Tạp chí Tin học và Điều khiển học, T.24, S.3 (2008), 204–216

THUẬT TOÁN HIỆU QUẢ KHAI PHÁ TẬP MỤC LỢI ÍCH CAO TRÊN CẤU TRÚC DỮ LIỆU CÂY

VŨ ĐỨC THI[1], NGUYỄN HUY ĐỨC[2]

[1] *Viện Công nghệ thông tin, Viện Khoa học và Công nghệ Việt Nam*
[2] *Khoa Thông tin - Máy tính, Trường Cao đẳng Sư phạm Trung ương*

Abstract. Mining High Utility Itemsets from a transaction database is to find itemsests that have utility above a user-specified threshold. The problem of finding High Utility Itemsets is challenging, because the anti-monotone property so useful for pruning the search space in conventional Frequent Itemset Mining does not apply to it. In this paper, we propose a new algorithm that mines high utility itemsets by bottom up traversal of a compressed utility pattern tree. We have tested our algorithm on several data sets, comparing it with the recent algorithms for High Utility Itemset Mining and the results show that our algorithm works more efficiently.

Tóm tắt. Khai phá tập mục lợi ích cao từ cơ sở dữ liệu giao tác là tìm các tập mục có lợi ích lớn hơn ngưỡng lợi ích cho trước. Khai phá tập mục lợi ích cao gặp nhiều khó khăn vì tính chất phản đơn điệu (anti-monotone) của tập mục thường xuyên không áp dụng được. Bài báo này đề xuất một thuật toán mới khai phá hiệu quả tập mục lợi ích cao bằng việc duyệt từ dưới lên của cây nén các giao tác của cơ sở dữ liệu. Thuật toán đã thử nghiệm trên một số tập dữ liệu, so sánh với các thuật toán khai phá tập mục lợi ích cao gần đây và kết quả cho thấy thuật toán làm việc hiệu quả hơn.

1. MỞ ĐẦU

Mô hình khai phá tập mục lợi ích cao đã được Yao và cộng sự đề xuất (Hong Yao và Hamilton, 2006; H. Yao, Hamilton và Butz, 2004) [6 – 8]. Trong mô hình khai phá tập mục lợi ích cao, giá trị của mục dữ liệu trong giao tác là một số (như số lượng đã bán của mặt hàng, gọi là giá trị khách quan), ngoài ra còn có bảng lợi ích cho biết lợi ích mang lại khi bán một đơn vị hàng đó (gọi là giá trị chủ quan, do người quản lý kinh doanh xác định). Lợi ích của một tập mục là số đo lợi nhuận đóng góp của tập mục đó trong cơ sở dữ liệu. Khai phá tập mục lợi ích cao là khám phá tất cả các tập mục có lợi ích không nhỏ hơn ngưỡng lợi ích tối thiểu quy định bởi người sử dụng.

Trong [6 – 8], Hong Yao và Howard Hamilton đã đề xuất phương pháp khai phá và các chiến lược tỉa dựa trên các tính chất của ràng buộc lợi ích, thể hiện trong hai thuật toán Umining và Umining-H. Các thuật tỉa mà hai thuật toán này áp dụng có khả năng thu gọn phần nào tập ứng viên, tuy vậy có những nhược điểm nên hiệu quả không cao.

Trong [9], Liu và cộng sự (Y. Liu, Liao và Choudhary, 2005) đã đưa ra khái niệm lợi ích của giao tác và lợi ích của tập mục tính theo lợi ích của các giao tác chứa nó, gọi là lợi ích TWU (Transaction-weighted Utilization). Tác giả đã chứng minh lợi ích theo giao tác TWU

có tính chất phản đơn điệu như tính chất của tập mục thường xuyên và tập tất cả các tập mục lợi ích cao chứa trong tập tất cả các tập mục lợi ích TWU cao. Nhờ những tính chất này của lợi ích TWU, Liu đã đề xuất thuật toán gồm hai pha để khai phá tập mục lợi ích cao. Thuật toán rút gọn nhanh không gian tìm kiếm nhờ áp dụng tính chất phản đơn điệu của lợi ích TWU. Tuy nhiên, thuật toán thực hiện kém hiệu quả khi khai phá các tập dữ liệu dày và mẫu dài vì tốn nhiều thời gian cho việc sinh các tập ứng viên và tính lợi ích TWU của các tập ứng viên đó trong mỗi lần duyệt cơ sở dữ liệu. Thuật toán phải duyệt cơ sở dữ liệu nhiều lần, số lần duyệt bằng với chiều dài của mẫu dài nhất tìm được, do đó, khi số mục dữ liệu lớn thì khối lượng tính toán là vô cùng lớn.

Bài báo đề xuất một thuật toán hiệu quả khai phá tập mục lợi ích cao theo cách tiếp cận khác. Thuật toán sử dụng cấu trúc cây FP-tree được J. Han giới thiệu năm 2000 [11] để nén toàn bộ cơ sở dữ liệu lên cấu trúc dữ liệu cây gọi là cây UP-tree, cây đó đầy đủ thông tin cho khai phá tập mục lợi ích cao. Khai phá cây UP-tree thực hiện theo phương pháp không đệ quy theo ý tưởng của thuật toán COFI-tree do Mohammad El-Hajj và Osmar R. Zaiane đề xuất năm 2003 [12, 13]. Thuật toán thực hiện hiệu quả vì 3 lí do: 1) Chỉ cần duyệt cơ sở dữ liệu 2 lần, 2) Tránh được khối lượng tính toán khổng lồ trong quá trình sinh các tập mục ứng viên và 3) Sử dụng tiết kiệm bộ nhớ.

Nội dung tiếp theo của bài báo gồm: Phần 2 nêu một số định nghĩa, thuật ngữ và phát biểu bài toán khai phá tập mục lợi ích cao. Phần 3 tóm tắt nội dung của phương pháp khai phá tập mục thường xuyên nhờ cây COFI-tree. Phần 4 trình bày thuật toán mới khai phá tập mục lợi ích cao. Phần 5 đánh giá thuật toán và kết luận dựa trên việc phân tích thuật toán và các thử nghiệm.

2. BÀI TOÁN KHAI PHÁ TẬP MỤC LỢI ÍCH CAO

Phần này nêu một số định nghĩa và thuật ngữ mô tả bài toán khai phá tập mục lợi ích cao theo $[6-8]$.

Cho tập các mục (item) $I = \{i_1, i_2, ..., i_n\}$. Một giao tác (transaction) T là một tập con của I, $T \subseteq I$. Cơ sở dữ liệu là một tập các giao tác $DB = \{T_1, T_2, ..., T_m\}$. Mỗi giao tác được gán một định danh TID. Một tập mục con $X \subseteq I$, gồm k mục phân biệt được gọi là một k-tập mục. Giao tác T gọi là chứa tập mục X nếu $X \subseteq T$.

Định nghĩa 2.1. Ta gọi giá trị của mục i_p trong giao tác T_q (giá trị tại cột i_p hàng T_q của cơ sở dữ liệu) là giá trị khách quan (objective value) của mục i_p tại giao tác T_q, ký hiệu là $o(i_p, T_q)$.

Định nghĩa 2.2. Ta gọi giá trị do nhà kinh doanh gán cho mục i_p trong cơ sở dữ liệu, dựa trên đánh giá lợi nhuận mà mỗi đơn vị mục dữ liệu có thể đem lại, là giá trị chủ quan (subjective value) của mục i_p và ký hiệu là $s(i_p)$.

Thông thường, giá trị chủ quan của các mục được cho trong một bảng kèm theo cơ sở dữ liệu và gọi là bảng lợi ích. Chẳng hạn, cơ sở dữ liệu các giao tác và bảng lợi ích ở Bảng 2.1 và Bảng 2.2, giá trị khách quan của mục B tại giao tác T_2 là $o(B, T_2) = 12$, giá trị chủ quan của B là $s(B) = 5$, tức là trong giao tác T_2 bán được 12 đơn vị mặt hàng B và mỗi đơn vị mặt hàng B lãi 5\$.

Lợi ích của một mục trong một giao tác được đánh giá thông qua hàm 2 biến sau.

Định nghĩa 2.3. Ký hiệu x là giá trị khách quan, y là giá trị chủ quan của một mục. Một hàm 2 biến $f(x, y) : R \times R \to R$, đơn điệu tăng theo x và theo y, được gọi là hàm lợi ích.

Thông thường hàm lợi ích được xác định như sau $f(x, y) = x \times y$.

Định nghĩa 2.4. Cho hàm lợi ích $f(x, y)$. Lợi ích của mục i_p tại giao tác T_q, ký hiệu $u(i_p, T_q)$ là giá trị của hàm $f(x, y)$ tại $o(i_p, T_q)$ và $s(i_p)$, tức là $u(i_p, T_q) = f(o(i_p, T_q), s(i_p))$.

TID	A	B	C	D	E
T1	0	12	2	0	2
T2	0	12	0	2	1
T3	2	0	1	0	1
T4	1	0	0	2	1
T5	0	0	4	0	2
T6	1	2	0	0	0
T7	0	20	0	2	1
T8	3	0	25	6	1
T9	1	2	0	0	0
T10	0	0	16	0	1

Bảng 2.1. Cơ sở dữ liệu giao tác

Mục	Lợi nhuận ($/đơn vị)
A	3
B	5
C	1
D	3
E	5

Bảng 2.2. Bảng lợi ích

Định nghĩa 2.5. Cho tập mục X chứa trong giao tác T_q. Lợi ích của tập mục X tại giao tác T_q, ký hiệu $u(X, T_q)$ là tổng lợi ích của tất cả các mục i_p thuộc X tại giao tác T_q, tức là

$$u(X, T_q) = \sum_{i_p \in X \subseteq T_q} u(i_p, T_q).$$

Ký hiệu db_X là tập các giao tác chứa tập mục X trong cơ sở dữ liệu DB, tức là

$$db_X = \{T_q | X \subseteq T_q, T_q \in DB\}.$$

Định nghĩa 2.6. Lợi ích của tập mục X trong cơ sở dữ liệu DB, ký hiệu $u(X)$ là tổng lợi ích của tập mục X tại các giao tác thuộc db_X, tức là

$$u(X) = \sum_{T_q \in db_X} u(X, T_q) = \sum_{T_q \in db_X} \sum_{i_p \in X} u(i_p, T_q).$$

Ví dụ, trong cơ sở dữ liệu Bảng 2.1 và Bảng 2.2, $u(B, T_2) = 12 \times 5 = 60$. Xét $X = \{B, D\}$, $u(X, T_2) = u(B, T_2) + u(D, T_2) = 12 \times 5 + 2 \times 3 = 66$, có 2 giao tác T_2 và T_7 chứa tập mục X, do đó $db_X = \{T_2, T_7\}$, $u(X) = u(X, T_2) + u(X, T_7) = (12 \times 5 + 2 \times 3) + (20 \times 5 + 2 \times 3) = 172$.

Định nghĩa 2.7. Cho ngưỡng lợi ích $minutil (> 0)$ và xét tập mục X, X được gọi là tập mục lợi ích cao nếu $u(X) \geq minutil$. Trường hợp ngược lại, X được gọi là tập mục lợi ích thấp.

Định nghĩa 2.8. Cho cơ sở dữ liệu giao tác DB và ngưỡng lợi ích $minutil$, bài toán khai phá tập mục lợi ích cao là việc tìm tập HU tất cả các tập mục lợi ích cao, tức là tập $HU = \{X | X \subseteq I,\ u(X) \geq minutil\}$.

Có thể coi bài toán khai phá tập mục thường xuyên là trường hợp đặc biệt của bài toán khai phá tập mục lợi ích cao khi tất cả các mục dữ liệu đều có giá trị khách quan bằng 0 hoặc 1 và giá trị chủ quan bằng 1. Dễ thấy, tính chất Apriori không còn đúng với ràng ruộc

lợi ích. Chẳng hạn, trong cơ sở dữ liệu Bảng 2.1, ta có $u(BC) = 62 < 72 = u(BCE)$, trong khi đó $u(BC) = 62 > 0 = u(BCD)$.

Phần tiếp theo sau đây trình bày nội dung cơ bản của phương pháp khai phá tập mục thường xuyên nhờ cấu trúc cây COFI-tree.

3. KHAI PHÁ TẬP MỤC THƯỜNG XUYÊN TRÊN CẤU TRÚC CÂY COFI-TREE

Năm 2000, Han đã giới thiệu cấu trúc cây FP-tree (frequent-pattern tree) và thuật toán FP-growth để khai phá các mẫu thường xuyên trên cây FP-tree [11]. Thuật toán nén toàn bộ cơ sở dữ liệu lên một cấu trúc dữ liệu nhỏ hơn là cây FP-tree, tránh được việc duyệt nhiều lần cơ sở dữ liệu (thuật toán chỉ duyệt cơ sở dữ liệu 2 lần). Tiếp theo thuật toán khai phá trên cây bằng cách phát triển dần các mẫu mà không sinh các tập mục ứng viên, do đó tránh được khối lượng tính toán lớn. Tuy vậy, thuật toán FP-growth khai phá cây FP-tree sử dụng phương pháp đệ quy, do vậy đòi hỏi khối lượng tính toán lớn và cần nhiều bộ nhớ. Năm 2003, nhóm tác giả Mohammad El-Hajj và Osmar R. Zaiane ở đại học Alberta Edmonton, Canada đã đề xuất thuật toán khai phá cây FP-tree dựa trên một cấu trúc cây nhỏ COFI-tree (Co-Occurrence Frequent Item tree) không đệ quy [12, 13]. Thuật toán COFI-tree có nhiều ưu điểm hơn thuật toán FP-growth.

Thuật toán COFI-tree gồm 2 giai đoạn chính. Giai đoạn thứ nhất xây dựng cây FP-tree. Giai đoạn thứ hai khai phá cây FP-tree nhờ cấu trúc cây COFI-tree cho từng mục dữ liệu trong bảng đầu mục của cây FP-tree.

Mỗi nút của cây FP-tree gồm 3 trường: tên mục dữ liệu, độ hỗ trợ và một con trỏ (trỏ đến nút tiếp theo cùng tên trên cây hoặc là null nếu không có). Cây FP-tree có một bảng đầu mục (header table). Mỗi mục của bảng có 3 trường: tên mục dữ liệu, biến độ hỗ trợ và con trỏ (trỏ đến nút đầu tiên biểu diễn mục dữ liệu này trong cây).

Cây COFI-tree có bảng đầu mục giống như cây FP-tree nhưng thứ tự các mục dữ liệu ngược lại. Mỗi mục trong bảng đầu mục chứa 3 trường: tên mục dữ liệu, độ hỗ trợ địa phương (số lần xuất hiện của mục dữ liệu trong cây COFI-tree) và con trỏ (trỏ đến nút đầu tiên biểu diễn mục dữ liệu này trong cây). Một danh sách liên kết được duy trì giữa các nút cùng tên để thuận lợi cho quá trình khai phá. Mỗi nút của cây COFI-tree có 4 trường: tên mục dữ liệu, hai biến s và p (biến s biểu diễn độ hỗ trợ của nút, biến p cho biết số lần nút đó đã tham gia tạo mẫu), con trỏ (trỏ đến nút tiếp theo cùng tên trên cây).

Chi tiết về cây COFI-tree có thể tham khảo trong [12, 13].

4. THUẬT TOÁN KHAI PHÁ TẬP MỤC LỢI ÍCH CAO TRÊN CẤU TRÚC CÂY

Trong [9], Y.Liu đã đưa ra khái niệm lợi ích TWU của tập mục tính theo lợi ích các giao tác chứa nó.

Định nghĩa 4.1. (Lợi ích giao tác-Transaction Utility) Ta gọi tổng lợi ích của tất cả các mục có mặt trong T_q là lợi ích của giao tác T_q, ký hiệu là $tu(T_q)$, ta có

$$tu(T_q) = \sum_{i_p \in T_q} u(i_p, T_q).$$

Định nghĩa 4.2. (Transaction Weighted Utility-TWU) Lợi ích theo giao tác TWU của tập mục X, ký hiệu $twu(X)$ là tổng lợi ích của tất cả các giao tác chứa X trong cơ sở dữ liệu, tức là

$$twu(X) = \sum_{T_q \in DB \wedge X \subseteq T_q} tu(T_q).$$

Ví dụ, trong cơ sở dữ liệu Bảng 2.1 và Bảng 2.2, $tu(T_2) = 12 \times 5 + 2 \times 3 + 1 \times 5 = 71$.

Xét $X = DE$, $db_X = \{T_2, T_4, T_7, T_8\}$, $twu(X) = tu(T_2) + tu(T_4) + tu(T_7) + tu(T_8) = 253$.

Nhận xét: Vì $u(X, T_q) \leq tu(T_q)$ nên $u(X) = \sum_{T_q \in DB \wedge X \subseteq T_q} u(X, T_q) \leq \sum_{T_q \in DB \wedge X \subseteq T_q} tu(T_q) = twu(X)$.

Có thể coi $twu(X)$ như là cận trên của $u(X)$. Với ngưỡng lợi ích *minutil*, nếu X là tập mục lợi ích cao thì X cũng là tập mục lợi ích TWU cao vì $twu(X) \geq u(X) \geq minutil$, ngược lại, nếu X là tập mục lợi ích TWU thấp thì X cũng là tập mục lợi ích thấp.

Theo [9], ràng buộc lợi ích TWU có tính chất phản đơn điệu, tức là mọi tập mục cha của tập mục lợi ích TWU thấp cũng là tập mục lợi ích TWU thấp. Do vậy, nếu X là tập mục lợi ích TWU thấp, $twu(X) < minutil$, thì tập X và mọi tập cha của X đều là tập mục lợi ích thấp, chúng có thể loại bỏ trong quá trình khai phá tập mục lợi ích cao.

Dựa trên ý tưởng của thuật toán COFI-tree khai phá tập mục thường xuyên, bài báo đề xuất thuật toán mới khai phá tập mục lợi ích cao, gọi là thuật toán COUI-Mine (Co-Occurrence Utility Items Mine). Thuật toán COUI-Mine gồm hai giai đoạn. Giai đoạn thứ nhất xây dựng cây chứa toàn bộ thông tin để khai phá tập mục lợi ích cao, gọi là cây UP-tree (Utility Pattern tree) và giai đoạn thứ hai khai phá cây để tìm các tập mục lợi ích cao nhờ cấu trúc dữ liệu cây COUI-tree (Co-Occurrence Utility Items tree).

4.1. Xây dựng cây UP-tree

Cây UP-tree có hai phần là bảng đầu mục và cây. Bảng đầu mục có cấu trúc giống như bảng đầu mục của cây FP-tree trong [11] nhưng có bổ sung thêm một số trường. Mỗi mục của bảng gồm 4 trường: tên mục dữ liệu, lợi ích của mỗi đơn vị mục dữ liệu, số lượng mục dữ liệu đó trong toàn bộ cơ sở dữ liệu và con trỏ trỏ đến nút đầu tiên biểu diễn mục dữ liệu này trên cây. Các mục dữ liệu trong bảng đầu mục sắp xếp giảm dần theo độ hỗ trợ của chúng. Mỗi nút của cây UP-tree gồm 4 trường: tên mục dữ liệu, giá trị *twu* của mục dữ liệu, mảng lưu số lượng các mục dữ liệu trên đường đi từ nút đó đến nút gốc của cây và con trỏ (trỏ đến nút tiếp theo cùng nhãn trên cây hoặc là *null* nếu không có).

Thuật toán cần duyệt cơ sở dữ liệu hai lần. Lần duyệt thứ nhất, thuật toán tính lợi ích các giao tác, số lần xuất hiện (độ hỗ trợ), tổng số lượng, lợi ích theo giao tác TWU của từng mục dữ liệu. Tiếp theo, thuật toán loại bỏ các mục dữ liệu có lợi ích TWU thấp, các mục còn lại sắp xếp giảm dần theo độ hỗ trợ của chúng và xây dựng bảng đầu mục của cây.

Lần duyệt thứ hai, mỗi giao tác chỉ lấy ra các mục dữ liệu có mặt trong bảng đầu mục, sắp xếp các mục này theo thứ tự giảm dần của độ hỗ trợ và cất lên cây UP-tree như sau. Giả sử danh sách là $[x|L]$ với x là mục dữ liệu đầu và L là phần còn lại. Ta kiểm tra xem

mục x có là nhãn của nút con nào của nút gốc không? Nếu có thì điều chỉnh nút con này như sau: Biến *twu* được tăng thêm lợi ích của giao tác, mảng số lượng được tăng thêm số lượng các mục dữ liệu tương ứng trong giao tác. Ngược lại, thêm cho nút gốc một nút con mới với nhãn là x. Tại nút con mới này, gán biến *twu* bằng lợi ích của giao tác và đặt số lượng các mục dữ liệu tương ứng trong giao tác vào mảng số lượng. Tiếp theo, coi nút hiện thời là nút gốc và lặp lại tương tự với các mục dữ liệu tiếp theo trong giao tác. Khi thêm một nút mới nhãn x vào cây cần duy trì danh sách liên kết giữa nút này với mục dữ liệu x tương ứng trong bảng đầu mục. Thuật toán làm tương tự như vậy cho đến giao tác cuối cùng.

Ta minh họa xây dựng cây UP-tree qua xét cơ sở dữ liệu Bảng 2.1 và Bảng 2.2, ngưỡng lợi ích bằng 30% của tổng lợi ích, $minutil = 30\% \times 398 = 119,4$.

Bảng 4.1. Lợi ích các giao tác của cơ sở dữ liệu Bảng 2.1 và Bảng 2.2

TID	A	B	C	D	E	tu
T1	0	12	2	0	2	72
T2	0	12	0	2	1	71
T3	2	0	1	0	1	12
T4	1	0	0	2	1	14
T5	0	0	4	0	2	14
T6	1	2	0	0	0	13
T7	0	20	0	2	1	111
T8	3	0	25	6	1	57
T9	1	2	0	0	0	13
T10	0	0	16	0	1	21
Tổng	8	48	48	12	10	398

Bảng 4.2. Số lượng, *twu*, độ hỗ trợ của các mục

Mục dữ liệu	Lợi nhuận/ đơn vị	Số lượng	*twu*	Độ hỗ trợ
A	3	8	109	5
B	5	48	280	5
C	1	48	176	5
D	3	12	253	4
E	5	10	372	8

Duyệt cơ sở dữ liệu lần thứ nhất, tính được lợi ích của các giao tác (Bảng 4.1), tổng số lượng, số lần xuất hiện, lợi ích TWU của từng mục dữ liệu (Bảng 4.2). Vì $twu(A) = 109 <$ *minutil* nên ta loại mục A, các mục còn lại sắp theo thứ tự giảm dần của độ hỗ trợ và nhận được Bảng 4.3. Từ Bảng 4.3 xây dựng bảng đầu mục của cây UP-tree.

Bảng 4.3. Danh sách các mục dữ liệu có lợi ích TWU cao đã sắp thứ tự

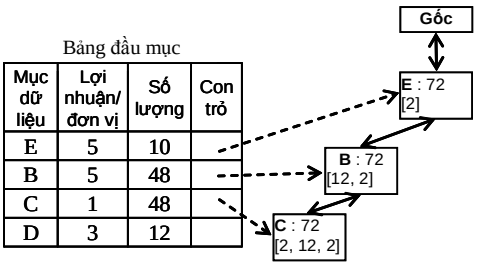
Mục dữ liệu	Lợi nhuận/ đơn vị	Số lượng	*twu*	Độ hỗ trợ
E	5	10	372	8
B	5	48	280	5
C	1	48	176	5
D	3	12	253	4

Duyệt cơ sở dữ liệu lần thứ 2, mỗi giao tác được đọc ra các mục dữ liệu có lợi ích TWU cao, sắp chúng theo thứ tự của bảng đầu mục. Giao tác thứ nhất (B:12, C:2, E:2) được sắp thành (E:2, B:12, C:2), lợi ích giao tác $tu = 72$ và sinh ra nhánh thứ nhất trên cây (Hình 4.1). Ở đây biểu diễn nút của cây là một hình chữ nhật gồm tên mục dữ liệu, lợi ích *twu*, mảng số lượng các mục. Giao tác T_2 =(E:1, B:12, D:2) cất lên cây có chung đoạn EB với

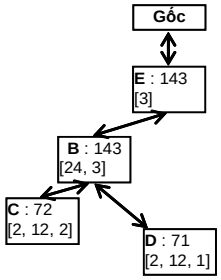
đường đi đã có, trường *twu* của 2 nút E và B tăng thêm 71 là lợi ích của giao tác, mảng số lượng các mục được tăng thêm tương ứng. Hình 4.2 là cây sau khi cất $T1$ và $T2$. Hình 4.3 biểu diễn kết quả xây dựng cây.

Bảng 4.4. Lợi ích và các giao tác đã sắp xếp lại các mục dữ liệu

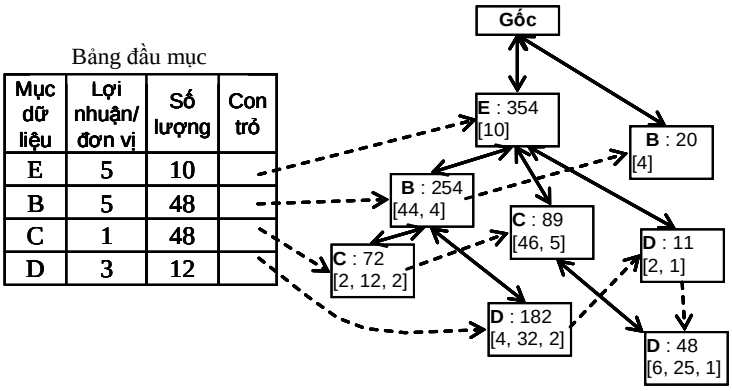
TID	E	B	C	D	tu
T1	2	12	2	0	72
T2	1	12	0	2	71
T3	1	0	1	0	6
T4	1	0	0	2	11
T5	2	0	4	0	14
T6	0	2	0	0	10
T7	1	20	0	2	111
T8	1	0	25	6	48
T9	0	2	0	0	10
T10	1	0	16	0	21
Tổng	10	48	48	12	374



Hình 4.1. Cây sau khi cất giao tác *T1*



Hình 4.2. Cây sau khi cất giao tác *T1, T2*



Hình 4.3. Cây UP-tree cuả CSDL Bảng 2.1 và 2.2

Ta thấy rằng các giao tác của cơ sở dữ liệu Bảng 2.1 và Bảng 2.2 đã được biểu diễn lên cây. Cây UP-tree chứa đủ thông tin để khai phá các tập mục lợi ích cao.

Thuật toán xây dựng cây UP-tree như sau.

Thuật toán 1. Xây dựng cây UP-tree

Input: Cơ sở dữ liệu giao tác DB, hàm lợi ích, ngưỡng lợi ích *minutil*.

Output: Cây UP-tree, cây nén các giao tác của cơ sở dữ liệu DB để khai phá lợi ích cao.
Method:
1. Duyệt cơ sở dữ liệu lần thứ nhất

1.1. Tính lợi ích các giao tác, số lần xuất hiện, tổng số lượng, lợi ích TWU của từng mục dữ liệu.

1.2. Chọn ra tập U các mục dữ liệu có lợi ích TWU cao, sắp xếp tập U theo thứ tự giảm dần của độ hỗ trợ được danh sách $UList$ và xây dựng bảng đầu mục của cây.
2. Tạo cây

2.1. Tạo nút gốc R của cây.

2.1. Duyệt cơ sở dữ liệu lần thứ 2, với mỗi giao tác T trong DB, chọn ra các mục dữ liệu có lợi ích TWU cao, sắp các mục này theo thứ tự của $UList$. Giả sử danh sách đã sắp của T là $([x|L])$, ở đó x là mục dữ liệu đầu và L là phần còn lại của danh sách, gọi hàm $insert\_tree([x|L], R)$.

Hàm $insert\_tree([x|L], R)$ làm việc như sau: Nếu R có nút con N nhãn x thì điều chỉnh các trường của nút N, ngược lại, tạo nút N mới là nút con của nút R và gán nhãn của nút N là x, bổ sung thêm đường liên kết của các nút cùng nhãn đến nút N này. Nếu L khác rỗng thì gọi đệ quy hàm $insert\_tree(L, N)$.

Bài báo đề xuất một kỹ thuật lưu số lượng các mục dữ liệu trong giao tác vào nút của cây như sau:

Xét giao tác $T = (A_1 : s_1, A_2 : s_2, ..., A_j : s_j, ..., A_k : s_k)$ với lợi ích $tu(T)$, ở đó cặp $A_j : s_j$ biểu diễn tên mục dữ liệu và số lượng của mục đó trong giao tác. Giả sử ta đã cất $j-1$ mục đầu lên cây, bây giờ xét mục A_j tại nút N. Nút N có nhãn A_j, đường đi từ nút N lên nút gốc là $A_j, A_{j-1}, ..., A_1$. Nút N có trường twu và mảng số lượng các mục tương ứng với các mục trên đường đi từ nút N lên nút gốc.

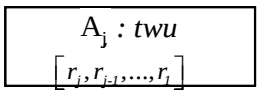
Nếu N là nút đã có của cây với mảng số lượng là $[r_j, r_{j-1}, ..., r_1]$, thay đổi tại nút N như sau

$$N.twu := N.twu + tu(T),$$

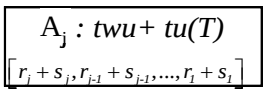
cộng tương ứng dãy số $s_j, s_{j-1}, ..., s_1$ vào mảng $[r_j, r_{j-1}, ..., r_1]$ và nhận được mảng $[r_j + s_j, r_{j-1} + s_{j-1}, ..., r_1 + s_1]$ (Hình 4.4a và 4.4b).

Nếu N là nút mới được tạo ra, ta thiết đặt tại nút N như sau $N.twu := tu(T)$, mảng số lượng các mục là $[s_j, s_{j-1}, ..., s_1]$ (Hình 4.5).

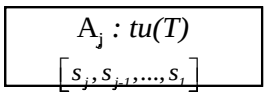
Chú ý: Xét nút N trong Hình 4.4.a, r_j là số lượng mục dữ liệu A_j, tức là phần tử đầu tiên của mảng số lượng tại một nút là số lượng của mục dữ liệu mà nút đó biểu diễn.



Hình 4.4a. Nút N



Hình 4.4b. Nút N sau khi điều chỉnh



Hình 4.5. Nút N mới tạo

Từ cách xây dựng cây UP-tree như trên, đường đi từ một nút N Hình 4.4a có nhãn A_j lên nút gốc của cây là $A_j, A_{j-1}, ..., A_1$ xác định một mẫu $X = (A_j : r_j, A_{j-1} : r_{j-1}, ..., A_1 : r_1)$ và lợi ích TWU của mẫu X bằng trường twu của nút N, $twu(X) = N.twu$. Ví dụ, trong

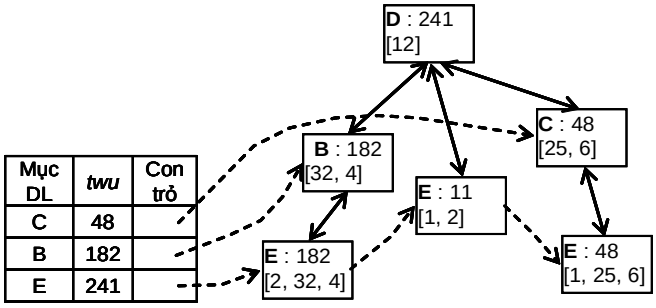
cây UP-tree Hình 4.3, đường đi từ nút B lên nút gốc xác định mẫu $(B : 44, E : 4)$ và $twu(BE) = 254$.

4.2. Khai phá cây UP-tree

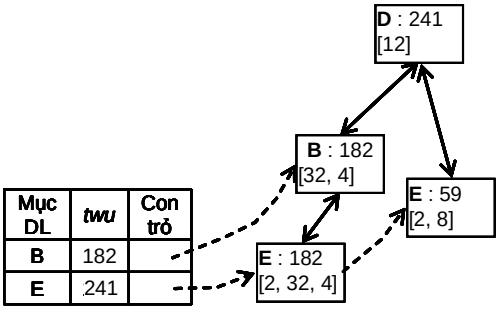
Xét lần lượt các mục dữ liệu từ dưới lên trong bảng đầu mục của cây UP-tree, với mỗi mục dữ liệu này, xây dựng cây COUI-tree, khai phá cây COUI-tree để tìm các mẫu (tập mục) lợi ích cao, sau khi khai phá xong loại bỏ cây này và xây dựng cây COUI-tree cho mục dữ liệu tiếp theo.

Cây COUI-tree có bảng đầu mục, mỗi mục trong bảng đầu mục chứa 3 trường: Tên mục dữ liệu, giá trị twu và con trỏ. Bảng đầu mục chứa các mục dữ liệu tham gia xây dựng cây và được sắp thứ tự theo thứ tự ngược với thứ tự trong bảng đầu mục của cây UP-tree. Nút của cây COUI-tree gồm các trường như nút của cây UP-tree. Quá trình xây dựng cây COUI-tree giống như xây dựng cây UP-tree.

Ta minh họa thuật toán qua xét mục dữ liệu đầu tiên D.



Hình 4.6. Cây D-COUI-tree



Hình 4.7. Cây D-COUI-tree sau khi tỉa mục C

Từ con trỏ của bảng đầu mục tìm thấy 3 nút có nhãn D, đường đi từ các nút này lên nút gốc xác định 3 mẫu là (D:4,B:32, E:2) với twu(DBE) =182, (D:2,E:1) với twu(DE)=11 và (D:6, C:25, E:1) với twu(DCE)=48. Lưu các mẫu này lên cây D-COUI-tree giống như chèn các giao tác vào cây UP-tree với chú ý là phải điều chỉnh trường twu của bảng đầu mục của cây cho phù hợp. Hình 4.6 biểu diễn cây D-COUI-tree.

Khai phá cây D-COUI-tree

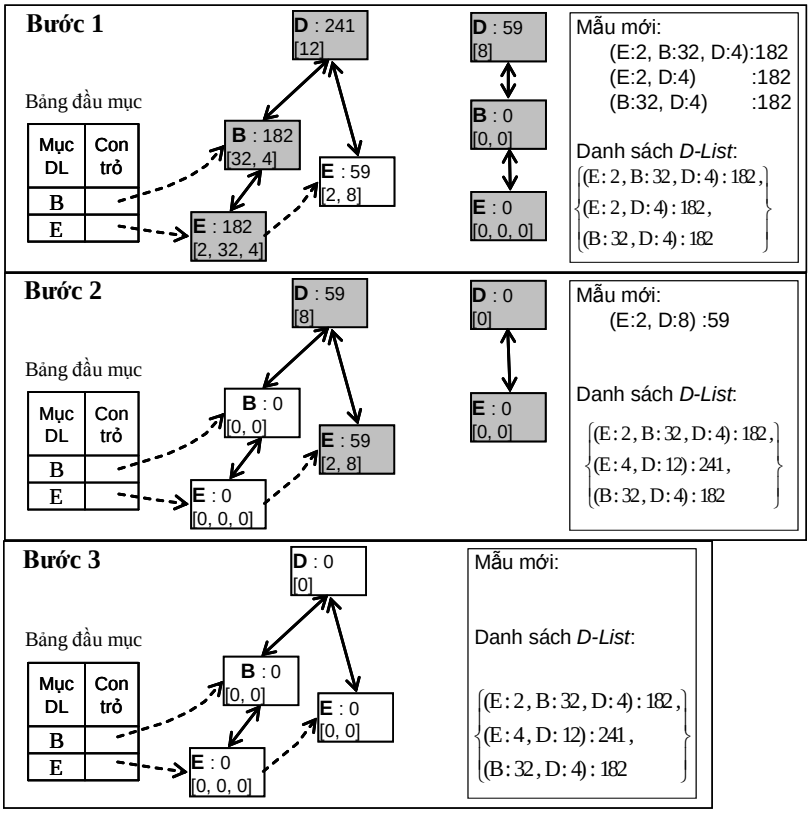
Bước tỉa cây: Xét các mục dữ liệu trong bảng đầu mục, mục C có lợi ích TWU thấp

$twu(C) = 48 < minutil$, vì vậy tỉa mục dữ liệu này (Hình 4.7).

Xét mục E: Từ con trỏ của mục E trên bảng đầu mục tìm được 2 nút trên cây có nhãn E. Đường đi từ nút E thứ nhất lên nút gốc xác định mẫu (E:2,B:32, D:4) với $twu = 182$. Kết nạp mẫu này cùng các mẫu con có chứa D của nó vào danh sách $D - list$ chứa các ứng viên của mục D, $D - list = \{$(E:2,B:32, D:4):182; (E:2, D:4):182; (B:32, D:4):182$\}$. Điều chỉnh biến twu và mảng số lượng của các nút nhãn E, B và D trên đường đi: biến twu giảm đi 182, mảng số lượng được trừ đi tương ứng (Bước 1).

Đường đi từ nút E thứ hai lên nút gốc xác định mẫu (E:2,D:8) với $twu = 59$. Kết nạp mẫu này vào $D - List$. Trong $D - List$ đã có mẫu (E:2,D:4):182 nên điều chỉnh mẫu này thành (E:4,D:12):241 (Bước 2).

Xét mục B tiếp theo, nút nhãn B trên cây có $twu = 0$, do vậy không sinh ra mẫu nào nữa (Bước 3). Hình 4.8 minh họa các bước khai phá cây D-COUI-tree.



Hình 4.8. Các bước khai phá cây D-COUI-tree

Kết thúc khai phá cây D-COUI-tree, nhận được danh sách $D - list$. Duyệt danh sách $D - list$, với mỗi tập ứng viên $X \in D - list$, tính lợi ích thực sự của nó, nếu $u(X) \geq minutil$ thì X là tập mục lợi ích cao.

Mẫu (E:2, B:32, D:4) có u(EBD)=2×5+32×5+4×3=182, mẫu (B:32, D:4) có u(BD)=172, mẫu (E:4, D:12) có u(ED)=56. So với $minutil = 119,4$, có hai tập mục lợi ích cao được phát hiện là EBD và BD, tập $HU = \{$EBD(182), BD(172)$\}$.

Thuật toán xóa cây D-COUI-tree và danh sách $D - List$, tiếp tục xây dựng cây C-COUI-

tree và B-COUI-tree. Khai phá cây C-COUI-tree không nhận được tập mục lợi ích cao nào. Khai phá cây B-COUI-tree nhận được EB(240), B(240).

Hoàn thành khai phá cây UP-tree, thuật toán tìm được

$$HU = \{\text{EBD(182), BD(172), EB(240), B(240)}\}.$$

Sau đây là thuật toán xây dựng và khai phá các cây COUI-tree.

Thuật toán 2. Khai phá cây UP-tree

Input: Cây UP-tree, hàm lợi ích, ngưỡng lợi ích *minutil*.
Output: Tập HU chứa tất cả các tập mục lợi ích cao của cơ sở dữ liệu DB.
Method:
1. Xét từ dưới lên của bảng đầu mục của UP-tree, biến A nhận mục dữ liệu đầu tiên;
2. repeat
3. Tính lợi ích của mục A; // từ số lượng và lợi nhuận lưu trong bảng đầu mục.
4. if $u(A) \geq minutil$ then $HU = HU \cup \{A\}$;
5. Tạo nút gốc của cây có nhãn A và gán *twu* bằng 0, mảng số lượng bằng 0;
6. for each (nút N trên cây UP-tree có nhãn A) // tìm theo con trỏ
7. begin
8. - Đọc biến *twu* và mảng số lượng các mục $[r_j, r_{j-1}, ..., r_1]$ của nút N;
9. - Xác định mẫu $(A : r_j, A_{j-1} : r_{j-1}, ..., A_1 : r_1)$ từ đường đi từ nút N lên gốc của cây;
10. - Chèn mẫu $(A : r_j, A_{j-1} : r_{j-1}, ..., A_1 : r_1)$ vào cây (A)-COUI-tree;
11. end;
12. Gọi hàm MineCOUI-tree (A); // hàm thực hiện khai phá cây (A)-COUI-tree.
13. Xóa cây (A)-COUI-tree;
14. Biến A nhận mục dữ liệu tiếp theo trong bảng đầu mục của cây UP-tree;
15. Until (A là mục dữ liệu cuối cùng của bảng đầu mục của cây UP-tree);
16. Tính lợi ích của mục A; // tính lợi ích mục dữ liệu cuối cùng trong bảng đầu mục.
17. if $u(A) \geq minutil$ then $HU := HU \cup \{A\}$;
18. Return HU;

Function MineCOFI-tree (A); // hàm thực hiện khai phá cây (A)-COUI-tree.
Method:
1. $(A) - List := \phi$ // khởi tạo danh sách các mẫu ứng viên chứa mục A là rỗng.
2. Tỉa cây: Duyệt bảng đầu mục của (A)-COUI-tree, tỉa các mục dữ liệu có $twu < minutil$;
// tỉa trên bảng đầu mục và các nút tương ứng trên cây.
3. for each (mục dữ liệu B của bảng đầu mục của cây) // từ dưới lên.
4. for each (nút N trên cây (A)-COUI-tree có nhãn B) // tìm theo con trỏ
5. begin
6. - Đọc biến *twu* và mảng số lượng các mục của nút N;
7. - Xác định mẫu X từ đường đi từ nút N lên nút gốc của cây;
8. - Kết nạp X và các mẫu con của X có chứa mục A vào $(A) - List$;
9. - Điều chỉnh biến *twu* và mảng số lượng các mục của các nút trên đường đi từ nút N lên nút gốc của cây;
10. end; // hoàn thành khai phá cây (A)-COUI-tree.
11. for each $Y \in (A) - List$ // duyệt các mẫu ứng viên của danh sách $(A) - List$.

12. begin
13. - Tính lợi ích $u(Y)$ của mẫu Y;
14. - if $u(Y) \geq minutil$ then $HU = HU \cup \{Y\}$;
15. end;
16. Return HU;

5. ĐÁNH GIÁ THUẬT TOÁN VÀ KẾT LUẬN

Thuật toán đã được thử nghiệm trên một số cơ sở dữ liệu giao tác được tạo bằng phương pháp tạo số ngẫu nhiên. Từ các kết quả thử nghiệm và phân tích thuật toán, cho thấy thuật toán COUI-Mine có những ưu điểm sau:

- Thuật toán chỉ cần duyệt cơ sở dữ liệu hai lần để xây dựng cây UP-tree.

- Khai phá cây UP-tree nhờ cấu trúc cây COUI-tree theo phương pháp không đệ quy, do đó tiết kiệm bộ nhớ và giảm thời gian tính toán.

- Các cây COUI-tree thực chất là kết quả chiếu của cây UP-tree cho từng mục dữ liệu. Cây COUI-tree cho mỗi mục dữ liệu biểu diễn các mục dữ liệu cùng xuất hiện với mục dữ liệu đó trong các giao tác của cơ sở dữ liệu. Cách làm này đã chia bài toán thành nhiều bài toán nhỏ đơn giản hơn.

- Thuật toán tránh được khối lượng tính toán lớn vì không sinh các tập mục ứng viên theo cách tiếp cận sinh ứng viên rồi kiểm tra ràng buộc như một số thuật toán khác thực hiện.

- Thuật toán đã sử dụng khái niệm lợi ích TWU một cách hiệu quả, nhờ đó giảm đáng kể việc sinh ra các tập mục ứng viên dư thừa.

Với những ưu điểm trên đây và qua kết quả thử nghiệm, cho thấy thuật toán COUI-Mine là thuật toán hiệu quả để khai phá tập mục lợi ích cao.

TÀI LIỆU THAM KHẢO

[1] Vũ Đức Thi, *Cơ sở dữ liệu - Kiến thức và thực hành*, Nhà xuất bản Thống kê, năm 1997.

[2] Nguyễn Thanh Tùng, Khai phá tập mục lợi ích cao trong cơ sở dữ liệu, *Tạp chí Tin học và Điều khiển học* **23** (4) (2007) 364–373.

[3] Nguyễn Huy Đức, Khai phá luật kết hợp trong cơ sở dữ liệu lớn, *Kỷ yếu Hội thảo khoa học Quốc gia lần thứ nhất về nghiên cứu cơ bản và ứng dụng CNTT*, Hà Nội, 10/2003.

[4] R. Agrawal and R. Srikant, Fast algorithms for mining association rules, *Proceedings of 20th International Conference on Very Large Databases*, Santiago, Chile, 1994.

[5] B. Goethals and M. Zaki, Advances in frequent itemset mining implementations: Introduction to fimi03, *Workshop on Frequent Itemset Mining Implementations (FIMI03) in Conjunction with IEEE-ICDM*, 2003.

[6] H. Yao, H. J. Hamilton, and C. J. Butz, A foundational approach to mining itemset utilities from databases, *Proceedings of the 4th SIAM International Conference on Data Mining*, Florida, USA, 2004.

[7] H. Yao, H. J. Hamilton, Mining itemsets utilities from transaction databases, *Data and Knowledge Engeneering* **59** (3) (2006).

[8] H. Yao, H. J. Hamilton, and L. Geng, A unified framework for utility based measures for mining itemsets, *UBDM06 Philadelphia*, Pennsylvania, USA, August 2006.

[9] Y. Liu, W. K. Liao, A. Choudhary, A fast high utility itemsets mining algorithm, *Proc. 1st Intl. Conf. on Utility-Based Data Mining*, Chicago, IL, Aug. 2005 (90–99).

[10] G. Grahne and J. Zhu. Efficiently using prefix-trees in mining frequent itemsets, *Proc. IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, Melbourne, FL, Nov. 2003.

[11] J. Han, J. Pei, and Y. Yin, Mining frequent patterns without candidate generation, *Proc. 2000 ACM-SIGMOD Intl. Conf. on Management of Data*, Dallas, TX, May 2000 (1–12).

[12] M. El-Hajj, Osmar R. Zaiane, Non recursive generation of frequent k-itemsets from frequent pattern tree representations, *Procceding of 5th International Conference on Data Warehousing and Knowledge Discovery,* DaWak, September 2003.

[13] M. El-Hajj và Osmar R. Zaiane, COFI-tree mining: A new approach to pattern growth with reduced candidacy generation,*Procceding 2003 Intl Conf. on Data Mining and Knowledge Discovery (ACM SIGKDD)*, Chicago, Illinois, USA, August 2003.

[14] IBM Synthetic data:
http://www.almaden.ibm.com/software/quest/Resources/index.shtml , 2004.

Nhận bài ngày 5 - 11 - 2008