỒ THỊ

TRANG THÔNG TIN LUẬN VĂN

Tên đề tài luận văn: Bảo vệ tính riêng tư trong khai thác mẫu hữu ích.

Ngành: Khoa học máy tính

Mã số ngành: 8480101.

Họ tên học viên cao học: Nguyễn Ngọc Đức.

Khóa đào tạo: 28.

Người hướng dẫn khoa học: GS.TS Lê Hoài Bắc.

Cơ sở đào tạo: Trường Đại học Khoa học Tự nhiên, ĐHQG.HCM.

1. Tóm tắt nội dung

Hiện nay, bảo vệ tính riêng tư trong khai thác mẫu hữu ích (PPUM) là một định hướng nghiên cứu cấp thiết. Các phương pháp PPUM bảo vệ thông tin của chủ sở hữu bằng cách chỉnh sửa dữ liệu trên dữ liệu gốc. Tuy nhiên, việc thay đổi dữ liệu sẽ làm cho thông tin thu được không chuẩn xác, thậm chí là khiến cho quá trình khai thác dữ liệu trở nên bất khả thi. Để giảm thiểu ảnh hưởng phụ tạo ra trong quá trình chỉnh sửa dữ liệu, Li và cộng sự đã đề xuất thuật toán PPUM-ILP dựa trên hướng tiếp cận quy hoạch số nguyên. Thuật toán PPUM-ILP không sử dụng ý tưởng heuristic mà mô hình hóa quy trình ẩn dữ liệu thành bài toán thỏa mãn ràng buộc (CSP) với mục tiêu ẩn các itemset nhạy cảm và giảm thiểu ảnh hưởng phụ. Kết quả cho thấy thuật toán này có tỷ lệ ảnh hưởng phụ thấp hơn so với các thuật toán khác này. Đóng góp chính của luận văn là giới thiệu phương pháp PPUM theo hướng tiếp cận quy hoạch số nguyên cũng như một số cải tiến, cụ thể:

- Kiến trúc bảng HIT được thiết kế giúp giảm thời gian tiền xử lý. Bên cạnh

đó, một cơ chế tiền xử lý trên GPU (Graphics processing unit) cũng được đề xuất.

- Lưu trữ biến số nguyên bằng hash table giúp tăng tốc quá trình mô hình hóa bài toán thỏa mãn ràng buộc (CSP).
- Chiến lược xấp xỉ lời giải CSP hiệu quả với bộ giải Gurobi [9].

Kết quả thực nghiệm cho thấy hiệu quả của thuật toán đề xuất đặc biệt là trên các tập dữ liệu thưa.

2. Những kết quả mới

- Việc mô hình hóa bài toán ẩn dữ liệu thành một bài toán ràng buộc giúp giải quyết bài toán một cách chính xác và tổng quát.
- Tiền xử lý trên GPU giảm thời gian thực thi trên các tập dữ liệu thưa.
- Chiến lược xấp sỉ lời giải trên Gurobi có thời gian thực thi ổn định.
- Các thuật toán chỉnh sửa item dựa trên ý tưởng heuristic cho kết quả gần giống nhau trên các tập dữ liệu thưa và lớn.

3. Ứng dụng và hướng phát triển

Ứng dụng

Khai thác mẫu hữu ích là kỹ thuật khai thác dữ liệu được ứng dụng rộng rãi trong nhiểu lĩnh vực (kinh tế, y học, sinh học, công nghệ thông tin,...). Do đó việc bảo vệ thông tin riêng tư của cá nhân, tổ chức trong khai thác mẫu hữu ích là cần thiết.

Hướng phát triển

- Phát triển một phương pháp lưu trữ dữ liệu tốt hơn cho GPU.

- Xây dựng các ràng buộc giúp giảm artificial cost tạo ra bởi quá trình ẩn dữ liệu.
- Áp dụng phương pháp đề xuất cho bài toán hẹp bảo vệ tính riêng tư trong khai thác tập phổ biến (PPFIM).

CHƯƠNG 1

MỞ ĐẦU

1.1. Giới thiệu bài toán

Quá trình khám phá tri thức từ cơ sở dữ liệu (còn gọi là khai thác dữ liệu) tập trung chủ yếu vào việc trích xuất các tri thức giá trị được ẩn giấu trong một khối lượng dữ liệu lớn và phức tạp. Tuy nhiên, thông tin khám phá trong khai thác dữ liệu có thể tiết lộ bí mật, vi phạm quyền riêng tư các cá nhân, tổ chức. Vì vậy bảo vệ tính riêng tư trong khai thác dữ liệu trở thành vấn đề quan trọng.

Khai thác mẫu hữu ích là một trong những kỹ thuật khai thác dữ liệu điển hình được ứng dụng rộng rãi. Việc bảo vệ tính riêng tư trong khai thác mẫu hữu ích (PPUM) trở thành vấn đề cấp thiết hiện nay [27]. Trong phần lớn các nghiên cứu được công bố, ý tưởng chính của PPUM là che giấu các itemset lợi ích cao nhạy cảm (SHUI), làm nhiễu cơ sở dữ liệu gốc bằng cách loại bỏ một số transaction hoặc giảm số lượng một số item nhất định. Chiến lược PPUM đầu tiên được Yeh và Hsu giới thiệu cùng với 2 thuật toán HHUIF và MSICF [26]. Mỗi thuật toán tìm transaction và item trong mỗi lần lặp theo tiêu chí riêng rồi giảm số lượng các item để hạ thấp lợi ích của SHUI. Hai hướng tiếp cận khác được Lin và các đồng sự đề xuất [**LIN2014**, 17]: chèn transaction mới vào cơ sở dữ liệu và xóa transaction hiện có. Lin và đồng sự cũng đề xuất thêm 2 thuật toán cùng với 3 biện pháp đánh giá hiệu suất PPUM [16]. Mặc dù ẩn giấu hoàn toàn các itemset nhạy cảm nhưng các phương pháp trên đều sử dụng ý tưởng heuristic. Vì vậy kết quả thu được có tính tổng quát thấp và tỉ lệ ảnh hưởng phụ đến cơ sở dữ liệu cao. Nhằm giải quyết vấn đề này, một

hướng tiếp cận mới đã được Li và đồng sự đề xuất [11]. Họ thiết kế một cơ chế tiền xử lý giúp giảm quy mô bài toán sau đó áp dụng quy hoạch số nguyên với mục đích hiệu chỉnh trạng thái của itemset theo cách hiệu quả, chính xác và tổng quát hơn.

1.2. Động lực và mục đích nghiên cứu

Trong thời điểm hiện nay, đi kèm với sự tiến bộ của công nghệ, sự ra đời của các kênh truyền thông mới và các thiết bị công nghệ tiên tiến là sự gia tăng chóng mặt của dữ liệu. Dữ liệu ngày càng đóng vai trò quan trọng trong cách các cơ quan, doanh nghiệp lập kế hoạch hoạt động, dự đoán và giải quyết vấn đề trong tương lai. Việc khai thác dữ liệu trong đó khai thác mẫu theo định hướng "độ hữu ích" trở thành chủ đề quan trọng và được nghiên cứu rộng rãi. Tuy nhiên quá trình khai thác mẫu hữu ích cũng sẽ trích xuất các thông tin nhạy cảm; vi phạm quyền riêng tư của cá nhân, tổ chức. Do đó, việc nghiên cứu và đề xuất một phương pháp bảo vệ tính riêng tư trong khai thác mẫu hữu ích là vô cùng cần thiết.

1.3. Đối tượng nghiên cứu

Đề tài nghiên cứu vấn đề chính: Bảo vệ tính riêng tư trong khai thác mẫu hữu ích.

1.4. Nội dung và phạm vi nghiên cứu

Nội dung của đề tài gồm những thành phần sau:

- Tìm hiểu về các phương pháp bảo vệ tính riêng tư trong khai thác mẫu hữu ích [**LIN2014**, 26, 28, 16, 17, 19, 11].
- Đề xuất một phương pháp bảo vệ tính riêng tư trong khai thác mẫu hữu ích.

- Áp dụng và đánh giá phương pháp đề xuất trên các tập dữ liệu khác nhau.

1.5. Phương pháp nghiên cứu

- Tìm hiểu về các phương pháp khai thác mẫu hữu ích và bảo vệ tính riêng tư trong khai thác mẫu hữu ích.
- Đề xuất một phương pháp bảo vệ tính riêng tư trong khai thác mẫu hữu ích.
- Xây dựng chương trình thử nghiệm.
- Đánh giá phương pháp đề xuất dựa trên kết quả thử nghiệm.

1.6. Cấu trúc của luận văn

Luận văn được trình bày trong 4 chương gồm:

- Chương 1: Mở đầu.
- Chương 2: Tổng quan.
- Chương 3: Thuật toán đề xuất IPPUM-ILP.
- Chương 4: Kết luận và hướng phát triển.

1.7. Phát biểu bài toán

Phần này sẽ mô tả ngắn gọn về bài toán PPUM và một số khái niệm liên quan. Một cơ sở dữ liệu D chứa N transaction: $D = \{T_1, T_2, ..., T_N\}$. Transaction $T_n \in D (1 \leq n \leq N)$ bao gồm một hoặc nhiều item khác nhau. $I = \{i_1, i_2, \ldots, i_M\}$ là M item khác nhau, $T_n \subset I$. Mỗi transaction có một *id* riêng biệt: *tid*. Bảng 1.1 thể hiện một cơ sở dữ liệu định lượng, giá trị external utility của các item được thể hiện trong bảng 1.2.

Bảng 1.1: *Cơ sở dữ liệu định lượng*

tid	A	B	C	D	E
0	3	7	0	4	5
1	3	6	0	0	0
2	0	0	4	6	0
3	0	0	6	4	3
4	7	1	0	6	0
5	0	0	0	3	9
6	7	3	0	0	0
7	5	0	2	3	0
8	8	0	6	0	0
9	6	0	8	0	6

Bảng 1.2: *Bảng giá trị external utility*

Item	External Utility
A	9
B	8
C	1
D	6
E	4

Định nghĩa 1 (*Internal Utility*)**.** *Internal utility của item i_m trong transaction T_n ký hiệu $In(i_m, T_n)$ là số lượng (hay giá trị phổ biến) của item i_m trong transaction T_n.*

Ví dụ: Trong bảng 1.1 internal utility của item A trong transaction T_0 là 3, cụ thể $\text{In}(A, T_0) = 3$.

Định nghĩa 2 (*External Utility*)**.** *External utility của item i_m ký hiệu $Ex(i_m)$ thể hiện tầm quan trọng hay độ lợi (lợi nhuận) của item i_m.*

Ví dụ: Trong bảng 1.2 external utility của item B là 8, cụ thể $\text{Ex}(B) = 8$.

Định nghĩa 3 (*Lợi ích của item i_m trong transaction T_n*)**.** *Lợi ích của item i_m trong transaction T_n ký hiệu $u(i_m, T_n)$ được tính bởi công thức:*

$$u(i_m, T_n) = In(i_m, T_n) \times Ex(i_m).$$

Ví dụ: Lợi ích của item B trong transaction T_0: $u(B, T_0) = \text{In}(B, T_0) \times \text{Ex}(D) = 7 \times 8 = 56$.

Định nghĩa 4 (*Lợi ích của itemset X trong transaction T_n*)**.** *Lợi ích của itemset X trong transaction T_n $(X \subseteq T_n)$ ký hiệu $u(X, T_n)$ được tính bằng tổng lợi ích tất cả*

các item thuộc X trong transaction T_n:

$$u(X, T_n) = \sum_{i_m \in X} u(i_m, T_n).$$

Lưu ý ta chỉ tính $u(X, T_n)$ khi $X \subseteq T_n$ nếu không giá trị của $u(X, T_n)$ sẽ bằng không. Ví dụ: lợi ích của itemset $\{AD\}$ trong transaction T_0 được tính như sau: $u(\{AD\}, T_0) = u(A, T_0) + u(D, T_0) = 3 \times 9 + 4 \times 6 = 51$. Vì $\{AD\} \nsubseteq T_1$ nên $u(\{AD\}, T_1) = 0$.

Định nghĩa 5 (*Lợi ích của itemset X trong cơ sở dữ liệu D*)**.** *Lợi ích của itemset X trong cơ sở dữ liệu D được tính bởi tổng lợi ích của X trong tất cả các transaction chứa X:* $u(X) = \sum_{T \in D, X \subseteq T_n} u(X, T_n)$.

Ví dụ: Lợi ích của itemset $\{AC\}$ trong cơ sở dữ liệu ở bảng 1.1 được tính như sau: $u(\{AC\}) = u(\{AC\}, T_7) + u(\{AC\}, T_8) + u(\{AC\}, T_9) = (5 \times 9 + 2 \times 1) + (8 \times 9 + 6 \times 1) + (6 \times 9 + 8 \times 1) = 47 + 78 + 62 = 187$.

Định nghĩa 6 (*Ngưỡng lợi ích tối thiểu*)**.** *Giá trị lợi ích là một thước đo trực quan được sử dụng để ước tính tầm quan trọng của các itemset. Bên cạnh đó, ngưỡng lợi ích tối thiểu là một tiêu chí để xác định liệu một itemset có giá trị hay mang lại lợi nhuận cho người dùng hay không. Thông thường, ngưỡng lợi ích tối thiểu ký hiệu δ là hằng số được người dùng đặt theo tỷ lệ với tổng giá trị lợi ích của cơ sở dữ liệu, tức là $\delta = c$ $\big(0 \leq c \leq$ tổng giá trị lợi ích$\big)$.*

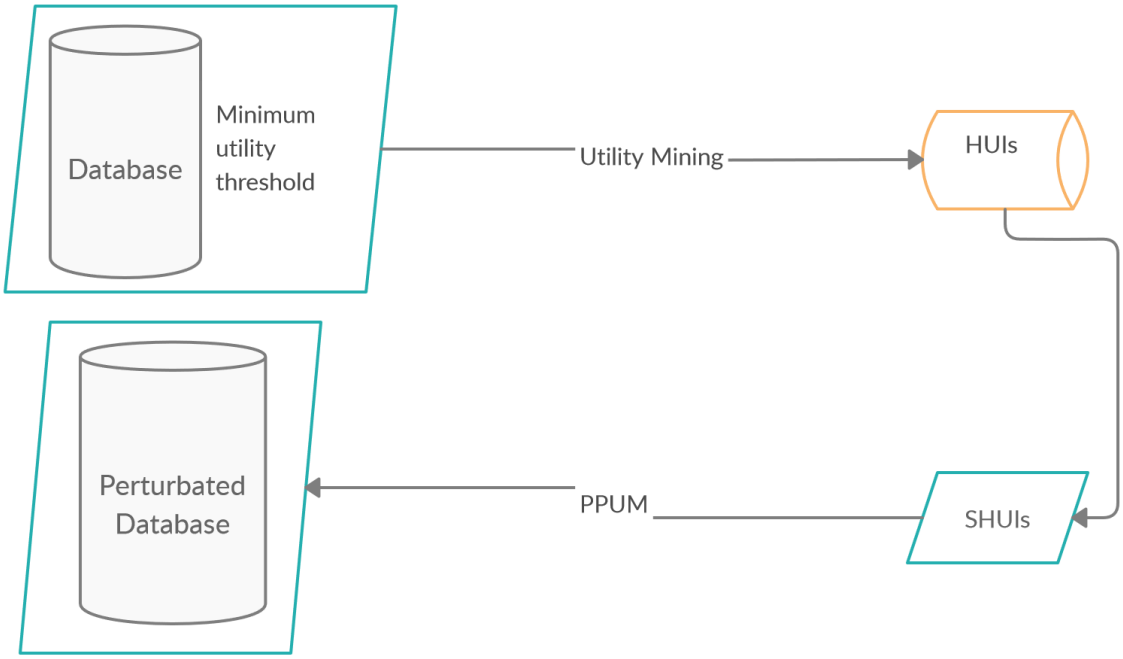
Định nghĩa 7 (*Itemset lợi ích cao - HUI*)**.** *Một itemset X được gọi là một itemset lợi ích cao khi lợi ích của itemset X lớn hơn hoặc bằng ngưỡng lợi ích tối thiểu δ.*

Gọi I_H là tập chứa toàn bộ các itemset có lợi ích cao trong cơ sở dữ liệu D ta có: $I_H = \{X \subseteq I : u(X) \geq \delta\}$.

Phát biểu bài toán. *Cho một cơ sở dữ liệu D, một tập các itemset lợi ích cao I_H và tập con của I_H là I_S chứa các itemset nhạy cảm lợi ích cao do người dùng chỉ định. Vấn đề chính của PPUM là tìm ra cách thích hợp để biến đổi cơ sở dữ liệu D thành*

cơ sở dữ liệu D'sao cho tất cả các itemset trong I_S đều được ẩn giấu đồng thời giảm thiểu các ảnh hưởng tiêu cực đến cơ sở dữ liệu D và các tri thức không nhạy cảm trong I_H.

Nhìn chung quy trình tổng quát của bài toán PPUM có thể được mô tả như trong hình 1.1.



Hình 1.1: *Kiến trúc tổng quát của PPUM.*

CHƯƠNG 2

TỔNG QUAN

2.1. Khai thác mẫu hữu ích

Khai thác mẫu hữu ích là kỹ thuật được sử dụng rộng rãi trong khai thác dữ liệu. Phương pháp này xem mỗi item có giá trị lợi ích khác nhau và số lượng (hay giá trị phổ biến) của các item trong các transaction không phải chỉ thể hiện dưới dạng nhị phân như trong cơ sở dữ liệu transaction truyền thống. Ngoài ra, để tìm được các mẫu lợi ích cao thì một ngưỡng lợi ích tối thiểu - δ phải được cân nhắc lựa chọn. Nếu δ quá thấp, quá nhiều mẫu được tìm ra, chi phí bộ nhớ và thời gian tính toán cao. Tuy nhiên nếu δ cao ta có thể bỏ qua nhiều mẫu hữu ích có giá trị lợi ích thấp.

Khám phá các mẫu lợi ích cao đóng vai trò quan trọng trong lĩnh vực y học như: phân tích dữ liệu y sinh và DNA. Các mẫu gene sinh học khác nhau có mức độ quan trọng khác nhau và dữ liệu một mẫu gene không bị giới hạn trong các giá trị nhị phân. Bằng cách tìm ra sự kết hợp của các mẫu gene lợi ích cao, những mẫu liên quan mật thiết đến một chứng bệnh có thể được phát hiện và tạo cơ sở cho việc nghiên cứu liệu pháp điều trị. Mẫu lợi ích cao cũng được ứng dụng vào lĩnh vực khai thác dữ liệu web. Nếu chúng ta xem xét mức độ quan trọng của các website và thời gian sử dụng của người dùng, các mẫu hữu ích cao khám phá được sẽ thể hiện mối liên hệ giữa người sử dụng internet và các website. Một số ứng dụng có thể kể đến khác như phân tích dữ liệu tài chính, dự đoán xu hướng chứng khoán,...

Trong những năm gần đây khai thác mẫu lợi ích cao (HUPM) [2, 24, 25] ngày càng trở thành chủ đề nghiên cứu phổ biến. Năm 2004, Yao và các đồng

sự [24] đã phát triển thuật toán khai thác itemset lợi ích cao (HUIM) bằng cách xem xét cả thông tin về số lượng (hay độ phổ biến) và lợi ích của các item. Vì tính bất đơn điệu không còn chính xác trong HUIM, mô hình Transaction Weighted Utilization (TWU) [20] được đề xuất nhằm tăng tốc quá trình khám phá HUI bằng cách tìm các itemset có TWU cao (HTWUI). Phương pháp này tìm kiếm HUI theo từng tầng.

Để khắc phục nhược điểm của hướng tiếp cận theo tầng, Lin và đồng sự [12] thiết kế kiến trúc cây HUP-tree. Bước đầu thuật toán HUP-tree tìm các 1-HTWUI (HTWUI chứa 1 item). Sau đó xây dựng cây HUP-tree rồi tìm toàn bộ các HUI. Năm 2012, một thuật toán mới mang tên HUI-miner được Liu và Qu [18] thiết kế dựa trên kiến trúc utility list với mục đích khai thác trực tiếp các HUI mà không cần phát sinh ứng viên. Thuật toán HUI-Miner được xem là thuật toán tốt nhất để khai thác tập lợi ích cao cho đến khi có sự xuất hiện của thuật toán FHM được để xuất bởi Phillipe và các đồng sự [5]. Một số vấn đề mở rộng và phương pháp cải tiến cũng được công bố vào các năm sau đó: [13, 14, 15, 8, 22].

2.2. Bảo vệ tính riêng tư trong khai thác mẫu hữu ích

Cùng với sự phát triển của các thuật toán khai thác mẫu hữu ích, sự quan ngại về việc rò rỉ thông tin đặc biệt là trong môi trường thương mại ngày càng trở nên nghiêm trọng; yêu cầu bảo vệ thông tin nhạy cảm ngày càng trở nên cấp thiết. Để giảm thiểu các mối đe dọa bảo mật trong khai thác mẫu hữu ích, bảo vệ tính riêng tư trong khai thác mẫu hữu ích (PPUM) đã phát sinh và trở thành chủ đề nghiên cứu quan trọng.

Công trình đầu tiên nghiên cứu về PPUM được công bố bởi Yeh và Hsu vào năm 2008 [26], cùng với hai thuật toán heuristic: HHUIF và MSICF được xây dựng để ẩn giấu các SHUIs. Ý tưởng chính của cả hai thuật toán là giảm lợi ích của các SHUIs bằng cách giảm số lượng của các item trong các transaction cụ thể, nhờ đó đạt được mục đích của bài toán. Điểm khác biệt duy nhất nằm

ở cách lựa chọn item cho việc chỉnh sửa và loại bỏ. Tuy nhiên, việc duyệt cơ sở dữ liệu quá mức cần thiết trong HHUIF, MSICF dẫn đến thời gian thực thi của hai thuật toán này rất lớn. Để giải quyết, kiến trúc cây FPUTT được xây dựng bởi Yun và Kim [28], hai kiến trúc dữ liệu phụ được áp dụng để tăng tốc tiến trình làm nhiễu. FPUTT đạt được kết quả chỉ với ba lần duyệt cơ sở dữ liệu. Mặc dù nhanh hơn các thuật toán trước, cây FPUTT rất cồng kềnh và vẫn chiếm một không gian lớn trong bộ nhớ, chi phí tính toán và sử dụng bộ nhớ cao.

Năm 2016, Lin và cộng sự đã phát triển hai thuật toán heuristic khác có tên MSU-MAU và MSU-MIU [16], trong đó khái niệm maximum utility và minimum utility được sử dụng để xóa hay giảm số lượng các item từ đó giảm lợi ích của các SHUI một cách hiệu quả. Vì PPUM khác với PPDM truyền thống, Lin và đồng sự cũng lập luận rằng những biện pháp đánh giá trong PPDM là không đủ để đánh giá các phương pháp đã phát triển cho PPUM, họ đề xuất ba tiêu chí tương tự mới. Ngoài ra, một số phương pháp meta-heuristic cũng được ứng dụng vào giải quyết bài toán PPUM. Hai thuật toán dựa trên giải thuật di truyền với tên gọi PPUMGA+insert, PPUMGAT được Lin và đồng sự thiết kế [**LIN2014**, 17]. PPUMGA+insert thêm transaction mới vào cơ sở dữ liệu trong khi PPUMGAT xóa transaction hiện có trong cơ sở dữ liệu. Tuy nhiên việc thêm và xóa transaction làm thay đổi số lượng transaction của cơ sở dữ liệu và tạo ra các itemset "giả" trong quá trình làm nhiễu dữ liệu.

Việc hạn chế ảnh hưởng phụ tạo ra do quá trình ẩn thông tin vẫn còn là một thách thức cần phải giải quyết, hiện tượng sai lệch và mất thông tin xảy ra thường xuyên. Nhằm giải quyết vấn đề này, Liu và đồng sự [19] đề xuất thuật toán IMSICF cải tiến từ thuật toán MSICF. IMSICF có chi phí tính toán và thời gian thực thi cao hơn các thuật toán khác. Mặc dù vậy, kết quả cho thấy khả năng giảm thiểu ảnh hưởng phụ của IMSICF cũng không mấy khả quan. Trong trường hợp xấu nhất gần 80% thông tin không nhạy cảm bị mất

Bảng 2.1: *Phân loại các thuật toán PPUM*

Thuật toán	Heuristics	CSP	Chỉnh sửa internal utility của item	Thêm hoặc xóa transaction
HHUIF	✓		✓	
MSICF	✓		✓	
FPUTT	✓		✓	
MSU-MAU	✓		✓	
MSU-MIU	✓		✓	
PPUMGAT	✓			✓
PPUMGA+insert	✓			✓
PPUM-ILP		✓	✓	
MSICF	✓		✓	

đi khi chỉnh sửa cơ sở dữ liệu. Một hướng tiếp cận mới tổng quát hơn được Li và đồng sự đề xuất vào năm 2019; thay vì sử dụng heuristic, họ giảm tỉ lệ ảnh hưởng phụ bằng cách mô hình hóa quá trình ẩn giấu các SHUI thành một bài toán thỏa mãn ràng buộc (CSP) sau đó giải quyết bằng kỹ thuật quy hoạch số nguyên [11].

Hai phần tiếp theo của chương này sẽ giới thiệu sơ lược về một số thuật toán khai thác mẫu hữu ích và bảo vệ tính riêng tư trong khai thác mẫu hữu ích.

Bảng 2.2: *So sánh các thuật toán PPUM.*

Thuật toán	Ý tưởng (lựa chọn transaction và item để chỉnh sửa)	Điểm mạnh và điểm yếu	Năm công bố
HHUIF	Transaction: chứa item mục tiêu. Item: có utility cao nhất.	Đơn giản nhưng không hiệu quả	2008

MSICF	Transaction: có utility của item mục tiêu cao nhất. Item: tần suất xuất hiện nhiều nhất.	Hiệu suất tốt trên tập dày.	2008
FPUTT	Tương tự HHUIF	Sử dụng kiến trúc cây để giảm thời gian thực thi.	2015
MSU-MAU	Transaction: MSU cao nhất. Item: có utility cao nhất trong transaction mục tiêu.	Khả năng giảm thiểu ảnh hưởng phụ tốt hơn so với các thuật toán trước.	2016
MSU-MIU	Transaction: MSU cao nhất. Item: có utility thấp nhất trong transaction mục tiêu.	Khả năng giảm thiểu ảnh hưởng phụ tốt hơn so với các thuật toán trước.	2016
PPUMGAT	Xóa transaction dựa trên giải thuật di truyền.	Cần phải có kinh nghiệm để định nghĩa các tham số, ảnh hưởng phụ cao.	2014
PPUMGA+insert	Chèn transaction dựa trên giải thuật di truyền	Cần phải có kinh nghiệm để định nghĩa các tham số, tạo ra các itemset giả, ảnh hưởng phụ cao.	2017
PPUM-ILP	Chỉnh sửa dựa trên kết quả CSP.	Ảnh hưởng phụ thấp nhưng thời gian thực thi không ổn định.	2019

IMSICF	Tương tự như MSICF nhưng tính "conflict count" liên tục trong suốt quá trình ẩn dữ liệu.	Giảm mất mát utility của dữ liệu nhưng thời gian thực thi lớn, tỷ lệ ảnh hưởng phụ gần như tương tự với MSICF.	2020

2.3. Thuật toán khai thác mẫu hữu ích HUI-Miner

Để có dữ liệu thực nghiệm, việc cần làm đầu tiên là cần phải tìm hiểu và cài đặt thuật toán khai thác mẫu hữu ích. Chương này sẽ giới thiệu bài toán khai thác mẫu hữu ích cũng như thuật toán khai thác được sử dụng trong quá trình thực nghiệm: HUI-Miner [18].

Phát biểu bài toán. *Cho một cơ sở dữ liệu định lượng D và một ngưỡng lợi ích tối thiểu δ. Khai thác mẫu hữu ích là kỹ thuật khai thác giúp khám phá các itemset có lợi ích không bé hơn δ.*

2.3.1. Một số định nghĩa

Định nghĩa 8 (*Utility của transaction T_n*)**.** *Utility của transaction T_n ký hiệu $tu(T_n)$ được tính bằng tổng utility của các item trong T_n:*

$$tu(T_n) = \sum_{i_m \in T_n} u(i_m, T_n)$$

Ví dụ utility các transaction trong cơ sở dữ liệu ở bảng 1.1 được tính trong bảng 2.3.

Định nghĩa 9. *Transaction-weighted utility (TWU) của itemset X trong cơ sở dữ*

tid	0	1	2	3	4	5	6	7	8	9
TU	127	75	40	42	107	54	87	65	78	86

Bảng 2.3: *Utility của các transaction trong cơ sở dữ liệu ở bảng 1.1*

liệu D ký hiệu $twu(X)$, là tổng utility của tất cả các transaction chứa X:

$$twu(X) = \sum_{T \in D \wedge X \subseteq T} tu(T)$$

Tính chất 1. *Nếu $twu(X)$ nhỏ hơn δ, tập cha của X không là HUI:*

$$\text{Nếu } X \subseteq X', \text{ thì } u(X') \leq twu(X') \leq twu(X) \leq \delta.$$

2.3.2. Kiến trúc Utility List

Các thuật toán dựa trên hướng tiếp cận FP-Growth thường tạo các itemset ứng viên từ prefix-trees và cần phải duyệt cơ sở dữ liệu để có thể tính chính xác utility. Để giải quyết, Liu và Qu thiết kế kiến trúc Utility-List [18] lưu trữ lại thông tin của cơ sở dữ liệu khai thác chỉ với 2 lần duyệt dữ liệu.

Khởi tạo Utility-List

Trong thuật toán HUI-Miner mỗi itemset được thể hiện bởi một Utility List. Bước đầu tiên HUI-Miner duyệt qua dữ liệu và tính TWU của tất cả item. Ở bước duyệt dữ liệu thứ hai, các item có TWU thấp hơn δ trong transaction bị loại bỏ, các item còn lại được sắp xếp theo thứ tự tăng dần TWU. Cũng trong bước này utility-list của từng item được khởi tạo. Với cơ sở dữ liệu trong bảng 1.1 các item được sắp xếp như sau: $E < C < B < D < A$.

Item	A	B	C	D	E
TWU	625	396	311	435	309

Bảng 2.4: *TWU của các item trong cơ sở dữ liệu ở bảng 1.1*

Định nghĩa 10. *Cho itemset X và transaction T_n: $X \subseteq T_n$ tập các item xếp sau X được ký hiệu: T_n / X.*

Định nghĩa 11. *Lợi ích còn lại của itemset X trong transaction T_n ký hiệu $ru(X, T_n)$ là tổng utility của các item thuộc T_n / X trong T_n: $ru(X, T_n) = \sum_{i \in T_n / X} u(i, T_n)$.*

Utility-list của itemset X bao gồm 3 thành phần (bảng 2.5, 2.6, 2.7):

- tid: tid của transaction T_n chứa X.
- iutil: utility của X trong transaction T_n: $u(X, T_n)$.
- rutil: rutil của X trong transaction T_n: $ru(X, T_n)$

Bảng 2.5: *Utility-list của itemset $\{A\}$*

tid	iutil	rutil
0	27	0
1	27	0
4	63	0
6	63	0
7	45	0
8	72	0
9	54	0

Bảng 2.6: *Utility-list của itemset $\{D\}$*

tid	iutil	rutil
0	24	27
2	36	0
3	24	0
4	36	63
5	18	0
7	18	45

Bảng 2.7: *Utility-list của itemset $\{B\}$*

tid	iutil	rutil
0	56	51
1	48	27
4	8	99
6	24	63

Utility-list của 2-Itemsets

Utility-list của 2-itemset có thể được tạo mà không cần phải duyệt dữ liệu. Utility-list của itemset $\{xy\}$ được xây dựng bằng cách lấy giao của utility-list $\{x\}$ và utility-list $\{y\}$.

Utility-list của k-Itemsets

Để xây dựng utility-list của k-itemset $c = \{i_1, i_2, ...i_{k-1}, i_k\} (k \geq 3)$, ta lấy giao utility-list của $a = \{i_1, i_2, ...i_{k-2}, i_{k-1}\}$ và $b = \{i_1, i_2, ...i_{k-2}, i_k\}$ tương tự như khi

Bảng 2.8: *Utility-list của itemset* $\{BA\}$

tid	iutil	rutil
0	83	0
1	75	0
4	71	0
6	87	0

Bảng 2.9: *Utility-list của itemset* $\{BD\}$

tid	iutil	rutil
0	80	27
4	44	63

ta xây dựng utility-list của các 2-itemsets. Tuy nhiên, nếu gọi t là tập các transaction 2 itemset a và b cùng xuất hiện; khi lấy giao của 2 itemset a và b, utility của itemset $\{i_1, i_2, ...i_{k-2}\}$ trong các transaction $T_n \in t$ được tính 2 lần. Vì vậy utility của c trong t được tính theo công thức sau:

$$u(c,t) = u(a,t) + u(b,t) - u(\{i_1, i_2, ...i_{k-2}\}, t).$$

Ví dụ: utility của itemset $\{BDA\}$ trong transaction 0: 83 + 80 -56 =107; utility của itemset $\{BDA\}$ trong transaction 4: 44 + 71 - 8 =107.

Bảng 2.10: *Utility-list của itemset* $\{BDA\}$

tid	iutil	rutil
0	107	0
4	107	0

Giả sử itemset Px và itemset Py là kết hợp của itemset P với item x và item y (x có thứ tự nằm trước y). Theo thứ tự, $P.UL$, $Px.UL$, và $Py.UL$ là các utility list của itemset P, Px, Py. Thuật toán 1 mô tả các bước cụ thể xây dựng utility-list của itemset Pxy. Utility list của 2-itemset được xây dựng khi $P = \varnothing$. Utility-list của k-itemset ($k \geq 3$) được xây dựng khi $P \neq \varnothing$.

2.3.3. Khai thác itemset lợi ích cao

Sau khi khởi tạo các utility-list từ cơ sở dữ liệu. HUI-Miner có thể sử dụng các utility-list đó để khai thác hiệu quả tất cả các HUI. Phần này sẽ mô tả không gian tìm kiếm, và chiến lược khai thác của thuật toán.

Thuật toán 1 Construct k-itemset utility-list[18].

Input:

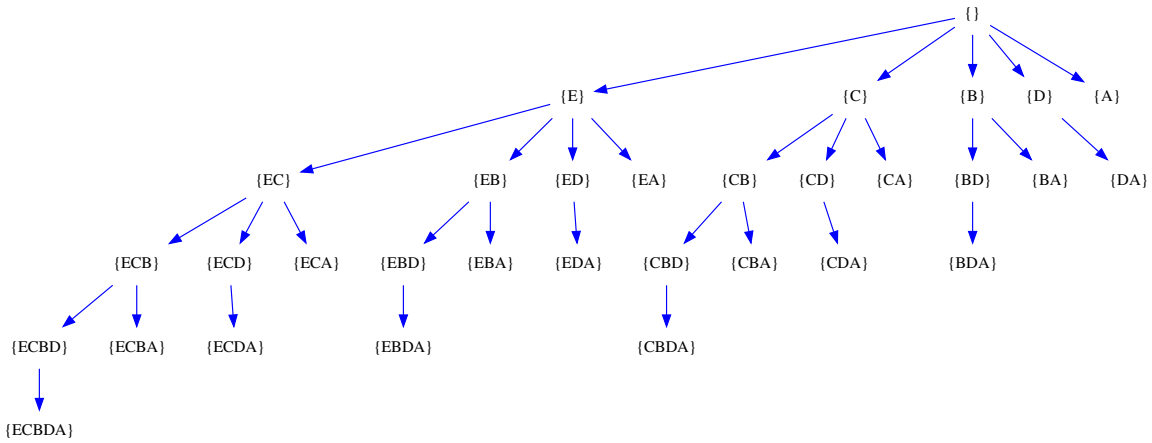
$P.UL$, the utility-list of itemset P
$Px.UL$, the utility-list of itemset Px
$Py.UL$, the utility-list of itemset Py

Output: Pxy.UL, the utility-list of itemset Pxy.

1: $Pxy.UL = NULL$
2: **for** each element Ex in $Px.UL$ **do**
3: **if** $Ey \in Py.UL$ and $Ex.tid == Ey.tid$ **then**
4: **if** $P.UL$ is not empty **then**
5: search such element $E \in P.UL$ that
6: $E.tid = Ex.tid$
7: $Exy \leq Ex.tid, Ex.iutils + Ex.iutil - E.iutil$
8: $Exy.rutil = Ey.rutil$
9: **else**
10: $Exy = [Ex.tid, Ex.iutil + Ey.iutil, Ey.rutil]$
11: **end if**
12: **end if**
13: **end for**

Không gian tìm kiếm

Không gian tìm kiếm của bài toán khai thác mẫu hữu ích có thể được biểu diễn bằng kiến trúc cây set-enumerate [21]. Một kiến trúc cây set-enumerate có thể được xây dựng như sau. Giả sử các item đã được sắp xếp theo thứ tự tăng dần TWU; đầu tiên, ta tạo nút gốc sau đó khởi tạo m nút con ứng với m 1-itemset. Bước thứ ba, với mỗi nút đại diện cho itemset $\{i_s,...,i_e\}$ $1 \leq s \leq e < n)$, $(n-e)$ ta thêm các nút con đại diện cho các itemset $\{i_s,...,i_e,i_{e+1}\}$, $\{i_s,...,i_e,i_{e+2}\}$,..., $\{i_s,...,i_e,i_n\}$. Lặp lại bước thứ ba cho đến khi ta đã tạo được tất cả các nút lá. Hình 2.1 thể hiện không gian tìm kiếm cho dữ liệu ở bảng 1.1.



Hình 2.1: *Không gian tìm kiếm.*

Chiến lược tỉa nhánh

Chiến lược vét cạn có thể tìm thấy tất cả HUI nhưng có chi phí về mặt thời gian rất lớn. Với một cơ sở dữ liệu có n item thì số lượng itemset cần phải xét lên đến 2^n. Để giảm bớt không gian tìm kiếm HUI-Miner tận dụng giá trị iutil và rutil lưu trữ trong utility-list của một itemset. Tổng iutil của itemset trong tất cả transaction là utility của itemset (theo định nghĩa 5). Nói cách khác itemset có tổng iutil vượt ngưỡng lợi ích tối thiểu - δ là một HUI. Nếu giá trị tổng iutil và rutil của một itemset nhỏ hơn δ ta không cần phải xét các nút con của itemset đó trong không gian tìm kiếm (chứng minh [18]). Chi tiết thuật toán HUI-Miner được mô tả trong Thuật toán 2.

2.4. Một số thuật toán bảo vệ tính riêng tư trong khai thác mẫu hữu ích

Phần này sẽ mô tả sơ lược về các thuật toán PPUM được sử dụng trong quá trình thực nghiệm.

Thuật toán 2 HUI-Miner Algorithm [18].

Input:

$P.UL$, the utility-list of itemset P, initially empty
ULs, the set of utility-lists of all P's 1-extensions
δ, the minimum utility threshold.

Output: all the itemset with P as prefix.

```
1: Pxy.UL = NULL
2: for each utility-list X in ULs do
3:     if SUM(X.iutils) ≥ δ then
4:         output the extension associated with X
5:     end if
6:     if SUM(X.iutils) + SUM(X.rutils) ≥ δ then
7:         exULs = NULL
8:         for each utility-list Y after X in ULs do
9:             exUls = exUls + Construct(P.UL, X, Y)
10:        end for
11:    end if
12: end for
```

2.4.1. HHUIF và MSICF

Ý tưởng chính của HHUIF [26] là ẩn giấu từng SHUI bằng cách lần lượt giảm internal utility của item có giá trị utility cao nhất. Cụ thể với sensitive itemset s, HHUIF tìm các transaction chứa s và lựa chọn item $i \in s$ có giá trị utility cao nhất làm mục tiêu. Transaction chứa item i được gọi là transaction mục tiêu ký hiệu $T|i$. Sau đó một chiến lược heuristic được áp dụng để xóa hoặc giảm utility của item i. Giá trị cách biệt ký hiệu $diff$ được tính cho mỗi SHUI. $diff$ thể hiện lượng utility ta cần phải giảm đi để ẩn SHUI hiện tại. Trong mỗi lần lặp, nếu $diff$ lớn hơn utility của item i, item i sẽ được xóa khỏi $T|i$; ngược lại giảm internal utility của item i đi $\left\lceil \frac{diff}{u(i)} \right\rceil$. Lặp lại cho đến khi $diff \leq 0$. Mã giả của thuật toán HHUIF được mô tả trong Thuật toán 3.

Định nghĩa 12 (*Conflict count*)**.** *Conflict count của một item nhạy cảm i_t trong SHUI s ký hiệu $Icount(i_t)$ là số lượng SHUI chứa i_t:* $|\{s_i \in I_S | i_t \in s_i\}|$.

Thuật toán 3 HHUIF algorithm.

Input: D, the original database; I_H, the set of HUIs discovered from D; I_S, the set of SHUIs to be hidden, minimum utility threshold δ.
Output: Sanitized database D'.

1: **for** each $s \in I_S$ **do**
2: diff = $u(s)$ - δ
3: **while** diff>0 **do**
4: $(i, T|i) = argmax_{i \in s, s \subseteq T}(u(i, T))$
5: Modify $o(i, T|i)$ with

$$o(i,T|i) = \begin{cases} 0, & \text{if } u(i,T|i) < diff \\ o(i,T|i) - \left\lceil \frac{diff}{u(i)} \right\rceil, & \text{if } u(i,T|i) > diff \end{cases}$$

6:

$$diff = \begin{cases} diff - u(i,T|i), & \text{if } u(i,T|i) < diff \\ 0, & \text{if } u(i,T|i) > diff \end{cases}$$

7: **end while**
8: **end for**
9: **return** the sanitized database D'

MSICF cũng thực hiện ẩn trên từng SHUI $s \in I_S$. Thuật toán lựa chọn item mục tiêu i trong s là item có conflict count lớn nhất, transaction mục tiêu $T|i$ là transaction mà internal utility của i là cao nhất. Chiến lược chỉnh sửa tương tự với HHUIF. Chi tiết của MSICF được trình bày trong Thuật toán 4.

Cả hai thuật toán HHUI và MSICF đều đơn giản và ẩn giấu được hoàn toàn tất cả SHUI. Tuy nhiên hiệu suất vẫn chưa đủ tốt và tạo ra ảnh hưởng phụ lớn đến cơ sở dữ liệu.

2.4.2. MSU-MAU và MSU-MIU

Năm 2016, Lin và đồng sự đề xuất hai thuật toán với tên gọi Maximum Sensitive Utility - Maximum Item Utility (MSU-MAU) và Maximum Sensitive Utility - Minimum Item Utility (MSU-MIU) với mục đích giảm thiểu ảnh hưởng phụ tạo ra bởi quá trình ẩn dữ liệu. Khái niệm MSU (maximum sensi-

Thuật toán 4 MSICF algorithm.

Input: D, the original database; I_H, the set of HUIs discovered from D; I_S, the set of SHUIs to be hidden, minimum utility threshold δ.
Output: Sanitized database D'.

1: Caculate conflict counts for all $i \in s$, $s \in I_S$. Arrange i by decreasing order of conflict counts.
2: **for** each i **do**
3: **for** each s contains i **do**
4: diff = $u(s)$ - δ
5: **while** diff>0 **do**
6: $(i, T|i) = argmax_{i \in s, s \subseteq T}(u(i, T))$
7: Modify $o(i, T|i)$ with

$$o(i,T|i) = \begin{cases} 0, & \text{if } u(i,T|i) < diff \\ o(i,T|i) - \left\lceil \frac{diff}{u(i)} \right\rceil, & \text{if } u(i,T|i) > diff \end{cases}$$

8:

$$diff = \begin{cases} diff - u(i,T|i), & \text{if } u(i,T|i) < diff \\ 0, & \text{if } u(i,T|i) > diff \end{cases}$$

9: **end while**
10: **end for**
11: **end for**
12: **return** the sanitized database D'

tive utility) được áp dụng để ẩn các SHUI một cách hiệu quả. Với mỗi SHUI s, hai thuật toán MSU duyệt qua cơ sở dữ liệu, tìm các transaction chứa s. Tiếp theo utility của itemset s trong mỗi transaction được tính toán, giá trị utility lớn nhất được lưu trữ và gọi là Maximum Sensitive Utility - MSU. Transaction có MSU lớn nhất được lựa chọn làm transaction mục tiêu để chỉnh sửa. Trong transaction mục tiêu, MSU-MAU chọn item có utility lớn nhất làm item mục tiêu để chỉnh sửa. Trái ngược với MSU-MAU, thuật toán MSU-MIU chọn item có utility nhỏ nhất làm item mục tiêu. Vì khi xóa bất kỳ một item $i \in s$ nào trong transaction T_n thì transaction T_n không còn chứa s ($s \nsubseteq T_n$) nên việc xóa item có utility thấp hay cao đều có ảnh hưởng như nhau. Lựa chọn item có

utility thấp để chỉnh sửa sẽ ít ảnh hưởng đến cơ sở dữ liệu hơn. Chiến lược giảm utility hoặc xóa item trong cả hai thuật toán đều tương tự như HHUIF và MSICF.

Thuật toán 5 MSU-MAU algorithm.

Input: D, the original database; I_H, the set of HUIs discovered from D; I_S, the set of SHUIs to be hidden, minimum utility threshold δ.
Output: Sanitized database D'.

1: **for** each $s \in I_S$ **do**
2: diff = $u(s)$ - δ
3: **while** diff>0 **do**
4: $T|i = argmax_{s \subseteq T}(u(s,T))$
5: $i = argmax_{i \subseteq T|i}(u(i,T|i))$
6: Modify $o(i,T|i)$ with

$$o(i,T|i) = \begin{cases} 0, & \text{if } u(i,T|i) < diff \\ o(i,T|i) - \left\lceil \frac{diff}{u(i)} \right\rceil, & \text{if } u(i,T|i) > diff \end{cases}$$

7:

$$diff = \begin{cases} diff - u(i,T|i), & \text{if } u(i,T|i) < diff \\ 0, & \text{if } u(i,T|i) > diff \end{cases}$$

8: **end while**
9: **end for**
10: **return** the sanitized database D'

Thuật toán 6 MSU-MIU algorithm.

Input: D, the original database; I_H, the set of HUIs discovered from D; I_S, the set of SHUIs to be hidden, minimum utility threshold δ.
Output: Sanitized database D'.

1: **for** each $s \in I_S$ **do**
2: diff = $u(s)$ - δ
3: **while** diff>0 **do**
4: $T|i = argmax_{s \subseteq T}(u(s, T))$
5: $i = argmin_{i \subseteq T|i}(u(i, T|i))$
6: Modify $o(i, T|i)$ with

$$o(i,T|i) = \begin{cases} 0, & \text{if } u(i,T|i) < diff \\ o(i,T|i) - \left\lceil \frac{diff}{u(i)} \right\rceil, & \text{if } u(i,T|i) > diff \end{cases}$$

7:

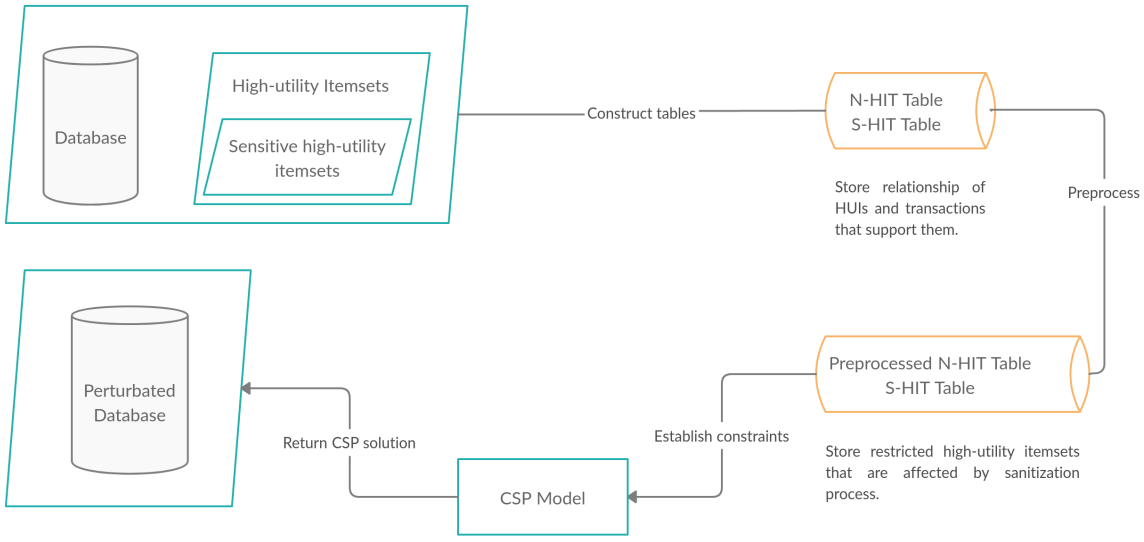
$$diff = \begin{cases} diff - u(i,T|i), & \text{if } u(i,T|i) < diff \\ 0, & \text{if } u(i,T|i) > diff \end{cases}$$

8: **end while**
9: **end for**
10: **return** the sanitized database D'

CHƯƠNG 3

THUẬT TOÁN ĐỀ XUẤT IPPUM-ILP

Ý tưởng chính của hướng tiếp cận PPUM dựa trên quy hoạch số nguyên là mô hình hóa quá trình ẩn giấu thông tin thành bài toán thỏa mãn ràng buộc (CSP) với 2 mục tiêu chính: ẩn giấu các SHUI và giảm thiểu ảnh hưởng phụ. Thuật toán đề xuất bao gồm 3 bước tương tự như [11]. Bước đầu tiên, ta sẽ xây dựng các kiến trúc dữ liệu lưu trữ mối liên hệ giữa các itemset và các transaction. Bước thứ hai, ta tiền xử lý dữ liệu để giảm quy mô bài toán, giữ lại những itemset chịu ảnh hưởng trong quá trình ẩn thông tin. Cuối cùng ta mô hình hóa CSP và giải bằng phương pháp quy hoạch số nguyên, lời giải của bài toán CSP được sử dụng để làm nhiễu cơ sở dữ liệu gốc. Các bước tổng quát của thuật toán IPPUM-ILP được mô tả chi tiết ở hình 3.1.



Hình 3.1: *Thuật toán IPPUM-ILP.*

3.1. Kiến trúc bảng HIT

Ngoài các thông tin nhạy cảm do người dùng chỉ định, việc lựa chọn các itemset nhạy cảm cần phải tuân theo một số chính sách quyền riêng tư nhất định và itemset được lựa chọn phải có lợi ích cao. Đầu vào của thuật toán sẽ bao gồm: cơ sở dữ liệu gốc D, tập các itemset nhạy cảm có lợi ích cao do người dùng chỉ định I_S, tập các itemset có lợi ích cao I_H. Ở bước đầu, ta xây dựng hai bảng với kiến trúc gần tương tự với 2 kiến trúc bảng trong [11] với tên gọi: high-utility itemset tidset (HIT). Kiến trúc bảng HIT được sử dụng để lưu giữ mối liên hệ giữa các itemset và các transaction; ngoài ra, kích thước và utility của itemset cũng được lưu giữ lại để phục vụ cho việc tiền xử lý và thiết lập ràng buộc cho mô hình CSP. Bảng sensitive high-utility itemset tidset (S-HIT) được xây dựng bằng cách duyệt qua toàn bộ dữ liệu gốc một lần với mỗi itemset nhạy cảm $s_i \in I_S$ ta lưu giữ bản thân s_i, các transaction chứa s_i cũng như kích thước và lợi ích của nó.

Bảng 3.1: *Kiến trúc bảng HIT*

High-utility itemset	Tidset	Itemset size	Itemset Utility
a_1	t_1	s_1	u_1
a_2	t_2	s_2	u_2
a_3	t_3	s_3	u_3

Bảng S-HIT được sử dụng để tăng tốc quá trình thiết lập các bất phương trình của CSP và tìm các HUI đặc biệt. Bằng cách sử dụng bảng S-HIT ta sẽ giảm bớt được số lần duyệt dữ liệu không cần thiết và tránh việc xử lý riêng biệt từng SHUI như 2 thuật toán heuristic HHUIF và MSICF [26]. Bảng non-sensitive high utility itemset tidset (N-HIT) cũng được xây dựng trong quá trình duyệt cơ sở dữ liệu. Điểm khác biệt là N-HIT lưu trữ các tập lợi ích cao không nhạy cảm (NSHUI) thay vì các SHUI.

Ý tưởng quan trọng ở đây là kiến trúc dữ liệu dùng để thể hiện các itemset và tidset trong bảng HIT. Đối với phiên bản tuần tự, mỗi itemset cũng như

tidset trong kiến trúc bảng HIT đều được lưu trữ dưới dạng hashset. Với hashset, độ phức tạp về mặt thời gian của việc kiểm tra một item có thuộc itemset i_n kích thước k là $O(1)$ thay vì $O(k)$, điều này giảm đáng kể thời gian khi tìm tập giao giữa hai itemset. Kích thước của itemset cũng được giữ lại để phục vụ cho phiên bản song song của IPPUM-ILP. Lợi ích của các itemset giúp tính ngưỡng một cách nhanh chóng cho bước thiết lập ràng buộc. Mã giả mô tả cụ thể các bước xây dựng bảng NHI và SHI trong phiên bản tuần tự được trình bày trong thuật toán 7.

Thuật toán 7 Table Construction

Input: D, the original database; I_H, the set of HUIs discovered from D; I_S, the set of SHUIs to be hidden
Output: N-HIT, S-HIT tables.

```
1:  for each transaction T_n ∈ D do
2:      for each itemset X ∈ I_H do
3:          if X ⊆ T then
4:              if X ∈ I_S then
5:                  Insert X, TID of T_n, X.size, U(X) into S-HIT.
6:              else
7:                  Insert X, TID of T_n, X.size, U(X) into N-HIT.
8:              end if
9:          end if
10:     end for
11: end for
```

Tất cả thông tin không ảnh hưởng đến quyền riêng tư cần được giữ lại cho quá trình khai thác dữ liệu. Tuy nhiên mâu thuẫn trong PPUM là không thể tránh khỏi. Quá trình ẩn SHUI gây thiệt hại đến các tri thức không nhạy cảm và tạo nên sự khác biệt lớn giữa cơ sở dữ liệu gốc và cơ sở dữ liệu nhiễu. Ba độ đo đánh giá ảnh hưởng phụ được Lin và đồng sự đề xuất [16]. Cụ thể, cho cơ sở dữ liệu gốc D, cơ sở dữ liệu nhiễu D'. I'_H là tập các HUI của D'. Cho I_N là tập các NHUIs trong D và I_S là tập các SHUIs trong D.

Định nghĩa 13 (*Hiding Failure (HF)*)**.** *: Hiding Failure thể hiện tỷ lệ các SHUI trong D vẫn được tìm thấy là các HUI trong D'. Gọi HF là tỷ lệ SHUI trong cơ sở*

dữ liệu trước và sau khi làm nhiễu, HF được tính theo công thức sau:

$$HF = \frac{|I_S \cap I'_H|}{|I_S|}$$

Định nghĩa 14 (*Missing Cost (MC):*)**.** *Missing cost thể hiện tỷ lệ các NSHUI trong D bị giảm lợi ích và trở thành các itemset có độ lợi thấp (LUI) trong D′. Gọi MC là tỷ lệ các NSHUI trước và sau khi làm nhiễu, MC sẽ được tính theo công thức sau:*

$$MC = \frac{|I_N - I_N \cap I'_H|}{|I_N|}$$

Định nghĩa 15 (*Artificial Cost (AC):*)**.** *Artificial cost thể hiện tỷ lệ các LUI trong D được gia tăng lợi ích và trở thành các HUI trong D′. Gọi AC là tỷ lệ các HUI gia tăng trước và sau khi làm nhiễu, AC sẽ được tính theo công thức sau:*

$$AC = \frac{|I'_H - I_H \cap I'_H|}{|I_N|}$$

3.2. Tiền xử lý dữ liệu

Trong quá trình ẩn dữ liệu, việc che giấu SHUI cũng ảnh hưởng đến các NSHUI có cùng item và transaction. Hậu quả là mất thông tin hoặc tạo ra các thông tin sai lệch, dư thừa. Bằng cách lọc ra những itemset thật sự bị ảnh hưởng khi làm nhiễu dữ liệu, chiến lược ẩn dữ liệu sẽ hiệu quả hơn, quy mô bài toán cũng giảm đáng kể. Thuật toán đề xuất sử dụng chiến lược tiền xử lý tương tự như trong [11]. Bước đầu tiên là lọc các NSHUIs bị ảnh hưởng bởi quá trình ẩn dữ liệu dựa trên cơ chế lọc dưới đây:

Định nghĩa 16 (*Cơ chế lọc [11]*)**.** *Cho tập các SHUIs I_S và tập các NHUIs I_N trong cơ sở dữ liệu D, gọi T_X là các transaction chứa tập X. itemset $n_i \in I_N$ không bị loại bỏ nếu n_i thỏa các điều kiện sau đối với ít nhất một itemset $s_i \in I_S$:*

1. *Itemset n_i và itemset s_i có chung ít nhất 1 item: $n_i \cap s_i \neq \varnothing$*

2. *Tidset T_{n_i} và tidset T_{s_i} có chung ít nhất một transaction: $T_{n_i} \cap T_{s_i} \neq \varnothing$.*

Gọi I'_N là tập các NSHUI còn lại của I_N sau quá trình lọc. Bước tiếp theo ta tìm các itemset trong I'_N chắc chắn bị ẩn đi trong quá trình làm nhiễu (Bound-to-lose Itemset) [11] để tránh xung đột trong quá trình xây dựng ràng buộc.

Định nghĩa 17 (*High-utility closed itemset (HUIC) [23]*)**.** *Cho itemset X và transaction set T_X chứa X, X được gọi là high-utility closed itemset khi và chỉ khi X thỏa các điều kiện sau:*

1. *Itemset X là itemset lợi ích cao;*
2. *Với bất kỳ itemset $Y \supset X$, $T_X \neq T_Y$.*

Định nghĩa 18 (*Bound-to-lose itemset* [11])**.** *Cho một NSHUI X và một transaction set T_X chứa X, X là Bound-to-lose Itemset khi và chỉ khi $\exists Y \in I_S$, X không closed với Y.*

Gọi tập các itemset còn lại là I''_N. Bước cuối cùng ta loại những itemset không cần thiết cho quá trình thiết lập ràng buộc. Cụ thể, trong I''_N nếu một itemset $n_{i_1} \in I''_N$ không closed với một itemset khác $n_{i_2} \in I''_N$ thì n_{i_2} có thể được loại bỏ an toàn khỏi I''_N mà không ảnh hưởng tới missing cost. Cuối cùng ta được tập I_R là output của bước tiền xử lý. Chi tiết quá trình tiền xử lý được mô tả ở thuật toán 8.

3.3. Tiền xử lý dữ liệu trên GPU

Phần này giới thiệu phiên bản song song của thuật toán IPPUM-ILP. Đầu tiên, cấu trúc dữ liệu biểu diễn các itemset cũng như tidset được định nghĩa lại. Sau đó, chiến lược tiền xử lý song song trên GPU được mô tả chi tiết.

3.3.1. Biểu diễn dữ liệu

Để ứng dụng GPU vào bước tiền xử lý, ta thay đổi cách lưu trữ các itemset và tidset: mỗi itemset được biểu diễn bằng một BitList [4] với kích thước tương

Thuật toán 8 Preprocessing

Input: N-HIT, S-HIT tables
Output: Modified N-HIT, S-HIT tables.

for each itemset n_i in N-HIT **do**
  $L \leftarrow \varnothing$;
  for each itemset s_i in S-HIT **do**
    if $n_i \cap s_i \neq \varnothing$ and $T_{n_i} \cap T_{s_i} \neq \varnothing$ **then**
      if $n_i \subseteq s_i$ and $T_{n_i} = T_{s_i}$ **then**
        Delete n_i from N-HIT; ▷ Bound-to-lose Itemset
      end if
    else
      Insert s_i into L;
    end if
  end for
  if $L == I_S$ **then**
    Delete n_i from N-HIT; ▷ $\nexists s_j \in I_S : s_j \cap n_i$
  end if
end for
for each itemset n_{i_2} in N-HIT **do**
  for each itemset n_{i_1} in N-HIT **do**
    if $n_{i_1} \neq n_{i_2}$ and $n_{i_1} \subseteq n_{i_2}$ and $T_{n_{i_1}} = T_{n_{i_2}}$ **then** ▷ n_{i_1} unclosed to n_{i_2}
      Delete n_{i_2} from N-HIT; break;
    end if
  end for
end for

ứng với số lượng item khác nhau M, mỗi tidset được biểu diễn bằng một BitList với kích thước tương ứng với số transaction của cơ sở dữ liệu N. Như vậy nếu **a** là vector bit biểu diễn itemset A, **b** là vector bit biểu diễn itemset B, số phần tử giao của 2 itemset A và B sẽ là tích vô hướng của hai vector **a** và **b**: $\mathbf{a} \cdot \mathbf{b}$. Cho tập $P = \{p_1, p_2, ..., p_x\}$ gồm x itemset khác nhau biểu diễn bằng một BitTable [4] gồm x BitList kích thước g, tập $Q = \{q_1, q_2, ..., q_y\}$ gồm y itemset khác nhau biểu diễn bằng một BitTable gồm y BitList kích thước g. Ta có ma trận bit **P** kích thước $(x \times g)$ và ma trận bit **Q** kích thước $(y \times g)$. Gọi **R** là kết quả phép nhân ma trận **P** và $\mathbf{Q}^T$: $\mathbf{R} = \mathbf{P}\mathbf{Q}^T$, **R** sẽ có kích thước $(p \times q)$ và giá trị $\mathbf{R}_{ij}$ là số phần tử giao của 2 itemset p_i và q_j $(p_i \in P, q_j \in Q)$.

Ví dụ:

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}, \mathbf{Q} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{R} = \mathbf{P}\mathbf{Q}^T = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$$

3.3.2. Xây dựng bảng N-HIT, S-HIT

Định lý 1. *Cho 2 tập hợp A và B, tập $I = A \cap B$. Nếu $|A| = |I|$ thì $A = I$ và $A \subseteq B$.*

Dựa trên định lý 1 một thuật toán xây dựng bảng N-HIT và S-HIT song song trên GPU được thiết kế. Gọi I_D là BitTable biểu diễn các itemset trong cơ sở dữ liệu D, bước đầu tiên ta khởi tạo bảng N-HIT với các itemset trong I_H, tính số phần tử giao của từng itemset giữa I_D và $I_H(I_{DH})$ và I_H và $I_S(I_{HS})$ bằng phương pháp nhân ma trận trên GPU. Sau đó ta duyệt (song song) qua I_{DH} và I_{HS}, lưu trữ tidset của các itemset và chuyển các sensitive itemset từ N-HIT sang S-HIT. Các bước thực hiện chi tiết được mô tả bằng mã giả trong Thuật toán 9.

3.3.3. Tiền xử lý

Dựa trên định lý 1, phương pháp tiền xử lý song song trên GPU (Thuật toán 10) cũng được xây dựng. . Điểm khác biệt so với thuật toán tuần tự là ta dùng L_n lưu trữ số lượng của các SHUI không giao với NSHUI n_i. Nếu L_n bằng với số itemset của I_S thì itemset n_i không giao với bất kỳ itemset nào trong I_S.

Thuật toán 9 GPU Table Construction

Input: I_D, BitTable representing the itemsets in database D; I_H, the set of HUIs discovered in D; I_S, the set of SHUIs to be hidden
Output: N-HIT, S-HIT tables.

```
1: Initialize N-HIT with itemsets in I_H
2: I_DH = matmul(I_D, I_H)          ▷ matmul: matrix multiplication using GPU
3: I_HS = matmul(I_H, I_S)
4: for threadIdx in threads do in parallel
5:     hIdx = threadIdx.x
6:     tid = threadIdx.y
7:     sIdx = threadIdx.z
8:     if I_DH[tid][hIdx] == I_H[hIdx].size then
9:         Insert tid to N-HIT[hIdx]    ▷ note that a tidset is represented by a BitList
10:    end if
11:    if I_HS[hIdx][sIdx] == I_H[hIdx].size == I_S[sIdx].size then
12:        Move N-HIT[hIdx] to S-HIT
13:    end if
14: end for
```

3.4. Mô hình hóa bài toán thỏa mãn ràng buộc (CSP)

Mục tiêu của thuật toán là làm nhiễu cơ sở dữ liệu, che giấu các SHUI. Để ẩn SHUI một cách hiệu quả, ta thay đổi internal utility của các item nhạy cảm. Mô hình CSP được thiết kế để tìm ra giá trị internal utility tối ưu, vừa ẩn được tất cả SHUI vừa giảm tối đa ảnh hưởng không mong muốn đến các itemset không nhạy cảm. Đầu tiên ta coi tất cả internal utility của các itemset nhạy cảm trong I_S là các biến số nguyên. Gọi tập các biến số nguyên là $V = \{v_{nm} | n \in [1, N] \cap \mathbb{N}, m \in [1, M] \cap \mathbb{N}\}$, v_{nm} đại diện cho internal utility của itemset m trong transaction n. Như vậy để loại bỏ một SHUI X ta có ràng buộc:

$$\sum_{T_n \in D, X \subseteq T_n} \sum_{i_m \in X} v_{nm} Ex(i_m) < \delta \tag{3.1}$$

Thuật toán 10 GPU Preprocessing

Input: N-HIT, S-HIT tables
Output: Modified N-HIT, S-HIT tables.

I_{NS} = matmul(I_N, I_S) ▷ I_N: itemsets in NHI, I_S: itemsets in SHI
T_{NS} = matmul(T_N, T_S) ▷ T_N: tidsets in NHI, T_S: tidsets in SHI
for threadIdx in threads **do** in **parallel**
  nIdx = threadIdx.x
  sIdx = threadIdx.y
  if $T_{NS} > 0$ and $I_{NS} > 0$ **then**
    if I_{NS}[nIdx][sIdx] == I_N [nIdx].size and T_N[nIdx].size == T_S[sIdx].size == T_{NS}[nIdx][sIdx] **then**
      Remove I_N[nIdx] from N-HIT
    end if
  else
    L[nIdx] += 1 ▷ use atomic operation for sychronizing results
  end if
  if L[nIdx] == number of sensitive itemsets **then**
    Remove I_N[nIdx] from N-HIT
  end if
end for
I_{NN} = matmul(I'_N, I'_N) ▷ I'_N: remain itemsets in NHI
T_{NN} = matmul(T'_N, T'_N) ▷ T'_N: remain tidsets in NHI
for threadIdx in threads **do** in **parallel**
  n_1 = threadIdx.x
  n_2 = threadIdx.y
  if $n_1 \neq n_1$ **then**
    if $I_{NN}[n_1][n_2] == I'_N[n_1]$.size and $T_{NN}[n_1][n_2] == T'_N[n_1]$.size == $T'_N[n_2]$.size **then**
      Remove $I'_N[n_2]$ from N-HIT
    end if
  end if
end for

Gọi U là tập chứa internal utilities của các item trong I_R là

$$U = \left\{ u_{n'm'} | n' \in [1, N] \cap \mathbb{N}, m' \in [1, M] \cap \mathbb{N} \right\}.$$

Với $u_{n'm'}$ thể hiện 2 trạng thái internal utility của mỗi item trong I_R, cụ thể

$$u_{n'm'} = \begin{cases} v_{n'm'}, \text{ nếu } v_{n'm'} \in V, \\ \text{In}(i_{m'}, T_{n'}), \text{ trường hợp còn lại} \end{cases}$$

Vậy để giữ lại itemset $Y \in I_R$ ta có ràng buộc:

$$\sum_{T_{n'} \in D, Y \subseteq T_{n'}} \sum_{i_{m'} \in Y} u_{n'm'} Ex(i_{m'}) \geq \delta \tag{3.2}$$

Giá trị tối thiểu biến số nguyên được đề xuất là 1 [11] vì nếu thay đổi internal utility của một item trong transaction T_n bằng giá trị 0 thì itemset chứa item đó sẽ mất toàn bộ utility trong transaction T_n. Bên cạnh đó, với những tập dữ liệu dày, một item có thể xuất hiện trong nhiều transaction. Nếu internal utility của item này bị đẩy lên quá cao thì khả năng hàng loạt các LUI sẽ trở thành HUI, làm tăng artifical cost và dẫn đến thông tin khai thác sai lệch nghiêm trọng. Mô hình CSP đề xuất dùng giá trị internal utility cao nhất của tập dữ liệu gốc làm ngưỡng giá trị tối đa cho các biến số nguyên (chỉ áp dụng trên các cơ sở dữ liệu dày). Hàm mục tiêu của CSP là hàm tối thiểu tổng giá trị các biến. Một cách tổng quát mô hình CSP được mô tả như sau:

$$min \sum_{v_{nm} \in V} v_{nm} \tag{3.3}$$

$$s.t. \sum_{i_m \in X} \sum_{T_n \in T_X} v_{nm} Ex(i_m) \leq \delta, \qquad \forall X \in I_S \tag{3.4}$$

$$\sum_{i_{m'} \in Y} \sum_{T_{n'} \in T_Y} u_{n'm'} Ex(i_{m'}) \geq \delta, \qquad \forall Y \in I_R \tag{3.5}$$

$$1 \leq v_{nm} \leq max(\text{internal utilities}), \qquad \forall v_{nm} \in V \tag{3.6}$$

Thuật toán 11 CSP Formulation

Input: S-HIT, N-HIT tables. **Output:** CSP model.

1: Using the S-HIT table create a hashmap V to store variable positions and utilities; establish inequalities arcoding to formula 3.4.
2: **for** each itemset n_i in N-HIT **do**
3: Subtract the sum of the variable utilities in n_i from n_i utility.
4: Establish inequalities arcoding to formula 3.5
5: **end for**

3.5. Giải bài toán thỏa mãn ràng buộc với Gurobi optimizer

3.5.1. Quy hoạch số nguyên (Integer programing)

Quy hoạch số nguyên viết tắt là IP, là bài toán quy hoạch mà trong đó một phần hoặc tất cả các biến chỉ nhận giá trị nguyên. IP có thể được giải bằng Gurobi Optimizer [9] - một trong những công cụ tối ưu toán học mạnh mẽ và nhanh nhất hiện nay. Các bài toán IP thường có dạng:

$$\begin{aligned} &\text{Hàm mục tiêu:} && \text{Cực trị } \mathbf{c}^T\mathbf{x} \\ &\text{Ràng buộc:} && \mathbf{A}\mathbf{x} = \mathbf{b} \text{ (tuyến tính)} \\ & && l \leq \mathbf{x} \leq u \text{ (cận)} \end{aligned}$$

Bài toán thỏa mãn ràng buộc (CSP) xây dựng cho việc ẩn dữ liệu có hàm mục tiêu là hàm tuyến tính. Dạng bài toán này thường được gọi là bài toán quy hoạch số nguyên tuyến tính (ILP). Thông thường ILP có thể được giải một cách tổng quát bằng phương pháp nhánh và cận dựa trên quy hoạch tuyến tính (linear programming based branch and bound). Phần tiếp theo sẽ giới thiệu cơ bản về phương pháp này.

3.5.2. Phương pháp nhánh và cận

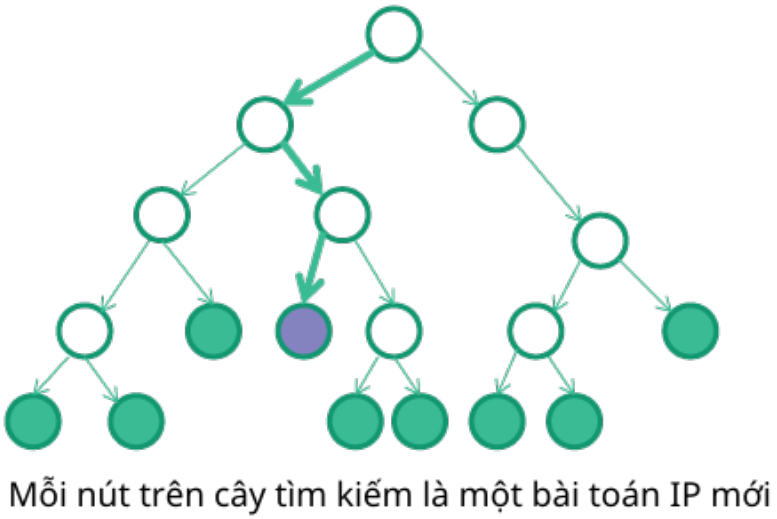
Bắt đầu với một bài toán IP mà ta không có cách giải trực tiếp. Nếu loại bỏ tất cả các ràng buộc nguyên, lúc này bài toán IP trở thành bài toán quy

hoạch tuyến tính (LP). Lời giải cho bài toán LP được gọi là lời giải nới lỏng (relaxation) của bài toán IP gốc. Nếu kết quả của LP thỏa mãn tất cả ràng buộc nguyên (mặc dù ta vẫn chưa sử dụng những ràng buộc này) thì khá may mắn ta có lời giải tối ưu cho bài toán IP. Nếu không, ta sẽ chọn ra các biến vi phạm ràng buộc nguyên trong LP relaxation. Để dễ hình dung, giả sử biến x có giá trị trong LP relaxation là 5.7. Ta loại bỏ giá trị này bằng cách sử dụng các ràng buộc như $x \leq 5.0$ hay $x \geq 6.0$.

Cho bài toán IP gốc ký hiệu là P_0, hai bài toán IP con mới lần lượt là P_1 và P_2 tương ứng với $x \leq 5.0$, $x \geq 6.0$. Biến x được gọi là biến phân nhánh (branching variable), và ta có thể nói rằng x có hai nhánh con tương ứng với hai bài toán IP P_1 và P_2. Nếu ta tính toán được lời giải tối ưu cho P_1 và P_2, lời giải tốt hơn trong hai lời giải sẽ là lời giải tối ưu cho bài toán gốc P_0. Hay nói cách khác ta đã thay thế P_0 bằng hai bài toán con cụ thể hơn P_1 và P_2. Ý tưởng tương tự cũng được áp dụng cho hai bài toán con mới, cũng sử dụng lời giải LP relaxation và lựa chọn biến phân nhánh nếu cần thiết. Quá trình này tạo thành một cây tìm kiếm. Các bài toán IP con là các nút trên cây với P_0 là nút gốc. Các nút lá là các nút chưa được phân nhánh. Về mặt tổng quát khi ta loại bỏ được tất cả các nút lá hay giải được tất cả bài toán ở nút lá thì cũng có nghĩa là ta đã giải quyết được bài toán gốc.

Nghiệm tối ưu của bài toán

Giả sử rằng hàm mục tiêu của ta là hàm tối thiểu và bên cạnh đó, ta đã có LP relaxation cho một số nút trên cây. Nếu tất cả ràng buộc nguyên trong mô hình IP gốc đều được thoả mãn thì ta đã tìm được lời giải khả thi cho bài toán. Ở đây ta sẽ thực hiện hai bước quan trọng. Đầu tiên ta chỉ định nút này là "xác nhận" (fathomed) và dĩ nhiên ta không cần thiết phải phân nhánh ở nút này. Tiếp theo ta phân tích lời giải vừa tìm được. Lời giải tốt nhất trong các lời giải khả thi được gọi là "nghiệm tối ưu". Ở thời điểm bắt đầu tìm lời giải ta sẽ không có nghiệm. Nếu lời giải khả thi có giá trị hàm mục tiêu thấp



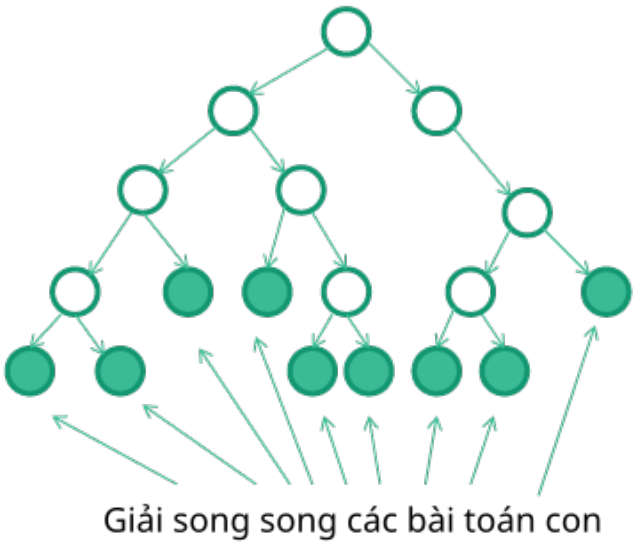
Hình 3.2: *Phương pháp nhánh và cận. [9]*

hơn nghiệm tối ưu hiện tại (hoặc nếu ta chưa có nghiệm) thì lời giải này sẽ được coi là nghiệm tối ưu mới.

Có hai trường hợp dẫn tới một nút được gọi là "xác nhận" (fathomed). Trường hợp một, các nhánh con của nút hiện tại không có LP relaxation. Hiển nhiên nút này không có LP relaxation hay bài toán IP không có lời giải. Trường hợp thứ hai, lời giải tối ưu cho bài toán IP ở nút này đã được tìm ra.

3.5.3. Gurobi Parallel Mixed Integer Programming

Bởi vì các nút khác nhau trong cây tìm kiếm có thể được xử lý độc lập nên ta có thể giải bài toán IP một cách song song. Bộ giải Gurobi MIP được thiết kế để thực hiện giải bài toán IP song song trên CPU. Tuy nhiên, việc giải bài toán song song ở nút gốc sẽ gặp hạn chế. Do đó, những bài toán có cây tìm kiếm lớn có thể khai thác khác các nhân của CPU hiệu quả hơn so với những bài toán mà lượng công việc phần lớn tập trung ở nút gốc.



Hình 3.3: *Phương pháp nhánh và cận - song song. [9]*

3.5.4. Xấp xỉ lời giải

Mặc dù ở bước tiền xử lý các tập NSHUI có khả năng tạo ra các ràng buộc dư thừa hoặc mâu thuẫn đã được loại bỏ, mô hình CSP không phải lúc nào cũng có lời giải. Do đó cần phải có chiến lược xấp xỉ lời giải một cách hiệu quả. Ở đây ta sử dụng cơ chế xấp xỉ lời giải được cài đặt sẵn trong Gurobi. Cụ thể ta coi các ràng buộc được tạo bởi bảng N-HIT (tương ứng với bất phương trình 3.5) là các ràng buộc có thể bị vi phạm và sử dụng 1.0 làm điểm phạt. Nếu một ràng buộc bị vi phạm 2.0 ta sẽ cộng vào hàm mục tiêu 2.0×1.0. Tối thiểu hóa hàm mục tiêu ta được lời giải xấp xỉ của bài toán CSP.

3.6. Ví dụ minh họa và kết quả thực nghiệm

3.6.1. Ví dụ minh họa

Quay trở lại với ví dụ ở bảng 1.1 với ngưỡng lợi ích thấp nhất $\delta = 124$ và itemset được chọn làm itemset nhạy cảm là $\{AB\}$, ta xây dựng bảng N-HIT (Bảng 3.2) và bảng S-HIT (Bảng 3.3). Để giảm quy mô bài toán, ta thực hiện

tiền xử lý trên bảng N-HIT. Bước đầu tiên, những itemset không bị ảnh hưởng bởi việc ẩn dữ liệu được xóa bỏ. Các itemset $\{D\}$, $\{DE\}$ không có chung item nào với itemset $\{AB\}$ được xóa khỏi N-HIT. Itemset $\{AC\}$ không xuất hiện trong bất kỳ transaction nào chung với $\{AB\}$ nên ta loại itemset $\{AC\}$. Tiếp theo ta có itemset $\{B\}$ unclosed với $\{AB\}$, loại bỏ $\{B\}$ để tránh mâu thuẫn khi tạo ràng buộc. Cuối cùng là tìm và xóa các itemset dư thừa để giảm quy mô bài toán. Itemset $\{BD\}$ unclosed với itemset $\{ABD\}$, ta loại bỏ itemset $\{ABD\}$. Bảng 3.4 là bảng N-HIT sau khi qua bước tiền xử lý.

Bảng 3.2: *Bảng S-HIT*

SHUI	Tidset	Itemset size	Itemset Utility
AB	$T_0T_1T_4T_6$	2	316

Bảng 3.3: *Bảng N-HIT*

NSHUI	Tidset	Itemset size	Itemset utility
A	$T_0T_1T_4T_6T_7T_8T_9$	1	351
B	$T_0T_1T_4T_6$	1	136
D	$T_0T_2T_3T_4T_5T_7$	1	156
AC	$T_7T_8T_9$	2	187
AE	T_0T_9	2	125
AD	$T_0T_4T_7$	2	213
BD	T_0T_4	2	124
DE	$T_0T_3T_5$	2	134
ABD	T_0T_4	3	214
$ABDE$	T_0	4	127

Bảng 3.4: *Bảng N-HIT sau tiền xử lý*

NSHUI	Tidset	Itemset size	Itemset utility
A	$T_0T_1T_4T_6T_7T_8T_9$	1	351
AD	$T_0T_4T_7$	2	213
AE	T_0T_9	2	125
BD	T_0T_4	2	124
$ABDE$	T_0	4	127

Sau tiền xử lý là bước thiết lập mô hình CSP. Cụ thể, chương trình duyệt qua bảng S-HIT (bảng 3.2) thiết lập các ràng buộc tương ứng với bất phương

trình 3.4 và xây dựng bảng hash lưu giữ các biến và utility của chúng. Tiếp theo, bảng N-HIT (bảng 3.4) được sử dụng cho việc xây dựng ràng buộc tương ứng với bất phương trình 3.5. Với itemset $\{A\}$, utility còn lại sau khi đã trừ bớt utility của A trong các transaction T_0, T_1, T_4 và T_6 là 171 > 124. Do đó ta không cần phải thiết lập ràng buộc cho tập $\{A\}$.

$$\begin{aligned}
&\min\ v_{00}+v_{10}+v_{40}+v_{60}+v_{01}+v_{11}+v_{41}+v_{61}\\
&\quad s.t. AB: 9v_{00}+9v_{10}+9v_{40}+9v_{60}+8v_{01}+8v_{11}+8v_{41}+8v_{61}<124\\
&\qquad AD: 9v_{00}+9v_{40}+123>=124\\
&\qquad AE: 9v_{00}+98>=124\\
&\qquad BD: 8v_{01}+8v_{41}+60>=124\\
&\qquad ABDE: 9v_{00}+8v_{01}+80>=124\\
&\qquad 1\le v_{00},v_{10},v_{40},v_{60},v_{01},v_{11},v_{41},v_{61}\le 9
\end{aligned}$$

Tuy nhiên mô hình CSP vừa được thiết lập không có lời giải chính xác, lời giải xấp xỉ được sử dụng để tạo cơ sở dữ liệu nhiễu D' (bảng 3.5).

Bảng 3.5: *Cơ sở dữ liệu nhiễu D'*

tid	A	B	C	D	E
0	3	5	0	4	5
1	1	1	0	0	0
2	0	0	4	6	0
3	0	0	6	4	3
4	1	1	0	6	0
5	0	0	0	3	9
6	1	1	0	0	0
7	5	0	2	3	0
8	8	0	6	0	0
9	6	0	8	0	6

Bảng 3.6 là kết quả khai thác mẫu hữu ích trên cơ sở dữ liệu nhiễu D'. Như vậy ở ví dụ này ta ẩn được itemset nhạy cảm $\{AB\}$; mất các NSHUI $\{B\}$ (bound-to-lose itemset), $\{ABDE\}$, $\{BD\}$; không có LUI nào trở thành HUI.

Bảng 3.6: *Bảng HIT của D'*

HUI	Tidset	Itemset size	Itemset utility
A	$T_0T_1T_4T_6T_7T_8T_9$	1	225
D	$T_0T_2T_3T_4T_5T_7$	1	156
AC	$T_7T_8T_9$	2	187
AE	T_0T_9	2	125
AD	$T_0T_4T_7$	2	159
DE	$T_0T_3T_5$	2	134
ABD	T_0T_4	3	144

Đánh giá kết quả thu được: $HF = 0$, $MC = 0.3$, $AC = 0$.

3.6.2. Kết quả thực nghiệm

Cơ sở dữ liệu và môi trường thực nghiệm

Kết quả thực nghiệm được thử nghiệm trên máy Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz, RAM 32GB; NVIDIA GEFORCE RTX2080, VRAM 8GB. Bốn cơ sở dữ liệu thực nghiệm [16] có đặc điểm như trong bảng 3.7 với density được tính bằng số lượng item trung bình trong một transaction chia số lượng item khác nhau: Density $= \frac{AvgLen}{|I|}$. Với mỗi dataset, external utility của mỗi item được phát sinh trong khoảng $[1,1000]$ dựa trên phân phối chuẩn log, internal utility của mỗi item phát sinh trong khoảng $[1,10]$ dựa trên phân phối đều. Thuật toán khai thác mẫu hữu ích được sử dụng là HUI-Miner [18]. Hiệu suất của thuật toán đề xuất phiên bản tuần tự (IPPUM-ILP(C)) và phiên bản song song trên GPU (IPPUM-ILP(G)) được đánh giá và so sánh với các thuật toán khác trên các bộ dữ liệu khác nhau và dựa trên 2 tiêu chí: thời gian thực thi, khả năng giảm thiểu ảnh hưởng phụ. Cụ thể các thuật toán lựa chọn so sánh bao gồm: HHUIF, MSICF [26], MSU-MAU, MSU-MIU [13]; chúng tôi không so sánh với 2 thuật toán PPUMGAT [**LIN2014**] và PPUMGAT+ [13] vì 2 thuật toán này ẩn thông tin theo hướng tiếp cận thêm và xóa transaction. Tất cả các thuật toán đều được cài đặt trên ngôn ngữ Python 3.7. Bộ giải Gurobi Optimization 9.0.2 [9] được sử dụng để tìm lời giải cho mô hình CSP.

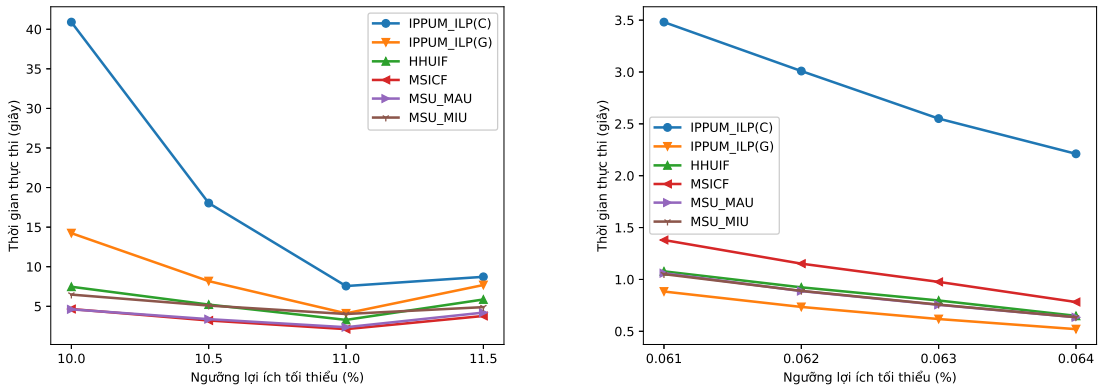
Bảng 3.7: *Đặc điểm của các cơ sở dữ liệu thực nghiệm*

Dataset	Số transaction	Số item	Density(%)
Mushrooms	8124	119	19.3
Foodmart	4141	1559	0.25
T25I10D10K	9976	893	2.66
T20I6D100K	99922	929	2.22

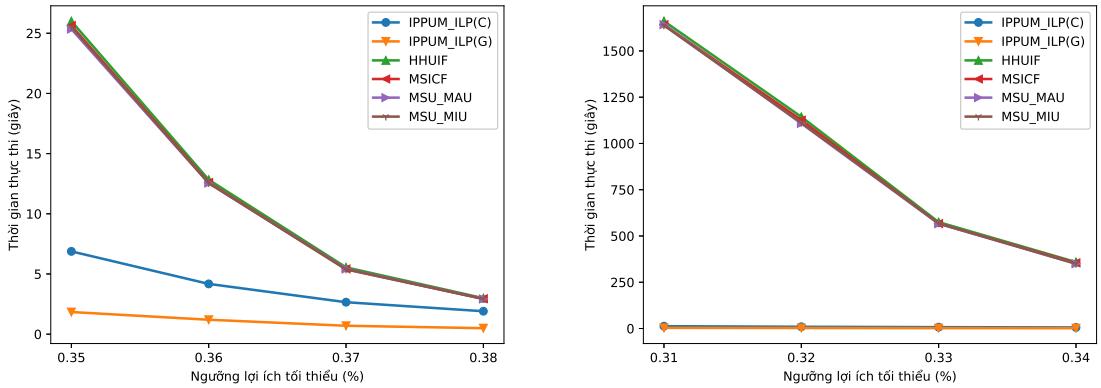
Thời gian thực thi

Thời gian thực thi của thuật toán được so sánh với các thuật toán khác khi tăng dần ngưỡng lợi ích tối thiểu - δ (hình 3.4) và tăng dần tỷ lệ thông tin nhạy cảm - SIP (hình 3.5). Quan sát kết quả ở hình 3.4, ta thấy hầu hết các trường hợp thời gian thực thi của các thuật toán giảm khi ngưỡng lợi ích tối thiểu δ tăng. Điều này tương đối dễ hiểu vì khi tăng δ số lượng itemset lợi ích cao giảm. Với tỷ lệ thông tin nhạy cảm cố định, số lượng SHUI cũng giảm theo. Do đó các phép tính toán và lặp để ẩn SHUI được giảm bớt. Ưu điểm của thuật toán cải tiến IPPUM-ILP được thể hiện rõ đối với các tập lớn và thưa. Lý do đơn giản là do các thuật toán khác cần phải duyệt dữ liệu nhiều lần để ẩn SHUI, kích thước dữ liệu càng lớn thì thời gian thực thi cũng càng tăng. Thuật toán IPPUM-ILP chỉ duyệt qua dữ liệu 2 lần, một lần duyệt để thiết lập các bảng S-HIT và N-HIT sau đó duyệt một lần nữa để ẩn SHUI. Hơn nữa, t25i10d10k và t20i6d100k là hai tập dữ liệu thưa, một item không xuất hiện trong quá nhiều transaction. Do đó các ràng buộc của mô hình CSP ít xung đột với nhau hơn dẫn đến thời gian giải ngắn hơn. Kết quả trên tập dữ liệu thưa và lớn cho thấy thời gian thực thi của thuật toán đề xuất trên cả hai phiên bản có thể thấp hơn các thuật toán khác đến 1500 lần. Đối với tập nhỏ và thưa như foodmart thì phiên bản song song của thuật toán IPPUM-ILP (IPPUM-ILP(G)) vẫn vượt trội hơn hẳn.

Hình 3.5 thể hiện thời gian thực thi của các thuật toán trên bốn tập dữ liệu với tỷ lệ thông tin nhạy cảm tăng dần. Từ kết quả ta thấy thời gian thực thi của các thuật toán nhìn chung là tăng khi tỷ lệ thông tin nhạy cảm tăng. Với



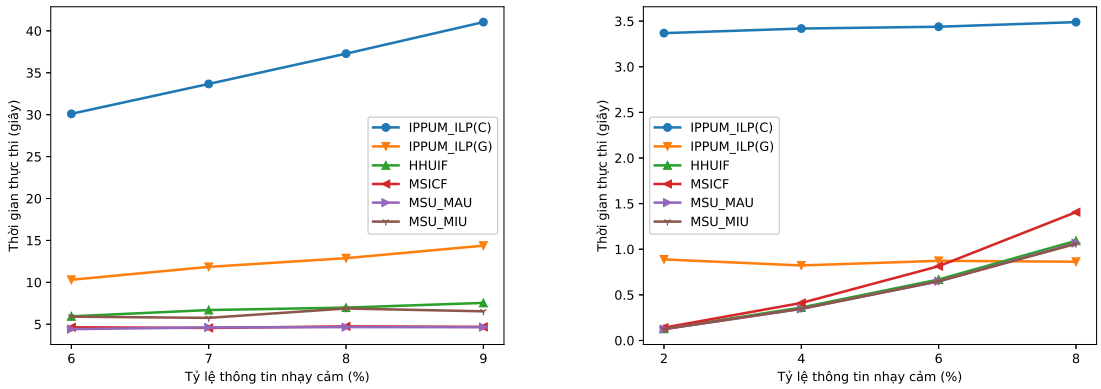
(a) *mushrooms* ***(b)*** *foodmart*



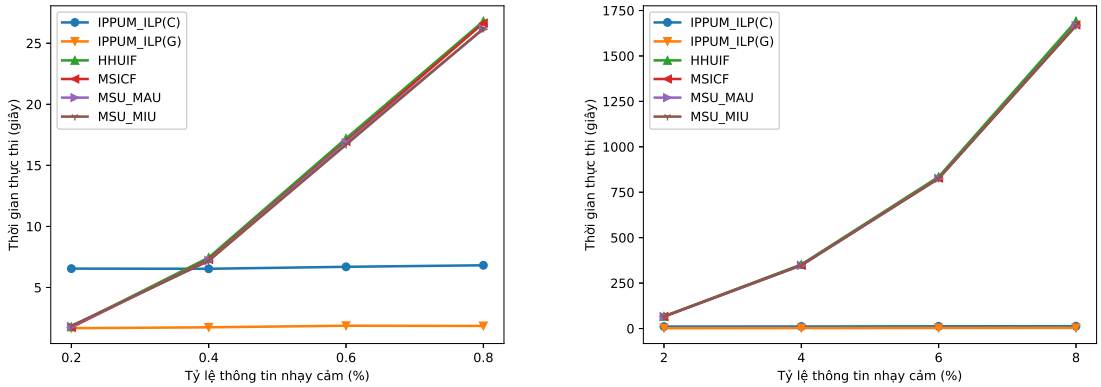
(c) *t25i10d10k* ***(d)*** *t20i6d100k*

Hình 3.4: *So sánh thời gian thực thi của các thuật toán với các ngưỡng lợi ích tối thiểu khác nhau.*

ngưỡng lợi ích tối thiểu cố định, khi ta tăng tỷ lệ thông tin nhạy cảm, số lượng SHUI cũng tăng theo. Do đó lượng công việc tính toán chắc chắn phải nhiều hơn đặc biệt là với các thuật toán heuristic. Đối với IPPUM-ILP do số lần duyệt dữ liệu là cố định, chi phí tiền xử lý chủ yếu phụ thuộc vào kích thước của tập các itemset lợi ích cao I_H nên thời gian thực thi không thay đổi nhiều. Với tập mushrooms (hình 3.5a) khi tăng số lượng SHUI, lượng ràng buộc ứng với bất phương trình 3.3 tăng dẫn đến chi phí giải bài toán CSP tăng theo.



(a) *mushrooms* ***(b)*** *foodmart*



(c) *t25i10d10k* ***(d)*** *t20i6d100k*

Hình 3.5: *So sánh thời gian thực thi của các thuật toán với tỷ lệ thông tin nhạy cảm khác nhau.*

Ảnh hưởng phụ

Để thấy rõ ảnh hưởng phụ của các thuật toán, ta áp dụng chúng trên các cơ sở dữ liệu với tỷ lệ thông tin nhạy cảm (SIP) khác nhau và sử dụng 3 độ đo đánh giá: hiding failure, missing cost, artificial cost. Các thuật toán đều ẩn giấu thành công tất cả SHUI nên ta sẽ tập trung vào missing cost và artificial cost. Ở bảng 3.8a không có artificial cost của bốn thuật toán HHUIF, MSICF, MSU-MAU, MSU-MIU vì bốn thuật toán này chỉ thực hiện giảm internal utility của item và xóa item trong transaction nên không tạo ra artificial cost. Nhìn chung missing cost của thuật toán IPPUM-ILP trên cả hai phiên bản đều rất

thấp và hầu như là vượt trội hơn hẳn các thuật toán khác. IPPUM-ILP chỉ thể hiện kém hơn thuật toán MSU-MIU trên tập mushrooms. Trên 3 tập dữ liệu foodmart, t25i10d10k, t20i6d100k ta hoàn toàn không mất đi bất kỳ tập lợi ích cao nào trong khi với tỷ lệ thông tin nhạy cảm cao thì các thuật toán còn lại mất đi gần như một nửa lượng thông tin. Hơn thế nữa lượng thông tin dư thừa do IPPUM-ILP tạo ra cũng rất thấp, cao nhất là trên tập t20i6d100k với artificial cost chỉ 2,65%.

Từ các thử nghiệm trên, hiệu suất của thuật toán IPPUM-ILP được chứng minh. Trong các tập dữ liệu lớn, IPPUM-ILP luôn thực thi nhanh hơn. Đối với các tập dữ liệu thưa, phiên bản song song IPPUM-ILP vượt trội hơn tất cả các thuật toán khác. Bên cạnh đó, ta có thể dễ dàng nhận thấy ảnh hưởng phụ tạo ra bởi các thuật toán heuristic trên các tập thưa là gần như giống nhau. Lý do là bởi vì chúng sử dụng cùng một heuristic để giảm số lượng hoặc loại bỏ các item trong các transaction được lựa chọn. Sự khác biệt giữa chúng nằm ở chiến lược lựa chọn các item và transaction cho việc chỉnh sửa. Trong các tập dữ liệu thưa, các SHUI hiếm khi có chung item và transaction với nhau; do đó các thuật toán heuristic cần phải lặp lại việc chỉnh sửa cho hầu hết các item và transaction của các SHUI để thực hiện quá trình ẩn dữ liệu. Nói cách khác, các thuật toán này sẽ thực hiện các tác vụ gần như giống nhau và dĩ nhiên là có thời gian thực thi và ảnh hưởng phụ tương tự nhau.

Bảng 3.8: *So sánh ảnh hưởng phụ của các thuật toán với tỷ lệ thông tin nhạy cảm khác nhau.*

(a) *Ảnh hưởng phụ của HHUIF, MSICF, MSU-MAU, MSU-MIU.*

Dataset	SIP (%)	HHUIF	MSICF	MSU-MAU	MSU-MIU
		MC(%)	MC(%)	MC(%)	MC(%)
mushrooms	6.00	11.57	11.91	11.52	10.33
	7.00	10.73	10.96	10.60	9.31
	8.00	9.76	9.89	9.62	8.78
	9.00	8.87	9.22	8.63	7.69
foodmart	2.00	0.26	0.26	0.26	0.26
	4.00	1.08	1.08	1.08	1.08
	6.00	29.77	29.77	29.77	29.77
	8.00	39.74	39.74	39.74	39.74
t25i10d10k	0.20	0.08	0.08	0.08	0.08
	0.40	20.09	20.09	20.09	20.09
	0.60	39.95	39.95	39.95	39.95
	0.80	40.03	40.03	40.03	40.03
t20i6d100k	2.00	0.83	0.83	0.83	0.83
	4.00	4.88	4.88	4.88	4.88
	6.00	19.52	19.52	19.52	19.52
	8.00	35.25	35.25	35.25	35.25

(b) *Ảnh hưởng phụ của IPPUM-ILP.*

Dataset	SIP (%)	IPPUM-ILP(C)		IPPUM-ILP(G)	
		MC(%)	AC(%)	MC(%)	AC(%)
mushrooms	6.00	10.98	2.61	10.98	2.20
	7.00	10.02	2.17	10.02	2.37
	8.00	9.04	2.24	9.04	2.20
	9.00	8.04	2.24	8.04	2.20
foodmart	2.00	0.00	0.05	0.00	0.05
	4.00	0.00	0.10	0.00	0.10
	6.00	0.00	0.80	0.00	0.96
	8.00	0.00	1.81	0.00	1.81
t25i10d10k	0.20	0.00	0.08	0.00	0.08
	0.40	0.00	1.00	0.00	1.00
	0.60	0.00	1.65	0.00	1.65
	0.80	0.00	1.65	0.00	1.65
t20i6d100k	2.00	0.00	0.82	0.00	0.82
	4.00	0.00	0.82	0.00	0.82
	6.00	0.00	1.02	0.00	1.02
	8.00	0.00	2.65	0.00	2.65

CHƯƠNG 4
KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

4.1. Kết luận

Bảo vệ thông tin trong khai thác dữ liệu nói chung và bảo vệ thông tin riêng tư trong khai thác mẫu hữu ích nói riêng là vấn đề quan trọng trong thời đại bùng nổ thông tin hiện nay. Tuy nhiên, việc ẩn giấu thông tin nhạy cảm trong khi vẫn giữ lại được các thông tin hữu ích là một bài toán khó cần phải giải quyết [**LIN2014**]. Một hướng tiếp cận khá mới là ứng dụng quy hoạch số nguyên vào việc tìm lời giải cho bài toán ẩn dữ liệu (PPUM-ILP). Bằng cách mô hình hóa quy trình ẩn dữ liệu thành một bài toán thỏa mãn ràng buộc (CSP) thay vì sử dụng heuristic như các phương pháp trước đó, việc chỉnh sửa dữ liệu trở nên chính xác hơn. Mặc dù vậy, thời gian thực thi của PPUM-ILP vẫn chưa ổn định và việc xấp xỉ lời giải bài toán CSP cũng chưa hiệu quả. Nhận thấy tính cấp thiết của vấn đề PPUM cũng như lấy ý tưởng từ hướng tiếp cận quy hoạch số nguyên, luận văn đã tìm hiểu, nghiên cứu các nghiên cứu liên quan để đề xuất một phương pháp cải tiến. Cụ thể luận văn đã hoàn thành những công việc sau:

- Tìm hiểu các khái niệm về bài toán khai thác mẫu hữu ích.
- Tìm hiểu các thuật toán khai thác mẫu hữu ích.
- Tìm hiểu các khái niệm về bài toán bảo vệ tính riêng tư trong khai thác mẫu hữu ích.
- Tìm hiểu các phương pháp, hướng tiếp cận cho bài toán bảo vệ tính riêng tư trong khai thác mẫu hữu ích.

- Tiến hành cài đặt phương pháp khai thác mẫu hữu ích HUI-Miner.
- Tiến hành cài đặt và kiểm chứng các phương pháp bảo vệ tính riêng tư trong khai thác mẫu hữu ích: HHUIF, MSICF, MSU-MAU, MSU-MIU.
- Đề xuất phương pháp cải tiến IPPUM-ILP với các chiến lược tiền xử lý, thiết lập ràng buộc và xấp xỉ lời giải nhanh và hiệu quả hơn.
- Tiến hành nhiều thực nghiệm để chứng minh hiệu quả của phương pháp đề xuất.

4.2. Hướng phát triển

Từ các kết quả cũng như nhận định trên, trong tương lai đề tài có thể phát triển theo những hướng sau:

- Việc ứng dụng GPU vào tiền xử lý cũng còn một số hạn chế. Cách biểu diễn dữ liệu cho việc tiền xử lý trên GPU vẫn còn lãng phí bộ nhớ, trên tập thưa số lượng bit 0 trong một BitList nhiều. Chính vì vậy ta cần phải nghiên cứu cách biểu diễn dữ liệu để giảm không gian bộ nhớ.
- Trong tương lai, ta cần phải xác định thêm các itemset phục vụ cho việc thiết lập các ràng buộc cụ thể để giảm artificial cost của quá trình làm nhiễu dữ liệu.
- Ứng dụng các kỹ thuật tương tự cho bài toán hẹp hơn: Bảo vệ tính riêng tư trong khai thác tập phổ biến.

TÀI LIỆU THAM KHẢO

Tiếng Anh:

[1] Mohammad Al-Rubaie and J Morris Chang (2019), "Privacy-preserving machine learning: Threats and solutions", *IEEE Security & Privacy*, 17 (2), pp. 49–58.

[2] Raymond Chan, Qiang Yang, and Yi-Dong Shen, "Mining High Utility Itemsets", in: *Proceedings of the Third IEEE International Conference on Data Mining*, ICDM '03, USA, 2003, p. 19.

[3] Duy-Tai Dinh et al., "A Survey of Privacy Preserving Utility Mining", in: *High-Utility Pattern Mining: Theory, Algorithms and Applications*, ed. by Philippe Fournier-Viger et al., Cham, 2019, pp. 207–232.

[4] Jie Dong and Min Han (2007), "BitTableFI: An efficient mining frequent itemsets algorithm", *Knowledge-Based Systems*, 20 (4), pp. 329–335.

[5] Philippe Fournier-Viger et al., "FHM: Faster High-Utility Itemset Mining Using Estimated Utility Co-occurrence Pruning", in: *Foundations of Intelligent Systems*, ed. by Troels Andreasen et al., Cham, 2014, pp. 83–92.

[6] Benjamin C. M. Fung et al. (2010), "Privacy-preserving data publishing: A survey of recent developments", *ACM Comput. Surv.*, 42 (4), 14:1–14:53, ISSN: 0360-0300, DOI: 10.1145/1749603.1749605, URL: https://www.bibsonomy.org/bibtex/2076319ebf0f40ffa5c906ca6216079a5/matthiashuber.

[7] W. Gan et al., "Privacy Preserving Utility Mining: A Survey", in: *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 2617–2626.

[8] Wensheng Gan et al. (2020), "Utility-Driven Mining of Trend Information for Intelligent System", *ACM Transactions on Management Information Systems*, 11 (3).

[9] LLC Gurobi Optimization, *Gurobi Optimizer Reference Manual*, 2020, URL: http://www.gurobi.com.

[10] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz, "Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning.", in: *ACM Conference on Computer and Communications Security*, ed. by Bhavani M. Thuraisingham et al., ACM, 2017, pp. 603–618, ISBN: 978-1-4503-4946-8, URL: http://dblp.uni-trier.de/db/conf/ccs/ccs2017.html#HitajAP17;https://doi.org/10.1145/3133956.3134012;https://www.bibsonomy.org/bibtex/25e8beef70561de70daa3e8b177e8dblp.

[11] Shaoxin Li et al. (2019), "A novel algorithm for privacy preserving utility mining based on integer linear programming", *Engineering Applications of Artificial Intelligence*, 81, pp. 300–312.

[12] Chun-Wei Lin, Tzung-Pei Hong, and Wen-Hsiang Lu (2011), "An effective tree structure for mining high utility itemsets", *Expert Systems with Applications*, 38 (6), pp. 7419–7424.

[13] Chun-Wei Lin et al. (2016), "Efficient algorithms for mining high-utility itemsets in uncertain databases", *Knowledge-Based Systems*, 96, pp. 171–187.

[14] Jerry Chun-Wei Lin, Philippe Fournier-Viger, and Wensheng Gan (2016), "FHN: An efficient algorithm for mining high-utility itemsets with negative unit profits", *Knowledge-Based Systems*, 111, pp. 283–298.

[15] Jerry Chun-Wei Lin et al., "Mining High-Utility Itemsets with Multiple Minimum Utility Thresholds", in: *Proceedings of the Eighth Interna-*

tional C Conference on Computer Science & Software Engineering*, C3S2E '15, Yokohama, Japan, 2015, pp. 9–17.

[16] Jerry Chun-Wei Lin et al. (2016), "Fast algorithms for hiding sensitive high-utility itemsets in privacy-preserving utility mining", *Engineering Applications of Artificial Intelligence*, 55, pp. 269–284.

[17] Jerry Chun-Wei Lin et al. (2017), "Efficient hiding of confidential high-utility itemsets with minimal side effects", *Journal of Experimental & Theoretical Artificial Intelligence*, 29 (6), pp. 1225–1245.

[18] Mengchi Liu and Junfeng Qu, "Mining High Utility Itemsets without Candidate Generation", in: *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, Maui, Hawaii, USA, 2012, pp. 55–64.

[19] Xuan Liu et al. (2020), "An Improved Sanitization Algorithm in Privacy-Preserving Utility Mining", *Mathematical Problems in Engineering*, 2020, pp. 1–14.

[20] Ying Liu, Wei-Keng Liao, and Alok Choudhary, "A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets", in: *Advances in Knowledge Discovery and Data Mining*, ed. by Tu Bao Ho, David Cheung, and Huan Liu, Berlin, Heidelberg, 2005, pp. 689–695.

[21] Ron Rymon, "Search through systematic set enumeration", in: *Proceedings of the Third International Conference. Principles of Knowledge Representation and Reasoning*, 1992, pp. 539–550.

[22] Tin Truong et al. (2020), "EHAUSM: An efficient algorithm for high average utility sequence mining", *Information Sciences*, 515, pp. 302–323.

[23] Cheng-Wei Wu; Philippe Fournier-Viger; Jia-Yuan Gu; Vincent S. Tseng, "Mining closed+ high utility itemsets without candidate generation", in: *2015 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, 2015, pp. 187–194.

[24] Hong Yao, Howard Hamilton, and Cory Butz, "A Foundational Approach to Mining Itemset Utilities from Databases", in: *Proceedings of the 2004 SIAM International Conference on Data Mining*, vol. 4, 2004.

[25] Hong Yao and Howard J. Hamilton (2006), "Mining itemset utilities from transaction databases", *Data & Knowledge Engineering*, 59 (3), Including: ER 2003, pp. 603–626.

[26] Jieh-Shan Yeh and Po-Chiang Hsu (2010), "HHUIF and MSICF: Novel algorithms for privacy preserving utility mining", *Expert Systems with Applications*, 37 (7), pp. 4779–4786.

[27] Unil Yun and Donggyu Kim, "Analysis of Privacy Preserving Approaches in High Utility Pattern Mining", in: *Advances in Computer Science and Ubiquitous Computing*, ed. by James J. (Jong Hyuk) Park et al., Singapore, 2017, pp. 883–887.

[28] Unil Yun and Jiwon Kim (2015), "A fast perturbation algorithm using tree structure for privacy preserving utility mining", *Expert Systems with Applications*, 42 (3), pp. 1149–1165.