DOI: 10.15625/vap.2020.00152

ĐÁNH GIÁ CÁC THUẬT TOÁN RÚT TRÍCH PHỤ THUỘC HÀM: TANE, FUN FD_MINE VÀ PHỤ THUỘC HÀM PHẢ HỆ TRI THỨC

Trịnh Minh Quốc Trung[1], Nguyễn Đức Thuần[2]

[1]Trung tâm Công nghệ thông tin - Viễn thông Khánh Hòa

[2] Khoa Công nghệ thông tin - Trường Đại học Nha Trang

trungtmq@gmail.com, thuan.inf@ntu.edu.vn

***TÓM TẮT*:** *Phát hiện phụ thuộc hàm là một bước cơ bản trong tổ chức và thiết kế cơ sở dữ liệu quan hệ. Tập phụ thuộc hàm càng được xác định đầy đủ giúp việc rút gọn, phân rã và chuẩn hóa cơ sở dữ liệu càng hiệu quả. Một số thuật toán phát hiện tập phụ thuộc hàm cổ điển đã được đề xuất như TANE, FUN, FD_MINE,.. Ngày nay, xu hướng tổ chức dữ liệu nhằm rút trích thông tin theo ngữ nghĩa ngày càng được quan tâm. Sự khác biệt lớn nhất dữ cơ sở dữ liệu quan hệ và cơ sở có ràng buộc ngữ nghĩa là mối quan hệ miền thuộc tính. Việc mở rộng khái niệm phụ thuộc hàm cổ điển thành phụ thuộc hàm theo phả hệ tri thức là một yêu cầu tất yếu. Trong bài báo này, chúng tôi đánh giá và thử nghiệm lại các thuật toán phát hiện phụ thuộc hàm cổ điển, khảo sát các thuật toán phát hiện phụ thuộc hàm tri thức nhằm tổng quan được các thuật toán phát hiện phụ thuộc hàm hiện nay.*

Từ khóa: *Phụ thuộc hàm, thuật toán rút trích phụ thuộc hàm, cây phả hệ tri thức, phụ thuộc hàm phả hệ tri thức.*

I. GIỚI THIỆU

Phụ thuộc hàm thể hiện sự ràng buộc giữa các tập thuộc tính dữ liệu. Phát hiện phụ thuộc hàm là vô cùng quan trọng nhằm làm đảm bảo tính nhất quán dữ liệu, chuẩn hóa lược đồ quan hệ, tối ưu hóa xây dựng các mô hình dữ liệu.

Phát hiện phụ thuộc hàm luôn là thách thức cho các nhà nghiên cứu cơ sở dữ liệu. Trong bài báo này, chúng tôi mô tả, đánh giá và so sánh các thuật toán phát hiện các phụ thuộc hàm phổ biến TANE, FUN và FD_Mine trên các bộ dữ liệu khác nhau của UCI để thấy được ưu nhược điểm của các thuật toán.

Bài toán xử lý dữ liệu dựa trên ngữ nghĩa trong thời gian gần đây được nhiều người quan tâm. Xử lý dữ liệu ngữ nghĩa có hiệu năng cao khi tiếp cận các phụ thuộc hàm mở rộng gọi là phụ thuộc hàm phả hệ tri thức. Các phụ thuộc hàm này được phát hiện dựa trên cấu trúc cây phả hệ tri thức. Các thuật toán phát hiện phụ thuộc hàm cổ điển không thể phát hiện ra các phụ thuộc hàm này mà phải dựa vào một thuật toán chuyên dụng hiệu năng cao.

Trong bài báo này chúng tôi khảo sát thuật toán FASTOFD phát hiện phụ thuộc hàm phả hệ tri thức, chỉ ra ưu nhược điểm và cải tiến thuật toán này.

II. CƠ SỞ LÝ THUYẾT

A. Các thuật toán rút trích phụ thuộc hàm (FD) cổ điển

1. Thuật toán TANE [1]

Thuật toán TANE là thuật toán được xây dựng bởi Huhtala dựa trên 3 ý tưởng chính: phân đoạn dữ liệu để kiểm tra các phụ thuộc hàm, (apriori-gen) chỉ tìm tất cả các phụ thuộc hàm tối thiểu và sử dụng các luật rút gọn không gian tìm kiếm.

Thuật toán:

Thuật toán TANE

Input: quan hệ r trên lược đồ quan hệ R
Output: tập các phụ thuộc hàm trong r
1: $L_0 = \{\emptyset\}$
2: $C^+(\emptyset) = R$
3: $L_1 = \{\{A\}|A \in R\}$
4: l =1
5: **while** $L_l \neq \emptyset$
6: *COMPUTE_DEPENDENCIES*(L_l)
7: *PRUNE*(L_l)
8: L_{l+1} = *GENERATE_NEXT_LEVEL*(L_l)
9: l = l + 1

Trong thuật toán này thủ tục *COMPUTE_DEPENDENCIES* dùng để tìm các phụ thuộc hàm trong tập ứng viên L_l, thủ tục *PRUNE* rút gọn tập ứng viên L_l, thủ tục *GENERATE_NEXT_LEVEL* sẽ tạo ra tập ứng viên mới L_{l+1} dựa vào tập ứng viên đã được rút gọn L_l.

2. Thuật toán FUN [2]

Thuật toán FUN là thuật toán được xây dựng bởi Novelli và Cicchetti. Tương tự thuật toán TANE, thuật toán FUN sử dụng tập tự do (Free set) là tập các thuộc tính không chứa phần tử nào mà phụ thuộc hàm vào tập con của tập thuộc tính còn lại nhằm rút gọn không gian tìm kiếm các phụ thuộc hàm.

Thuật toán:

Thuật toán FUN

Input: quan hệ r trên lược đồ quan hệ R
Output: tập các phụ thuộc hàm trong r
1: $L_0 = <\emptyset, 0, \emptyset, \emptyset>$
2: $L_1 = \{<A, Count(A), A, A> \mid A \in R\}$
3: R' = R - {A|A is a key}
4: **for** (k = 1; $L_k \neq \emptyset$; k = k + 1) **do**
5: *ComputeClosure*(L_{k-1}, L_k)
6: *ComputeQuasiClosure*(L_k, L_{k-1})
7: *DisplayFD*(L_{k-1})
8: *PurePrune*(L_k, L_{k-1})
9: L_{l+1} = *GenerateCandidate*(L_k)
10: *DisplayFD*(L_{k-1})

Trong thuật toán này *Count* là thủ tục đếm tần suất xuất hiện của tập thuộc tính A trong r, *ComputeClosure* là thủ tục tính bao đóng của tập ứng viên L_k dựa vào tập ứng viên trước đó là L_{k-1}, *ComputeQuasiClosure* là thủ tục tìm tập hợp các thuộc tính của tập ứng viên L_k với bao đóng của tập ứng viên L_{k-1}, *DisplayFD* là thủ tục hiển thị tập các phụ thuộc hàm đã tìm được, *PurePrune* là thủ tục rút gọn tập ứng viên L_k dựa vào tập ứng viên L_{k-1} và *GenerateCandidate* là thủ tục tạo tập ứng viên L_{l+1} dựa vào tập ứng viên L_k.

3. Thuật toán FD_MINE [3]

FD_MINE là thuật toán được xây dựng bởi Yao. Giống như TANE và FUN thì FD_MINE duyệt qua dàn (lattice) tổ hợp các thuộc tính từ dưới lên (bottom-up), sử dụng phân đoạn để khai phá các phụ thuộc hàm. FD_Mine xây dựng luật rút gọn dựa trên ý tưởng các lớp tương đương của tập thuộc tính. (Hai thuộc tính được gọi là tương đương nhau khi chúng xác định hàm lần nhau).

Thuật toán:

Thuật toán FD_MINE

Input: quan hệ r trên lược đồ quan hệ R
Output: FD_SET, EQ_SET và KEY_SET
1: Initialization Step
set R = {X_1, X_2,…, X_m}, **set** FD_SET = ∅
set EQ_SET = ∅, **set** KEY_SET = ∅
set CANDIDATE_SET = {X_1, X_2,…, X_m}
for all X_i ∈ CANDIDATE_SET **do**
set Closure'[X_i] = ∅
2: Iteration step
while CANDIDATE_SET ≠ ∅ **do**
for all X_i ∈ CANDIDATE_SET **do**
ComputeNoTrivialClosure(X_i)
ObtaintFDandKey(X_i)
ObtainEQSet(CANDIDATE_SET)
PruneCandidates(CANDIDATE_SET)
GenerateCandidates(CANDIDATE_SET)
3: *Display*(FD_SET, EQ_SET, KEY_SET)

Trong thuật toán trên, CANDIDATE_SET là tập các ứng viên, EQ_SET là tập các thuộc tính tương đương nhau, KEY_SET là tập các khóa phát hiện được, ComputeNoTrivialClosure là thủ tục để tính bao đóng của các ứng viên, ObtaintFDandKey là thủ tục xác định phụ thuộc hàm và khóa, ObtainEQSet là thủ tục tìm tập các thuộc tính tương đương nhau, PruneCandidates là thủ tục rút gọn tập ứng viên qua mỗi vòng lập, GenerateCandidates là thủ tục rút gọn tập ứng viên và Display là thủ tục hiển thị kết quả: FD_SET, EQ_SET, KEY_SET.

B. Thuật toán phát hiện phụ thuộc hàm phả hệ tri thức [4-8]

Khi các giá trị ứng với một thuộc tính dữ liệu đầu vào có sự liên quan về mặt ngữ nghĩa, các thuật toán phát hiện phụ thuộc hàm cổ điển không thể phát hiện đầy đủ các phụ thuộc hàm, mặc dù các tập thuộc tính này đã được mô hình hóa tương đương về ngữ nghĩa dựa vào một cây phả hệ tri thức (ontology).

Phụ thuộc hàm phả hệ tri thức (OFDs) được khám phá dựa trên các mối quan hệ ngữ nghĩa của các thuộc tính như: quan hệ tương đương (*synonyms*) và kế thừa (*inheritance*).

4. Phụ thuộc hàm phả hệ tri thức: Để hiểu rõ phụ thuộc hàm tri thức ta xét bảng dữ liệu sau

Bảng 1. Bảng dữ liệu 1

	Country Code	**Country**	**Symptom**	**Diagnosis**	**Medicine**
t_1	US	United states	joint pain	osteoarthritis	ibuprofen
t_2	IN	India	joint pain	osteoarthritis	NSAID
t_3	CA	Canada	joint pain	osteoarthritis	naproxen
t_4	IN	Bharat	nausea	migrane	analgesic
t_5	US	America	nausea	migrane	tylenol
t_6	US	USA	nausea	migrane	acetaminophen
t_7	IN	India	chest pain	hypertension	morphine

Đối với bảng dữ liệu này các thuật toán phát hiện phụ thuộc hàm cổ điển không thể phát hiện phụ thuộc hàm {Country Code} → {Country} vì các dữ liệu {'United states', 'USA', 'America'} là không giống nhau trong khi đó {'United states', 'USA', 'America'} đều là tên thể hiện của một quốc gia. Tương tự, cũng không thể phát hiện phụ thuộc hàm {Sumpton, Diagnosis} → {Medicine} vì các dữ liệu {'ibuprofen', 'NSAID', 'naproxen'} không giống nhau trong khi đó 'ibuprofen' và 'naproxen' là một loại thuốc của 'NSAID'.

Gọi E_i là các lớp thể hiện, ký hiệu *synonyms*(E_i) là tập tất cả các thể hiện tương đương của lớp E_i, thì *synonyms*(E_1) = {'United states', 'USA', 'America'}, synonyms(E_i) = {'India', 'Bharat'}.

Gọi C là một thể hiện, ký hiệu *names*(C) là tập tất cả các lớp mà có sự thể hiện của C, thì *names*('USA') = {E_1}, *names*('India') = {E_2}.

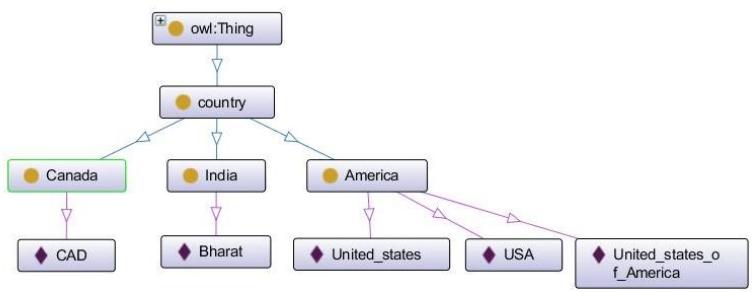
Ký hiệu, *descendants*(E) là một của các thể hiện của lớp E hoặc các lớp con cháu của nó, thì *descendants* (E)= { $E|E_n$ 'is-a' E_{n-1},…, E_1 'is-a' E}.

a) Phụ thuộc hàm phả hệ tri thức tương đương (*synonym OFD*) [6]

Định nghĩa: Một quan hệ r thỏa mãn *synonym OFD* $X \rightarrow_{syn} Y$ nếu mỗi thuộc tính $A \in Y$, mỗi $s \in \Pi_X(r)$, thì tồn tại một lớp E sao cho $\Pi_A(g_s[X]) \subseteq$ *synonyms*(E), với $g_s[X] = \{t|t \in r$ và $t[X] = s\}$.

$$\bigcap_{names(a), a \in \{t[A] | t \in s\}} \neq \emptyset$$

Cây phả hệ tri thức về country như sau ứng với bảng dữ liệu trên:



Hình 1. Cây phả hệ tri thức về country

Ta có {Country code} $\rightarrow_{syn}$ {Country}, $\Pi_{\{Country\ code\}}$ ={{ t_1, t_5, t_6}, { t_2, t_4, t_7}, {t_3}} do

- {t_1, t_5, t_6}: names('United states') ∩ names('America') ∩ names('USA') = 'America'.
- {t_2, t_4, t_7}: names('India') ∩ names('Bharat') = 'India'.
- {t_3}: chứa tập giá trị đơn nên không bị xung đột.

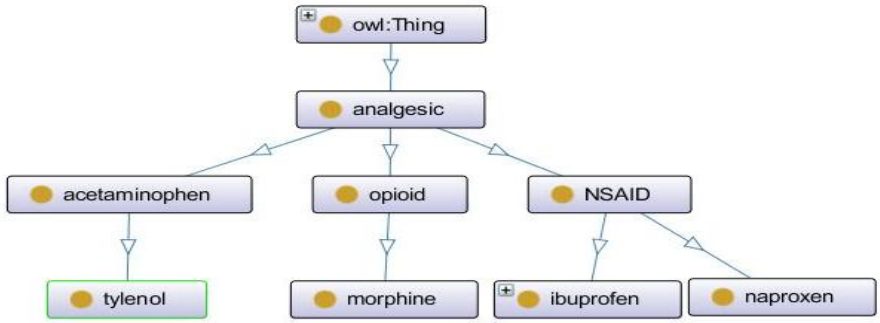
Vì vậy, theo định nghĩa có *synonym OFD* là {Country code} $\rightarrow_{syn}$ {Country}.

b) Phụ thuộc hàm phả hệ tri thức kế thừa (*inheritance OFD*) [5]

Định nghĩa: Ký hiệu LCA là tổ tiên (*descendant*) chung thấp nhất của các lớp, θ là ngưỡng tương ứng với chiều dài đường dẫn theo kế thừa của các lớp. Một quan hệ r thỏa mãn *inheritance OFD* $X \rightarrow_{inh} Y$ nếu mỗi thuộc tính $A \in Y$, mỗi $s \in \Pi_X(r)$, thì LCA(*descendants* (*names*(t_1)),…, *descendants* (*names* (t_n))) $\leq \theta$.

$$\bigcap_{descendants(names(a)),a\in\{t[A]|t\in s\}} \neq \emptyset$$

Từ ví dụ trên có thể xây dựng cây phả hệ tri thức về medicine như sau:



Hình 2. Cây phả hệ tri thức về medicine

Theo ví dụ trên, ta có {Sumpton, Diagnosis} $\rightarrow_{inh}$ {Medicine}. $\Pi_{\{Sumpton,Diagnosis\}}$ ={{t_1, t_2, t_3}, {t_4, t_5, t_6}, {t_7}}

Do dựa vào cây phả hệ tri thức đã xây dựng, ta có:

- {t_1, t_2, t_3}: descendant(names('ibuprofen')) ∩ descendant(names('NSAID')) ∩ descendant(names('naproxen')) = 'NSAID'.
- {t_4, t_5, t_6}: descendant(names('analgesic')) ∩ descendant(names('tylenol')) ∩ descendant(names('acetaminophen')) = 'analgesic.
- {t_7}: chứa tập giá trị đơn nên không bị xung đột.

Vì vậy, theo định nghĩa phát hiện được *inheritance OFD* {Sumpton, Diagnosis} $\rightarrow_{inh}$ {Medicine} với ngưỡng $\theta = 2$.

5. Thuật toán FASTOFD [5]

FASTOFD là thuật toán dùng để tìm các phụ thuộc hàm phả hệ tri thức, bằng cách dựa vào tập các thuộc tính để xây dựng tập ứng viên qua các mức và tính bao đóng của các ứng viên.

Thuật toán FASTOFD

Input: Quan hệ r
Ouput: Tập các phụ thuộc hàm phả hệ tri thức
1: L_0 = {}, M ={}
2: C^+({}) = R
3: L_1 = { A | A ∈ R}
4: l =1
5: **while** $L_l \neq$ {} **do**
6: *computeOFDs*(L_l)
7: L_{l+1} = *calculateNextLevel*(L_l)
8: l = l + 1
9: **return** M

a) Thuật toán tìm bao đóng

Thuật toán Tìm bao đóng

Input: Tập các phụ thuộc hàm phả hệ tri thức M và tập các thuộc tính X
Ouput: Bao đóng của X
1: $M_{unused} \leftarrow M$
2: $n \leftarrow 0$
3: $X^n \leftarrow X$
4: loop
5: **if** $\exists V \rightarrow Z \in M_{unused}$ and $V \subseteq X$ **then**
6: $X^{n+1} \leftarrow X^n \cup Z$
7: $M_{unused} \leftarrow M_{unused} \setminus \{V \rightarrow Z\}$
8: $n \leftarrow n + 1$
9: **else**
10: **return** X^n
11: **end loop**

Thuật toán dùng để tìm bao đóng của tập thuộc tính X dựa vào tập các phụ thuộc hàm phả hệ tri thức M đã tìm được. Bằng cách duyệt qua các phụ thuộc hàm trong M: $V \rightarrow Z \in M_{unused}$ và $V \subseteq X$ thì thêm Z và M_{unused} loại phụ thuộc hàm này. Kết quả ta được bao đóng của X.

b) Thủ tục calculateNextLevel

Thủ tục calculateNextLevel

1: $L_{l+1} = \{\}$
2: **for all** {YB, YC} ∈ *singleAttrDiffBlocks*(L_l) **do**
3: X = Y ∪ {B, C}
4: Add X to L_{l+1}
5: **return** L_{l+1}

Thủ tục xây dựng tập ứng viên ở mức kế tiếp dựa vào việc tổ hợp từ tập các thuộc tính đơn được tìm từ thủ tục con *singleAttrDiffBlocks* của tập ứng L_l.

c) Thủ tục computeOFDs

Thủ tục computeOFDs

1: **for all** $X \in (L_l)$ **do**
2: $C^+(X) = \bigcap_{A \in X} C^+(X \setminus A)$
3: **for all** $X \in (L_l)$ **do**
4: **for all** $A \in X \cap C^+(X)$ **do**
5: **if** $X \setminus A \rightarrow A$ **then**
6: Add $X \setminus A \rightarrow A$ to M
7: Remove A from $C^+(X)$

ComputeOFDs là thủ tục dùng để xác định các phụ thuộc hàm phả hệ tri thức dựa vào việc tìm bao đóng của các ứng viên.

III. KẾT QUẢ

A. Cài đặt

a) Dữ liệu:

Nhóm đã tiến hành cài đặt các thuật toán TANE, FUN, FD_MINE trên các bộ dữ liệu UCI: abalone, breast-cancer, iris, nursery, breast-cancer-wisconsin, balance-scale, bridges. Bên cạnh đó, để so sánh về tốc độ và việc khai phá phụ thuộc hàm trong các trường hợp khác nhau thì nhóm đã sử dụng bộ dữ liệu ncovter thay đổi số lượng thuộc tính để tạo ra các bộ dữ liệu con với số lượng thuộc tính khác nhau và sử dụng bộ dữ liệu nursery thay đổi số lượng dòng dữ liệu thay đổi.

b) Cấu hình:

Tất cả các thuật toán được thực thi trên máy tính Dell Vostro 5468 được cài đặt hệ điều hành Windows 10 với cấu hình chip Intel(R) Core i7-7500, 4 CPUs, 2.7GHz, RAM DDR4 2400 MHz dung lượng 16GB.

B. Bảng đánh giá

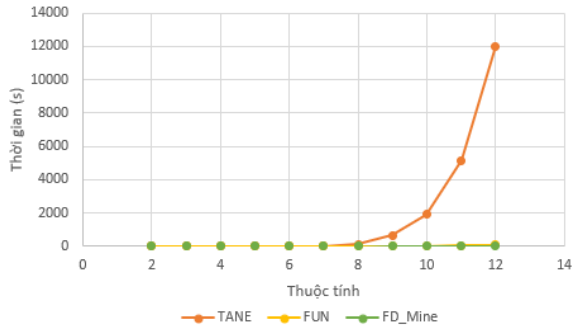
a) Bảng so sánh gữa các thuật toán khi áp dụng các bộ dữ liệu khác nhau:

Bảng 2. Bảng so sánh về thời gian thực hiện và số lượng phụ thuộc hàm phát hiện được của các thuật toán

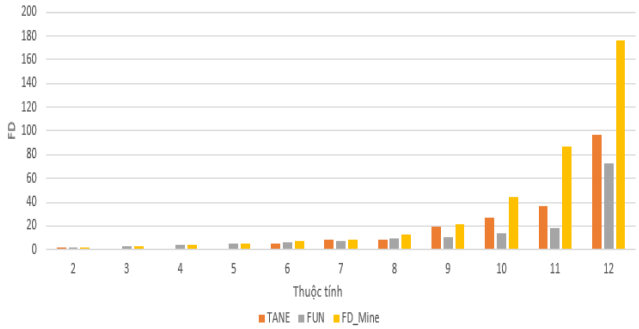
STT	Bộ dữ liệu	Số bản ghi	Số thuộc tính	Thời gian thực hiện (s)			Số lượng phụ thuộc hàm phát hiện được		
				TANE	FUN	FD_MINE	TANE	FUN	FD_MINE
1	iris	150	5	0	0	0	7	4	4
2	abalone	1000	9	969	168	237	32	49	212
3	glass	214	11	273	6	358	24	13	425
4	balance-scale	625	5	0	6	11	1	0	1
5	breast-cancer-wisconsin	699	11	6331	56	1035	1902	24	939
6	breast-cancer	286	10	45	5	9	2008	0	0
7	bridges	103	13	190	12	7792	2159	96	1720

Từ bảng trên ta thấy: với các bộ dữ liệu khác nhau thì các thuật toán xử lý với tốc độ khác nhau và cho ra kết quả số lượng phụ thuộc hàm khai phá được cũng khác nhau và không theo quy luật tuyến tính nào.

b) Biểu đồ so sánh trên cùng bộ dữ liệu với số lượng bản ghi không đổi và số lượng thuộc tính thay đổi:



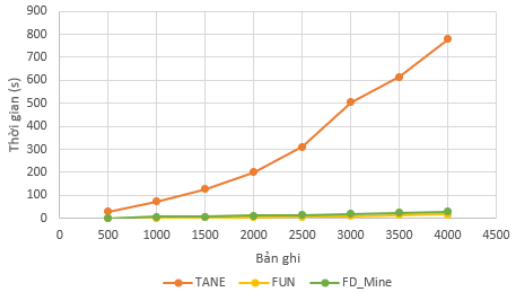
Hình 3. Biểu đồ so sánh thời gian thực hiện của các thuật toán



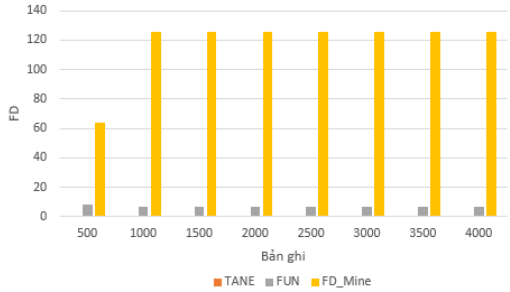
Hình 4. Biểu đồ so sánh số lượng phụ thuộc hàm phát hiện được của các thuật toán

Từ 2 biểu đồ ta thấy: khi cố định số lượng bản ghi và tăng số lượng thuộc tính thì thời gian thực hiện và số lượng phụ thuộc hàm phát hiện được cũng tăng theo tỷ lệ thuận. Thuật toán TANE luôn có thời gian xử lý lâu hơn các thuật toán khác và thuật toán FD_MINE luôn cho kết quả số lượng phụ thuộc hàm tìm được là nhiều hơn các thuật toán khác.

c) Biểu đồ so sánh trên cùng bộ dữ liệu với số lượng bản ghi thay đổi và số lượng thuộc tính không đổi:



Hình 5. Biểu đồ so sánh thời gian thực hiện của các thuật toán

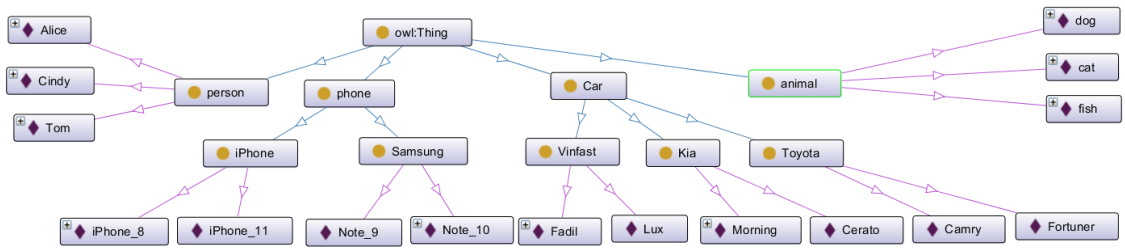


Hình 6. Biểu đồ so sánh số lượng phụ thuộc hàm phát hiện được của các thuật toán

Từ 2 biểu đồ ta thấy: khi tăng số lượng bản ghi và cố định số lượng thuộc tính thì thời gian thực hiện và số lượng phụ thuộc hàm phát hiện được cũng tăng theo tỷ lệ thuận. Thuật toán TANE luôn có thời gian xử lý lâu hơn các thuật toán khác và thuật toán FD_MINE luôn cho kết quả số lượng phụ thuộc hàm tìm được là nhiều hơn các thuật toán khác.

C. Cải tiến thuật toán FASTOFD

Thuật toán FASTOFD chỉ dựa vào các thể hiện tương đương của một lớp để xác định phụ thuộc hàm phả hệ tri thức tương đương *synonym OFD* và sự phân cấp cha con giữa các lớp trên cây phả hệ tri thức để xác định phụ thuộc hàm phả hệ tri thức kế thừa *inheritance OFD*. Tuy nhiên, trong các trường hợp các thể hiện không cùng một lớp hoặc các lớp không có mối quan hệ phân cấp cha con mặc dù các thể hiện này có chung một mối quan hệ thì thuật toán FASTOD không phát hiện được các phụ thuộc hàm có nghĩa. Xét cây phả hệ tri thức trong ví dụ sau:



Hình 7. Cây phả hệ tri thức

Trong cây tri phả hệ tri thức này, lớp Person có các mối quan hệ đến các lớp khác như:

- Person ‘has’ Car; Person ‘has’ Phone; Person ‘take care’ animal.

Và các thể hện có mối quan hệ như sau:

- Alice ‘has’ iPhone 8; Alice ‘take care’ fish.
- Cindy ‘has’ Fadil; Cindy ‘has’ iPhone 8; Cindy ‘take care’ cat; Cindy ‘take care’ dog.
- Tom ‘has’ Note 10; Tom ‘has’ Morning; Tom ‘take care’ cat; Tom ‘take care’ dog.

Gọi E là thể hiện của các lớp, relation là mối quan hệ giữa các thể hiện, ký hiệu *relations*(E, relation) là tập các thể hiện của lớp khác mà E có mối quan hệ đến: *relations*('Cindy', 'Fadil'} = {'has'}, *relations*('Cindy', 'Cat'} = {'take care'}, *relations*('Tom', 'Note 10')={'has'}.

Để phát hiện các phụ thuộc hàm có nghĩa, chúng tôi định nghĩa 1 phụ thuộc hàm phả hệ tri thức quan hệ (*relation OFD*) như sau:

Định nghĩa: Một quan hệ r thỏa mãn *relation OFD* $X \rightarrow_{rel} Y$ nếu mỗi thuộc tính $A \in Y$, mỗi $s \in \Pi_X(r)$ thì tồn tại một quan hệ sao cho $relation(s_1, A_1) \cap \ldots \cap relation(s_n, A_n) \neq \emptyset$:

$$\bigcap_{relations(A,\ a),a\in\{t[A]|t\in s\}} \neq \emptyset$$

Bảng 3. Bảng dữ liệu 2

	Person	**Using**	**Pet**
t_1	Tom	Morning	Cat
t_2	Alice	iPhone 8	Fish
t_3	Cindy	Fadil	Cat
t_4	Tom	Note 10	Dog
t_5	Cindy	iPhone 8	Dog

Với bảng dữ liệu 2, {t_1, t_2, t_3, t_4, t_5} là các bộ dữ liệu, thì kết quả khi sử dụng thuật toán tìm kiếm *relation OFD*, ta được: {Person} $\longrightarrow_{rel}$ {Using} vì $\Pi_{\{Person\}}$ ={{t_1, t_4}, {t_2}, {t_3, t_5}}, xét các lớp dữ liệu:

- {t_1, t_4}: *relations*('Tom', 'Morning') ∩ *relations*('Tom', 'Note 10') = {'has'}.
- {t_3, t_5}: *relations*('Alice', 'Fadil') ∩ *relations*('Alice', 'iPhone 8') = {'has'}.
- {t_2}: chứa tập giá trị đơn nên không bị xung đột.

Và {Person} $\longrightarrow_{rel}$ {Pet} vì $\Pi_{\{Person\}}$ ={{t_1, t_4}, {t_2}, {t_3, t_5}}, xét các lớp dữ liệu:

- {t_1, t_4}: *relations*('Tom', 'Cat') ∩ *relations*('Tom', 'Dog') = {'take care'}.
- {t_3, t_5}: *relations*('Cindy, 'Cat') ∩ *relations*('Cindy', 'Dog') = {'take care'}.
- {t_2}: chứa tập giá trị đơn nên không bị xung đột.

D. So sánh thuật toán phát hiện phụ thuộc hàm phả hệ tri thức FASTOFD với TANE, FUN, FD_MINE

Bảng so sánh kết quả khi chạy các thuật toán trên với bộ dữ liệu ở bảng dữ liệu 1.

Bảng 4. Bảng so sánh kết quả

STT	**Thuật toán**	**Kết quả**
1	FASTOFD	*Synonym OFDs*: {Country Code} → {Country} {Country Code,Symptom} → {Country} {Country Code,Diagnosis} → {Country} {Country Code,Symptom,Diagnosis} → {Country} *Inheritance OFDs*: {Symptom} → {Medicine} {Diagnosis} → {Medicine} {Country Code,Symptom} → {Medicine} {Country Code,Diagnosis} → {Medicine} {Symptom,Diagnosis} → {Medicine} {Country Code,Symptom,Diagnosis} → {Medicine}
2	TANE	{Medicine} → {Country Code} {Medicine} → {Country} {Country,Diagnosis} → {Country Code} {Country,Diagnosis} → {Symptom} {Country,Symptom} → {Country Code} {Country,Symptom} → {Diagnosis}
3	FUN	{Country} → {Country Code} {Symptom} → {Diagnosis} {Diagnosis} → {Symptom} {Medicine} → {Country Code,Country,Symptom,Diagnosis} {Country,Symptom} → {Medicine} {Country,Diagnosis} → {Medicine}
4	FD_MINE	{Country} → {Country Code} {Symptom} → {Diagnosis} {Diagnosis} → {Symptom} {Medicine} → {Country Code,Country,Symptom,Diagnosis} {Country,Symptom} → {Country Code}

Thuật toán FASTOFD tìm ra các phụ thuộc hàm tri thức là {Country code} $\rightarrow_{syn}$ {Country} và {Sumpton, Diagnosis} $\rightarrow_{inh}$ {Medicine} điều mà các thuật toán còn lại không thể tìm ra.

IV. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

A. Kết luận

Bài báo đã đánh giá các thuật toán phát hiện phụ thuộc hàm với nhiều tiêu chí khác nhau: thời gian, số lượng phụ thuộc hàm phát hiện được, số lượng thuộc tính và số lượng bản ghi nhằm có thể so sánh tính năng giữa các thuật toán.

Trong thuật toán phát hiện phụ thuộc hàm phả hệ tri thức FASTOD, bài báo đã chỉ ra một khiếm khuyết của thuật toán. Hiệu chỉnh nhược điểm này có thể phát hiện thêm phụ thuộc hàm có nghĩa.

B. Hướng phát triển

Với những kết quả đạt được, trong thời gian tới, chúng tôi tiếp tục khảo sát và cải tiến thuật toán phát hiện phụ thuộc hảm phả hệ tri thức với mối quan hệ dựa trên các mức khác nhau nhằm khai thác hết ngữ nghĩa của dữ liệu.

TÀI LIỆU THAM KHẢO

[1] Yka Huhtala, Juha Karkkainen, Pasi Porkka and Hannu Toivonen, "TANE: An effcient algorithm for discovering functional and approximate dependencies", 1999.

[2] Novelli and R. Cicchetti, "FUN: An efficient algorithm for mining functional and embedded dependencies", 2001.

[3] Hong Yao, Howard J. Hamilton and Cory J. Butz, "FD_Mine: Discovering functional dependencies in a database using equivalences", 2002.

[4] Jean-Paul Calbimonte, Fabio Porto and C. Maria Kee, "Functional dependencies in OWL ABoxes", 2009.

[5] Sridevi Baskaran, Alexander Keller, Fei Chiang, Lukasz Golab and Jaroslaw Szlichta, "Efficient discovery of ontology functional dependencies", 2017.

[6] Alexander Keller, "Data curation with ontology functional dependences", 2017.

[7] Sridevi Baskaran, "Discovering ontology functional dependencies", 2016.

[8] Alexander Keller and Jaroslaw Szlichta, "Ontology functional dependencies", 2016.

EVALUATE THE PERFORMANCE AND EFFECTIVENESS OF ALGORITHMS FOR DISCOVERING FUNCTIONAL DEPENDENCIES: TANE, FUN, FD_MINE AND ONTOLOGY FUNCTIONAL DEPENDENCIES

Trinh Minh Quoc Trung, Nguyen Duc Thuan

ABSTRACT: *Discovering functional dependencies is a fundamental step in the design and building of relational databases. The more fully functional dependency set the more efficient it is to reduce, decompose and normalize the database. There are many algorithms to discover classical functional dependencies as TANE, FUN, FD_MINE,... Nowadays, the trend of organizing data to extract information semantically is getting more and more attention. The biggest difference between a relational database and a ontology database with a semantic constraint is the attribute domain relationship.*

The expansion of the concept of classical functional dependence to ontology dependence of knowledge is an inevitable requirement. In this paper, we evaluate and retest the classical function dependency detection algorithms, investigate the knowledge function dependency detection algorithms to overview the functional dependency algorithms.

Keywords: *Functional dependency, Discovering functional dependencies algorithms, Ontology tree, ontology functional dependencies.*