Thuật Toán Khai Thác Nhanh Tập Sinh của Tập Phổ Biến Đóng từ Dữ Liệu Giao Dịch

Phan Thành Huấn[1,2][✉], Nguyễn Như Đồng[3], Phan Hoàng Chương[4]

[1] Bộ môn Tin học, Đại học Khoa học Xã hội và Nhân văn, Đại học Quốc gia, Tp. Hồ Chí Minh
[2] Khoa Toán -Tin học, Đại học Khoa học Tự nhiên, Đại học Quốc gia, Tp. Hồ Chí Minh
huanphan@hcmussh.edu.vn
[3] Phòng Đào tạo, Cao đẳng Kỹ nghệ II, Tp. Hồ Chí Minh
dongnguyennhu@hvct.edu.vn
[4] Khoa Khoa học Cơ bản, Đại học Công nghệ Thông tin, Đại học Quốc gia, Tp. Hồ Chí Minh
chuongph@uit.edu.vn

Tóm tắt. Trong khai thác dữ liệu, khai thác luật kết hợp là một trong những kỹ thuật quan trọng và được nghiên cứu nhiều. Trong thực tế, khi khai thác luật kết hợp thì số lượng luật sinh ra là rất lớn và chứa nhiều luật dư thừa. Một số tác giả đã đề xuất khai thác luật kết hợp không dư thừa từ tập sinh của tập phổ biến đóng và rút trích tập sinh dựa vào thuật toán Apriori, CHARM, FP-Growth. Các thuật toán này có độ phức tạp tính toán cao và không hiệu quả khi ngưỡng phổ biến nhỏ. Trong bài viết này, chúng tôi đề xuất thuật toán mới NOV-GCFI khai thác nhanh tập sinh của tập phổ biến đóng. Kết quả thực nghiệm trên bộ dữ liệu thực và giả lập, cho thấy thuật toán đề xuất hiệu quả.

Từ khóa: Tập phổ biến đóng, Tập sinh của tập phổ biến đóng, NOV-GCFI.

1 Giới Thiệu

Năm 1993, R.Agrawal đã đề xuất mô hình cơ bản khai thác luật kết hợp từ dữ liệu giao dịch dạng nhị phân theo hai giai đoạn [1]: sinh *tập phổ biến* (*Frequent Itemset* - **FI**) và sinh *luật tin cậy* kết hợp từ tập phổ biến. Khi đó, số lượng tập mục phổ biến được sinh ra rất lớn và chiếm rất nhiều thời gian. Một số tác giả trên thế giới đã đề xuất thuật toán sinh *tập phổ biến đóng* (*Closed Frequent Itemset* - **CFI**) [2-3], đây là tập không dư thừa và có kích thước nhỏ hơn **FI** rất nhiều lần (**CFI** $\subseteq$ **FI**). Tuy nhiên, để sinh các *luật kết hợp chính xác* cũng như *không dư thừa*, một số tác giả đã đề xuất khai thác luật kết hợp từ tập sinh của tập phổ biến đóng (*Minimal*/*Generator Closed Frequent Itemset* – **mGCFI**/**GCFI**) [4-8] như thuật toán **GrGrowth** [4], **Zart** [6] và **DefMe** [7-8]. Các thuật toán trên tựa **Apriori**, tìm kiếm theo chiều sâu và sử dụng cấu trúc **FP-Tree**, **IT-Tree** – các thuật toán này tốn nhiều bộ nhớ và đọc dữ liệu nhiều lần; cũng như không hiệu quả trên các loại dữ liệu có mật độ dày hoặc thưa.

Trong bài viết này, chúng tôi đề xuất thuật toán mới khai thác nhanh *tập sinh* của *tập phổ biến đóng* được gọi là thuật toán **NOV-GCFI**. Thuật toán đề xuất thực hiện hiệu quả trên các loại dữ liệu và sử dụng lại khi thay đổi ngưỡng *minsup*. Thuật toán đề xuất bao gồm các thuật toán liên quan:

- Xây dựng mảng **Index_COOC** chứa *items* đồng xuất hiện và *items* xuất hiện ít nhất trong một giao dịch của từng *item* hạt nhân;
- Dựa trên **Index_COOC** xây dựng cây **nLOOC-Tree**;
- Thuật toán **NOV-GCFI** khai thác nhanh tập sinh dựa trên cây **nLOOC-Tree.**

Trong phần 2, trình bày các khái niệm cơ bản về khai thác tập phổ biến, tập sinh của tập phổ biến đóng. Phần 3, trình bày thuật toán đề xuất liên quan và thuật toán **NOV-GCFI** khai thác *tập sinh* của tập phổ biến đóng. Kết quả thực nghiệm được trình bày trong phần 4 và kết luận ở phần 5.

2 Khai thác tập phổ biến

Cho $I = \{i_1, i_2, \dots i_m\}$ là tập gồm *m* mục hàng, mỗi mục hàng gọi là *item*. Tập các item $X = \{i_1, i_2, \dots i_k\}, \forall i_j \in I (1 \le j \le k)$ gọi là *itemset*, itemset có *k item* gọi là *k-itemset*. Ɗ là dữ liệu giao dịch, gồm *n* bản ghi gọi là tập các giao dịch $T = \{t_1, t_2, \dots t_n\}$, mỗi giao dịch $t_j = \{i_{k_1}, i_{k_2}, \dots i_{k_l}\}, \forall i_{k_l} \in I (1 \le k_l \le m)$.

Định nghĩa 1: Độ phổ biến (*support*) của *itemset* $X \subseteq I$, ký hiệu $sup(X)$ là số giao dịch trong *Ɗ* có chứa *X*.

Định nghĩa 2: Cho $X \subseteq I$, X gọi là *itemset* phổ biến nếu $sup(X) \ge minsup$, trong đó *minsup* là ngưỡng phổ biến tối thiểu. Ký hiệu tập chứa các itemset phổ biến là **FI**.

Cho dữ liệu giao dịch Ɗ trong Bảng 1.

Bảng 1. Dữ liệu giao dịch *Ɗ* cho Ví dụ.

Mã	Tập mục hàng							
t1	A	C		E	F			
t2	A	C				G		
t3				E			H	
t4	A	C	D		F	G		
t5	A	C		E		G		
t6							E	
t7	A	B	C				E	
t8	A		C	D				
t9	A	B	C				E	G
10	A		C				E	F G

Dữ liệu giao dịch *Ɗ* trong *Bảng 1*, có 8 *item I = {A, B, C, D, E, F, G, H}* và 10 giao dịch T = *{t1, t2, t3, t4, t5, t6, t7, t8, t9, t10}*.

Định nghĩa 3: Cho *X* ∈ **FI**, *X* được gọi là *itemset* phổ biến đóng nếu *X* không có tập cha cùng độ phổ biến. Ký hiệu tập chứa các *itemset* phổ biến đóng là **CFI**.

Định nghĩa 4: Cho *X* ∈ **CFI**, tất cả các *itemset* con thực sự của *X* có *cùng độ phổ biến* với *X* gọi là *itemset* sinh của *X*. Ký hiệu tập chứa các *itemset* sinh là **GCFI**.

Bảng 2. Tập CFI và GCFI trên dữ liệu giao dịch *Ɗ* với *minsup* = 3

k-itemset	Tập CFI (#FCI = 6)	Tập GCFI (#GCFI = 13)
1	E	F, G, A, C
2	AC	FA, FC, GA, GC, GE, EA, EC
3	FAC, GAC, EAC	GEA, GEC
4	GEAC	

Bảng 2, cho thấy tập **CFI** và **GCFI** chứa *k-itemset* với *minsup* = 3. Số lượng |**CFI**| = 6 và |**GCFI**| = 13. Tỷ suất |CFI|/|GCFI| = 6/13×100% ≈ 46%.

3 Thuật toán đề xuất

3.1 Tập chiếu và item xuất hiện ít nhất trên cùng một giao dịch

Tập chiếu của item i_k trên dữ liệu giao dịch D: $\pi(i_k) = \{t \in \mathcal{D} | i_k \in t\}$ là tập các giao dịch có chứa item i_k.

$$sup(i_k) = |\pi(i_k)| \quad (1)$$

Tập chiếu của tập $X = \{i_1, i_2, \dots i_k\}, \forall i_{j=\overline{1,k}} \in I, \pi(X) = \pi(i_1) \cap \pi(i_2) \dots \cap \pi(i_k)$.

$$sup(X) = |\pi(X)| \quad (2)$$

Để tránh trùng lặp *không gian sinh*, chúng tôi đưa ra Định nghĩa 5 và 6:

Định nghĩa 5: Cho $i_k \in I$ ($i_1 \prec i_2 \prec \dots \prec i_m$) thứ tự theo độ phổ biến, ta gọi i_k là *item hạt nhân.* Tập $X_{lexcooc} \subseteq I$ gọi *đồng xuất hiện có thứ tự* với *item* i_k: $X_{lexcooc}$ là tập các *item* có thứ tự theo độ phổ biến, đồng xuất hiện cùng i_k và $\pi(i_k) \equiv \pi(i_k \cup x_{lexcooc})$, $\forall x_{lexcooc} \in \mathcal{P}_{\geq 1}(X_{lexcooc}), \forall i_j \in X_{lexcooc}, i_k \prec i_j$. Ký hiệu, $lexcooc(i_k) = X_{lexcooc}$.

Định nghĩa 6: Cho $i_k \in I$ ($i_1 \prec i_2 \prec \dots \prec i_m$) thứ tự theo độ phổ biến, ta gọi i_k là *item hạt nhân*. Tập $X_{lexlooc} \subseteq I$ chứa các *item* xuất hiện có thứ tự cùng với i_k ít nhất trong một giao dịch, nhưng không đồng xuất hiện: $1 \leq |\pi(i_k \cup x_{lexlooc})| < |\pi(i_k)|$, $\forall x_{lexlooc} \in \mathcal{P}_{\geq 1}(X_{lexlooc}), \forall i_j \in X_{lexlooc}, i_k \prec i_j$. Ký hiệu, $lexlooc(i_k) = X_{lexlooc}$.

3.2 Thuật toán sinh item xuất hiện ít nhất trên cùng một giao dịch có thứ tự

Trong phần này, chúng tôi trình bày thuật toán sinh các *item* đồng xuất hiện và xuất hiện ít nhất trong một giao dịch với *item hạt nhân*, được lưu trữ vào mảng **Index_COOC**. Mỗi phần tử trong **Index_COOC** gồm có 4 thành phần thông tin:

- **Index_COOC[k].***item*: *item hạt nhân thứ k*;
- **Index_COOC[k].***sup*: *độ phổ biến* của *item hạt nhân thứ k*;
- **Index_COOC[k].***cooc*: các *item* đồng xuất hiện cùng *item hạt nhân thứ k*;
- **Index_COOC[k].***looc*: các *item* xuất hiện cùng *item hạt nhân thứ k ít nhất trong một giao dịch*;

Mã giả thuật toán 1. Xây dựng bảng Index_COOC

Đầu vào*:* Dữ liệu giao dịch *D*
Đầu ra*:* Mảng **Index_COOC**, ma trận dataset **BiM**//*ma trận bit*

```
1.  Với mỗi phần tử k của mảng Index_COOC thực hiện:
2.      Index_COOC[k].item = i_k;        Index_COOC[k].sup = 0
3.      Index_COOC[k].cooc= 2^m - 1;     Index_COOC[k].looc= 0
4.  Với mỗi giao dịch t_i thực hiện:
5.      Lưu giao dịch t_i vào ma trận BiM
6.      Với mỗi item k có trong giao dịch t_i thực hiện:
7.          Index_COOC[k].cooc = Index_COOC[k].cooc AND vectorbit(t_i)
8.          Index_COOC[k].looc = Index_COOC[k].looc OR vectorbit(t_i)
9.          Index_COOC[k].sup = Index_COOC[k].sup + 1
10. Sắp xếp mảng Index_COOC tăng dần theo sup
11. Với mỗi phần tử k của mảng Index_COOC: //định nghĩa 5,6
12.     Index_COOC[k].cooc= lexcooc(i_k)
13.     Index_COOC[k].looc= lexlooc(i_k)
```

Minh họa thuật toán 1: thực hiện từ *dòng 1* đến *9*

Khởi tạo mảng **Index_COOC**: (*looc biểu diễn dạng bit*) số *item* là m = 8

item	A	B	C	D	E	F	G	H
sup	0	0	0	0	0	0	0	0
cooc	11111111	11111111	11111111	11111111	11111111	11111111	11111111	11111111
looc	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

Đọc giao dịch t_1: {A, C, E, F} có biểu diễn dạng bit là **10101100**

item	A	B	C	D	E	F	G	H
sup	1	0	1	0	1	1	0	0
cooc	**10101100**	11111111	**10101100**	11111111	**10101100**	**10101100**	11111111	11111111
looc	10101100	00000000	10101100	00000000	10101100	10101100	00000000	00000000

Tương tự, đọc giao dịch t_{10}: {A, C, E, F, G} có biểu diễn dạng bit là **10101110**

item	A	B	C	D	E	F	G	H
sup	8	2	8	2	7	3	5	1
cooc	**10100000**	11101000	**10100000**	10110000	**00001000**	**10100100**	**10100010**	00001001
looc	11111110	11101010	11111110	10110110	11101111	10111110	11111110	00001001

Dòng *10*, sắp xếp theo độ phổ biến của item, ta có **Index_COOC** như sau:

Item	H	B	D	F	G	E	A	C
sup	1	2	2	3	5	7	8	8
cooc	E	A, C, E	A, C	A, C	A, C	∅	C	A
looc	∅	G	F, G	D, E, G	B, D, E, F	A,B,C,F,G,H	B,D,E,F,G	B,D,E,F,G

Thực hiện rút gọn ở *dòng 11*, *12* và *13*, ta có bảng 3:

Bảng 3. Mảng Index_COOC có thứ tự tăng theo độ phổ biến của item, *cooc* và *looc* có thứ tự.

Item	H	B	D	F	G	E	A	C
sup	1	2	2	3	5	7	8	8
cooc	E	A, C, E	A, C	A, C	A, C	∅	C	∅
looc	∅	G	F, G	G, E	E	A, C	∅	∅

3.3 Thuật toán sinh cây nLOOC-Tree [9]

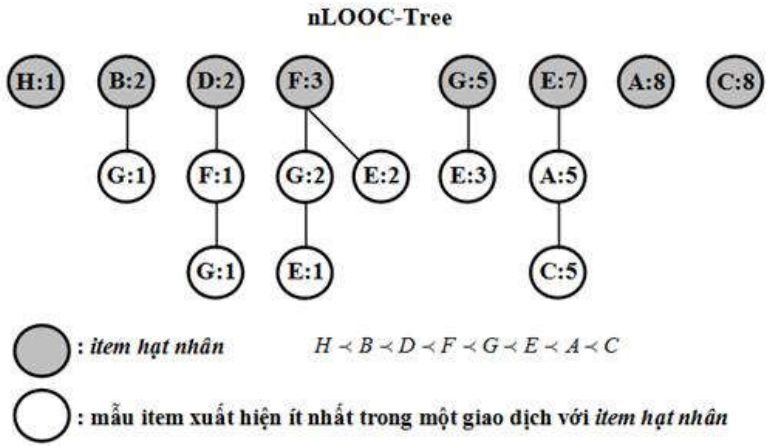
Từ **Index_COOC** xây dựng các cây chứa các mẫu *xuất hiện ít nhất* trong một giao dịch với *item* hạt nhân. Mỗi cây có nút gốc là *item hạt nhân* và các nút con là *items xuất hiện ít nhất* trong một giao dịch với *item hạt nhân*. Mỗi nút có 2 thành phần:

- **nLOOC_Tree[k].***item*: *item xuất hiện ít nhất với item hạt nhân (nút gốc)*;
- **nLOOC_Tree[k].***sup*: *độ phổ biến* của *item xuất hiện cùng với item hạt nhân*;

Mã giả thuật toán 2. Xây dựng danh sách cây nLOOC-Tree

Đầu vào*:* Mảng **Index_COOC**
Đầu ra*:* Danh sách cây **nLOOC-Tree**

1. *Với mỗi* phần tử *k* của mảng **Index_COOC**:
2. nLOOC_Tree[k].*item* = i_k ; nLOOC_Tree[k].*sup* = *sup*(i_k)
3. *Với mỗi* item $i_k \in$ giao dịch t_ℓ :
4. *Với mỗi* item $i_j \in$ **Index_COOC**[k].*looc*:
5. *Nếu* $i_j \notin$ nút con của nút gốc **nLOOC-Tree**[k] *thì*
6. Thêm nút con i_j vào nút gốc **nLOOC-Tree**[k]
7. *Ngược lại*
8. Thêm nút con i_j vào nút gốc **nLOOC-Tree**[k]
9. Cập nhật *sup* của nút con i_j của nút gốc **nLOOC-Tree**[k]



Hình 1. Cây nLOOC-Tree theo mảng Index_COOC ở Bảng 3

Cây **nLOOC-Tree** có các đặc trưng như sau:

- Chiều cao của cây *nhỏ hơn hoặc bằng* số lượng các *item xuất hiện ít nhất* trong một giao dịch với *item hạt nhân* (*các item có thứ tự theo độ phổ biến*).
- Mỗi đường đi đơn (*single-path*) là một mẫu có thứ tự từ nút gốc (*item hạt nhân*) đến nút lá và độ phổ biến của một mẫu là độ phổ biến của nút lá ($i_k \rightarrow i_{k+1} \rightarrow \ldots \rightarrow i_\ell$).
- Mỗi phân đoạn đường đi đơn (*sub-single-path*) từ nút gốc đến nút con bất kỳ trong đường đi đơn là một mẫu thứ tự và độ phổ biến của mẫu là độ phổ biến của nút con ở cuối của phân đoạn đường đi đơn.

Ví dụ 1: Xét *item hạt nhân* F, có đường đi đơn {**F** → G → E} và *sup*(**F**GE) = 1; phân đoạn đường đi đơn {**F** → G} và *sup*(**F**G) = 2 là độ phổ biến của nút con G.

3.4 Thuật toán NOV-GCFI

Trong phần này, chúng tôi trình bày các *bổ đề* và *tính chất* để phát sinh cũng như rút gọn không gian sinh **GCFI** và thuật toán khai thác nhanh **GCFI** dựa trên mảng **Index_COOC** và cây **nLOOC_Tree** (ký hiệu: $\mathcal{P}_k(X)$ – *powerset của X có k item*).

Bổ đề 1: $\forall\ i_k \prec i_j$, nếu $i_j \in$ *lexlooc*(i_k) thì *sup*($i_k \cup i_j$) < *sup*(i_k).

Chứng minh: *sup*($i_k \cup i_j$) < *sup*(i_k), theo (1) và (2) $\pi(i_k \cup i_j) = \pi(i_k) \cap \pi(i_j) \subset \pi(i_k)$■.

Bổ đề 2: $\forall\ i_k \in I$, *sup*(i_k) = *minsup*, nếu *lexlooc*(i_k) = $X_{lexlooc}$ và *sup*($i_k \cup x_{sub}$) < *minsup* thì $\{i_k \cup x_{sub}\} \notin$ **GCFI**, $\forall\ x_{sub} \in \mathcal{P}_{\geq 1}(X_{lexlooc})$.

Chứng minh: theo bổ đề 1, ta có *sup*($i_k \cup x_{sub}$) < *sup*(i_k) = *minsup*, nên $\{i_k \cup x_{sub}\} \notin$ **GCFI**, $\forall\ x_{sub} \in \mathcal{P}_{\geq 1}(X_{lexlooc})$■.

Bổ đề 3: $\forall\ i_k \in I$, *sup*(i_k) ≥ *minsup* và *lexcooc*(i_k) = $X_{lexcooc}$ thì $\{i_k \cup x_{sub}\} \in$ **GCFI**, $\forall\ x_{sub} \in \mathcal{P}_{<\lambda}(X_{lexcooc})$, $\lambda = |X_{lexcooc}|$.

Chứng minh: theo bổ đề 1, *sup*(Y) < *sup*($\{i_k \cup \mathcal{P}_{=\lambda}(X_{lexcooc})\}$), $\forall\ Y \supset \{i_k \cup \mathcal{P}_{=\lambda}(X_{lexcooc})\}$. Khi đó, $\{i_k \cup \mathcal{P}_{=\lambda}(X_{lexcooc})\} \in$ **CFI** và (2) thì *sup*($\{i_k \cup \mathcal{P}_{=\lambda}(X_{lexcooc})\}$) = *sup*($\{i_k \cup x_{sub}\}$), $\forall\ x_{sub} \in \mathcal{P}_{<\lambda}(X_{lexcooc})$■.

Tính chất 1: $\forall i_k \in I$, nếu *sup*(i_k) ≥ *minsup* và *lexcooc*(i_k) = {∅} thì $i_k \notin$ **GCFI**.

Tính chất 2: (*tập sinh từ các item đồng xuất hiện hoàn toàn*) $\forall\ i_{k-1} \prec i_k \in I$, nếu *sup*($i_{k-1}$) = *sup*($i_k$) và $i_k \in$ *lexcooc*(i_{k-1}) thì itemsets sinh từ *item hạt nhân* i_{k-1}, $\forall \{i_{k-1} \cup X\} \in$ **GCFI** được sinh thêm bằng cách thay bởi *item hạt nhân* i_k, $\forall \{i_k \cup X\} \in$ **GCFI**.

Thuật toán khai thác *tập sinh của tập phổ biến đóng* **GCFI** từ cây **nLOOC-Tree**:

Mã giả thuật toán 3. NOV-GCFI - Khai thác tập sinh của tập phổ biến đóng GCFI

Đầu vào: Dataset, **Index_COOC** và *minsup*
Đầu ra: Tập sinh của tập phổ biến đóng - **GCFI**

1. Với mỗi **Index_COOC**[*k*].*sup* ≥ *minsup*
2. Nếu (**Index_COOC**[*k*].*cooc* ≠{∅})*//tính chất 1*
3. **GCFI** [k] = ∪ {**Index_COOC**[*k*].*item* ∪ $\mathcal{P}_{<\lambda}$ (**Index_COOC**[*k*].*cooc*)}
4. Nếu (**Index_COOC**[*k*].*sup* > *minsup*)∧(**Index_COOC**[*k*].*looc* ≠{∅})*//bổ đề 2*
5. **nLOOC_Tree**(**Index_COOC**[*k*].*item*)
6. ***SSP*** ← GenPath(**Index_COOC**[*k*].*item*)*//sinh sub-single-path*
7. Với mỗi $ssp_j \in$ ***SSP***
8. Nếu ($sup(ssp_j) \geq minsup$) ∧ ($sup(ssp_j \setminus \{i_\ell\}) \geq minsup$) thì
9. **GCFI [***k***]** = ∪{($ssp_j \setminus \{i_\ell\}$)}
10. **GCFI [***k***]** = ∪{($ssp_j \setminus \{i_\ell\}$)∪ $\mathcal{P}_{<\lambda}$ (**Index_COOC**[*k*].*cooc*)}*//bổ đề 3*
11. Nếu (**Index_COOC**[*k-1*].*sup*=**Index_COOC**[*k*].*sup*)∧
(**Index_COOC**[*k*].*item* ∈ **Index_COOC**[*k*].*cooc*) thì
12. **GCFI**[k]=Replace(**GCFI**[k-1], **Index_COOC**[*k*].*item*)*//tính chất 2*

Ví dụ 2: Cho dữ liệu giao dịch Ɗ trong Bảng 1 và *minsup* = 3. Sau khi thực hiện thuật toán 1, ta có mảng **Index_COOC** chứa các *item* đồng xuất hiện và xuất hiện ít nhất trong một giao dịch với *item* hạt nhân, như trong Bảng 3.

Dòng 1 và 2: các item F, G, E, A, C là *item sinh* theo **tính chất 1**;

Xây dựng các cây **nLOOC-Tree** cho các *item* cần khai phá: F, G, E, A và C;

Xét *item* F, có *lexcooc*(F) = {A, C} sinh tập **GCFI$_{[F]}$** = {(**F**, 3), (**F**A, 3), (**F**C, 3)} (*dòng 3*) và *sup*(F) = *minsup*, nên không sinh **GCFI** từ **nLOOC-Tree**(F) (*dòng 4*).

Xét *item* G, có *lexcooc*(G) = {A, C} sinh tập **GCFI$_{[G]}$** = {(**G**, 5), (**G**A, 5), (**G**C, 5)} (*dòng 3*) xem cây **nLOOC-Tree**(G): có một đường đi đơn {**G** → E} và *sup*(**G**E) = 3 ≥ *minsup*. Ta có, **GCFI$_{[G]}$** = ∪{(**G**E, 3), (**G**EA, 3), (**G**EC, 3)} (*dòng 4 đến 10*).

Xét *item* E, có *lexcooc*(E) = {∅} nên **GCFI$_{[E]}$** = {∅}, xem cây **nLOOC-Tree**(E): có một đường đi đơn {**E** → A → C} và *sup*(**E**AC) = 5 ≥ *minsup*. Ta có, **GCFI$_{[E]}$** = {(**E**A, 5), (**E**C, 5)}.

Xét *item* A, có *lexcooc*(A) = {C} sinh tập **GCFI$_{[A]}$** = {(**A**, 8)} (*dòng 3*) và *lexlooc*(A) = {∅}, nên không sinh **GCFI** từ **nLOOC-Tree**(A) (*dòng 4*).

Xét *item* C, đồng xuất hiện hoàn toàn với *item* A: **GCFI$_{[C]}$** = {(**C**, 8)} (*dòng 11*).

Tập sinh **GCFI** trên dữ liệu giao dịch Ɗ ở Bảng 1 với *minsup* = 3:

Item hạt nhân	Tập sinh GCFI					
F	(**F**, 3)	(**F**A, 3)	(**F**C, 3)			
G	(**G**, 5)	(**G**A, 5)	(**G**C, 5)	(**G**E, 3)	(**G**EA,3)	(**G**EC,3)
E	(**E**A, 5)	(**E**C, 5)				
A	(**A**, 8)					
C	(**C**, 8)					

4 Kết Quả Thực Nghiệm

Thực nghiệm trên máy tính CF-74, Core Duo 2.0 GHz, 4GB RAM, thuật toán cài đặt trên C#, Microsoft Visual Studio 2010.

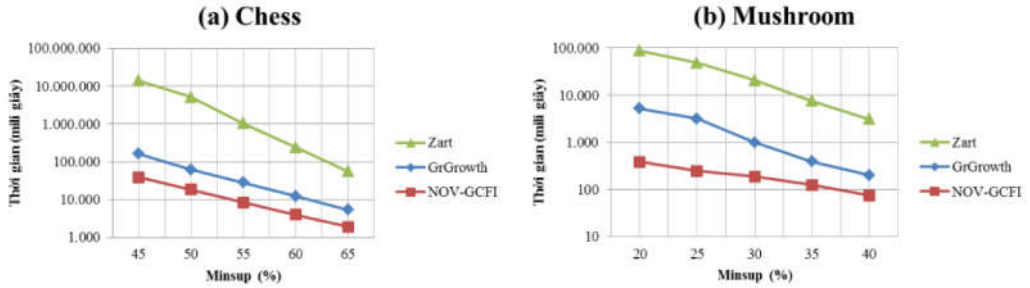
Nghiên cứu thực nghiệm trên hai nhóm dữ liệu:

- Nhóm dữ liệu thực có *mật độ dày*: 2 tập dữ liệu **Chess** và **Mushroom** [http://archive.ics.uci.edu/ml].
- Nhóm dữ liệu giả lập có *mật độ thưa*: 2 tập dữ liệu giả lập **T10I4D100K** và **T40I10D100K** [http://www.almaden.ibm.com].

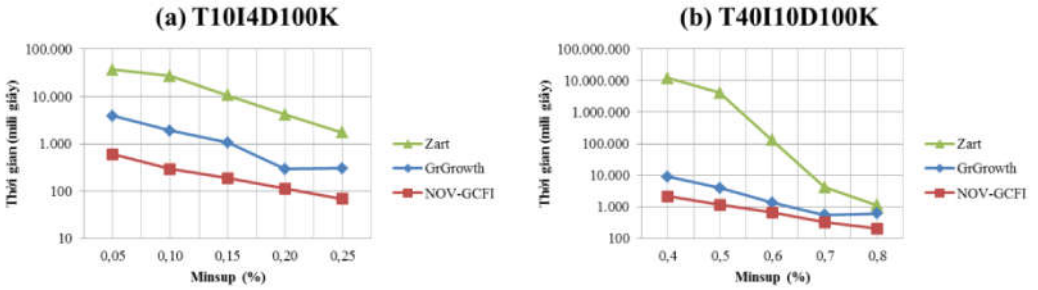
Bảng 4. Dữ liệu thực nghiệm

Dữ liệu	Số giao dịch	Số item	Số item trung bình/giao dịch	Mật độ (%)
Chess	3.196	75	37	49,3
Mushroom	8.142	119	23	19,3
T10I4D100K	100.000	870	10	1,1
T40I10D100K	100.000	942	40	4,2

Trong phần thực nghiệm, chúng tôi sử dụng 4 tập dữ liệu được mô tả ở Bảng 4 và so sánh thuật toán đề xuất **NOV-GCFI** với thuật toán **GrGrowth** [4] và thuật toán **Zart** [6]. Để so sánh thời gian thực hiện giữa thuật toán đề xuất với thuật toán **GrGrowth** và **Zart**, chúng tôi chỉ tiến hành so sánh theo từng ngưỡng *minsup* và cả 3 thuật toán đều cho cùng kết quả *số lượng tập sinh* của tập phổ biến đóng.



Hình 2. Thời gian thực hiện của 3 thuật toán trên Chess, Mushroom.



Hình 3. Thời gian thực hiện của 3 thuật toán trên T10I4D100K, T40I10D100K.

Hình 2a là kết quả thực nghiệm trên tập dữ liệu **Chess** (49,3%), ta thấy thuật toán đề xuất **NOV-GCFI** nhanh hơn cả 2 thuật toán **GrGrowth, Zart**.

Tương tự, hình 2b là kết quả thực nghiệm trên tập **Mushroom** (19,3%), ta thấy thuật toán đề xuất **NOV-GCFI** nhanh hơn cả 2 thuật toán **GrGrowth, Zart**.

Hình 3a là kết quả thực nghiệm trên tập dữ liệu **T10I4D100K** có mật độ thưa (1,1%), ta thấy thuật toán đề xuất **NOV-GCFI** nhanh hơn cả 2 thuật toán **GrGrowth, Zart**. Ngoài ra, chúng ta cũng thấy **GrGrowth** không tốt trên dữ liệu mật độ thưa.

Hình 3b là kết quả thực nghiệm trên tập dữ liệu **T40I10D100K** có mật độ thưa (4,2%), ta thấy thuật toán đề xuất **NOV-GCFI** nhanh hơn cả 2 thuật toán **GrGrowth, Zart**. Ngoài ra, chúng ta cũng thấy **GrGrowth** không tốt trên dữ liệu mật độ thưa.

Kết quả trên cho thấy thuật toán đề xuất **NOV-GCFI** khai thác *tập sinh* của *tập phổ biến đóng* tốt hơn và ổn định so với thuật toán **GrGrowth, Zart** trên cả 2 loại dữ liệu dày và thưa. Ngoài ra, thuật toán cũng cần thực nghiệm thêm trên nhiều tập dữ liệu có mật độ khác nhau, cũng như trên nhiều dữ liệu cỡ lớn.

5 Kết Luận

Nhóm tác giả đã đề xuất kiến trúc khai thác *tập sinh của tập phổ biến đóng* gồm ba giai đoạn: *giai đoạn thứ nhất* là tính nhanh mảng **Index_COOC** chứa các *item* xuất hiện với *item hạt nhân* ít nhất trong một giao dịch; *giai đoạn thứ hai*: xây dựng cây **nLOOC-Tree** dựa vào mảng **Index_COOC**; *giai đoạn thứ ba*: đề xuất thuật toán **NOV-GCFI** khai thác nhanh *tập sinh của tập phổ biến đóng* dựa trên cây **nLOOC-Tree**. Với kiến trúc như trên, khi người dùng khai thác *tập sinh của tập phổ biến đóng* với *ngưỡng khác* thì thuật toán đề xuất chỉ thực hiện khai thác *tập sinh* trên mảng **Index_COOC** đã tính ở lần khai thác trước làm giảm thời gian xử lý đáng kể.

Tương lai, nhóm tác giả mở rộng thuật toán để có thể khai thác luật kết hợp không dư thừa dựa vào *tập sinh của tập phổ biến đóng* trên dữ liệu giao dịch có *trọng số* của *item*, đây là hướng nghiên cứu đang được quan tâm với nhiều khả năng ứng dụng.

Tài liệu tham khảo

1. Agrawal R., Imilienski T., Swami A.: Mining association rules between sets of large databases. Proc. of the ACM SIGMOD Int Conf on Management of Data, 207-216 (1993)
2. Bastide Y., Pasquier N., Taouil R., Stumme G., Lakhal L.: Mining Minimal Non-Redundant Association Rules using Closed Frequent Itemssets. In: 1st International Conference on Computational Logic, 972 – 986 (2000)
3. Zaki M. J.: Mining Non-Redundant Association Rules. Data Mining and Knowledge Discovery, 9, 223–248 (2004)
4. Liu G., Li J., Wong L.: A new concise representation of frequent itemsets using generators and a positive border.Knowl. Inf. Syst. 17, 1 , 35-56 (2008)
5. Hamrouni T.: Key roles of closed sets and minimal generators in concise representations of frequent patterns. Intell. Data Anal, 16(4), 581-631 (2012)
6. Szathmary L., Napoli A., Kuznetsov S. O.: ZART: AMultifunctional Itemset Mining Algorithm. The 6th Intl Conf on Concept Lattices and Their Applications, 47–58 (2008)
7. Soulet A., Rioult F.: Efficiently Depth-First Minimal Pattern Mining. In: Tseng V.S., Ho T.B., Zhou ZH., Chen A.L.P., Kao HY. (eds) Advances in Knowledge Discovery and Data Mining. PAKDD 2014. LNCS. Springer, Cham, 8443, 28-39 (2014)
8. Soulet A., Rioult F.: Exact and Approximate Minimal Pattern Mining. In: Guillet F., Pinaud B., Venturini G. (eds) Advances in Knowledge Discovery and Management. SCI. Springer, Cham, 665, 61-81 (2017)
9. Phan Thành Huấn, Lê Hoài Bắc: Thuật toán hiệu quả khai thác tập hiếm tối thiểu. Hội nghị Khoa học Quốc gia FAIR lần thứ XI, 497-505, (2018)