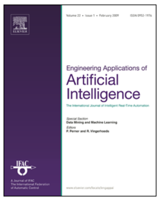
Engineering Applications of Artificial Intelligence 36 (2014) 64–80

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai



Simultaneous mining of frequent closed itemsets and their generators: Foundation and algorithm

Anh Tran [a,*], Tin Truong [a], Bac Le [b]

[a] Department of Mathematics and Computer Science, University of Dalat, Dalat, Vietnam
[b] Department of Computer Science, University of Science, VNU – Ho Chi Minh, Ho Chi Minh City, Vietnam

ARTICLE INFO

Article history:
Received 2 December 2013
Received in revised form
31 May 2014
Accepted 7 July 2014

Keywords:
Closed itemset
Generator
Association rule
Frequent itemset
Data mining

ABSTRACT

Closed itemsets and their generators play an important role in frequent itemset and association rule mining. They allow a lossless representation of all frequent itemsets and association rules and facilitate mining. Some recent approaches discover frequent closed itemsets and generators separately. The Close algorithm mines them simultaneously but it needs to scan the database many times. Based on the properties and relationships of closed itemsets and generators, this study proposes GENCLOSE, an efficient algorithm for mining frequent closed itemsets and generators simultaneously. The level-wise search over an ItemsetObject–setGenerator–Tree enumerates the generators by using a necessary and sufficient condition to produce $(i+1)$-item generators from i-item generators. This condition, based on transaction (object) sets that can be efficiently implemented using diffsets, is very convenient and reliably proved. In the search, pre-closed itemsets are gradually extended using three proposed extension operators. It is shown that these itemsets produce the expected closed itemsets. Extensive experiments on many benchmark databases confirm the efficiency of the proposed approach.

1. Introduction

Association rule mining (Agrawal et al., 1993) from transaction databases is a fundamental technique in data mining. The task is to determine the association rules that satisfy the pre-defined minimum support and confidence from a given database. It was originally designed for market basket applications (Agrawal et al., 1993), but has been extended to various domains, such as risk management, telecommunication networks, and bio-sequences. Association rule mining has two phases (Agrawal and Srikant, 1994): (a) extraction of all *frequent itemsets* whose occurrences exceed the minimum support, and (b) generation of *association rules* that satisfy the given minimum confidence from the itemsets. If all frequent itemsets and their supports are known, association rule generation is straightforward. Hence, most researchers have concentrated on finding efficient methods for mining frequent itemsets.

The basic algorithms for mining frequent itemsets are Apriori, FP-growth, and Eclat. Apriori and its variations (Agrawal et al., 1993; Agrawal and Srikant, 1994) are based on the Apriori property, which states that every subset of a frequent itemset is also frequent, i.e., the *support* of an itemset never exceeds the supports of its subsets. Although this anti-monotone property helps significantly reduce the search space, Apriori-based algorithms are not efficient as they generate many redundant candidates, which increase the CPU and memory burden. Further, they have to scan the database multiple times. To overcome these issues, frequent pattern tree-based algorithms were proposed by Han and Pei (2000) and Han et al. (2004). The original database is compressed into a FP-tree or a similar tree structure. Using divide-and-conquer and depth-first search approaches, all large itemsets are mined from frequent 1-itemsets[1] without having to rescan the database. However, in interactive or incremental mining systems, where the users often change the minimum support and insert new transactions into the original database, FP-tree-inspired structures are unsuitable because the trees need to be rebuilt. Both the Apriori and FP-tree based methods work with a horizontal data format. Zaki proposed Eclat (Zaki, 2000) and DEclat (Zaki and Gouda, 2003) for mining with a vertical data format. These algorithms all show good performance for sparse databases with short itemsets, such as

* Corresponding author. Postal address: 01 Phu dong thien vuong Street, Dalat City, Vietnam. Tel.: +84 983 185 834.
E-mail addresses: anhtn@dlu.edu.vn (A. Tran), tintc@dlu.edu.vn (T. Truong), lhbac@fit.hcmus.edu.vn (B. Le).

[1] Briefly, a set of i items is denoted as i-set, e.g., i-itemset, i-generator.

http://dx.doi.org/10.1016/j.engappai.2014.07.004
0952-1976/© 2014 Elsevier Ltd. All rights reserved.

market databases. For dense databases, which produce long frequent itemsets, such as bio-sequences and telecommunication networks, the frequent itemset class can grow to be unwieldy even if the minimum support is large (Bayardo, 1998). A frequent itemset of length n produces 2^{n-1} frequent non-empty, proper sub-itemsets. Hence, the generation of frequent itemsets not only has the large time complexity $O(2^N)$ (where N is the number of items) but also produces many duplicates in the huge search space. Mining only *maximal frequent itemsets* is one of the solutions for overcoming the drawbacks mentioned above. Many algorithms have been proposed for mining such itemsets (Bayardo, 1998; Burdick et al., 2001). An itemset is maximal frequent if none of its proper supersets are frequent. The number of maximal itemsets is much smaller than that of all frequent itemsets (Zaki and Hsiao, 2005). Although all sub-itemsets of a maximal itemset are frequent, their actual supports are unknown. Further, since frequent itemsets can come from different maximal ones, it takes a lot of time to mine and delete the duplicates. Therefore, maximal frequent itemsets are unsuitable for frequent itemset and association rule generation.

A more suitable approach to overcome this difficulty is using the closures of itemsets, i.e., closed itemsets. The maximal itemset class is contained in the closed itemset class, which, in turn, is a subset of the itemset class. An extensive experimental evaluation conducted by Zaki and Hsiao (2005) showed that, for real databases, the number of frequent closed itemsets is about 10 times greater than that of maximal frequent itemsets, but about 100 times smaller than the cardinality of frequent itemsets. Hence, mining *frequent closed itemsets* has received the attention of many researchers (Pasquier et al., 1999; Pei et al., 2000; Wang et al., 2003; Singh et al., 2005). An itemset is *closed* iff[2] it is identical to its *closure*. This concept is similar to the concept lattice (Birkhoff, 1967; Wille, 1982, 1992; Davey and Priestley, 1994; Ganter and Wille, 1999) and has been recently applied (Boulicaut et al, 2003; Zaki, 2004; Pasquier et al., 2005). A *generator* (Pasquier et al., 1999; Szathmary et al., 2009) of an itemset is its minimal subset that has the same closure as its own. Generators are also called *minimal generators* (Zaki, 2004; Dong et al., 2005), *key patterns* (Bastide et al., 2000), and *free-sets* (Boulicaut et al., 2003). Although there are many definitions of closed itemsets and generators, they are equivalent (see Section 2.1).

Closed itemsets together with their lattice structure and generators, called $\mathcal{LG}_{\mathcal{A}}$, play an important role in both itemset and association rule mining. First, their cardinality is typically orders of magnitude much lower than that of all itemsets. Whenever the user changes the minimum support, all frequently closed itemsets and their generators can be quickly derived from $\mathcal{LG}_{\mathcal{A}}$. Second, two itemsets are equivalent if they have the same closure. In the study by Anh et al. (2012b), based on this *equivalence relation*, all itemsets were partitioned into disjoint equivalence classes. This decreases most duplication in the generation of all itemsets. Further, each class can be explored independently. In each class, a closed itemset is a maximum set, its generators are minimal subsets, and each remaining itemset has a unique representation through its closure and generators. Thus, the duplication in the generation of all frequent itemsets is completely removed. Hence, frequent closed itemsets together with their generators produce a *lossless representation* of all frequent itemsets. Third, many studies have used the generators of closed itemset for mining association rules (Balcazar, 2010; Bay et al., 2012; Anh et al., 2012a; Pasquier et al., 1999, 2005; Zaki, 2004; Tin and Anh, 2010a; Tin et al., 2010b). All rules can be obtained based on frequent closed itemsets and their generators. The lattice of frequent closed itemsets and generators with constraints is essential for the discovery of frequent itemsets and association rules with item constraints, especially when the minimum support and confidence thresholds and item constraints often change (Anh et al., 2011, 2012b, 2014; Hai et al., 2013, 2014).

The problem of mining frequent closed itemsets and generators is stated as follows: given a transaction database and a minimum support threshold, the task is to find all frequent closed itemsets together with their generators. The algorithms for mining closed itemsets can be divided into three approaches, namely generate-and-test, divide-and-conquer and hybrid. Many algorithms have been proposed for mining closed itemsets, including Close (Pasquier et al., 1999) (generate-and-test), Closet (Pei et al., 2000) and Closet+ (Wang et al., 2003) (divide-and-conquer), and Charm (Zaki and Hsiao, 2005) and CloseMiner (Singh et al., 2005) (hybrid). Algorithms for mining generators include Talky-G (Szathmary et al., 2009) and MinimalGenerator (Zaki, 2004). However, most of these algorithms discover frequent closed itemsets and generators separately. The exception is Close, which mines them simultaneously. However, its execution is computationally expensive. The present study proposes GENCLOSE, which has the following key features:

1) Generators and frequent closed itemsets are simultaneously found using *breadth-first search* over an *IOG*-tree (Itemset-Object-set[3]-Generator tree).
2) At each level, the generators are first mined using a *necessary and sufficient condition* to determine the class of $(i+1)$-generators from the class of i-generators $(i \geq 1)$ based on the object-sets (or diffsets in practice).
3) Three *extension operators* are proposed to extend itemsets toward their closures when mining generators.

The rest of this paper is organized as follows. Section 2 gives the background of closed sets, generators, and their definitions. Related work is also discussed in this section. Section 3 proposes some necessary and sufficient conditions for producing generators and three operators for extending generators to their closures. Based on these results, the GENCLOSE algorithm is constructed. In Section 4, GENCLOSE is compared to CharmLMG and DTouch, two well-known algorithms for finding closed itemsets and generators. The conclusion is given in Section 5. For better readability, some proofs and implemented techniques are given in the appendices.

2. Foundation of mining closed itemsets and their generators

Consider two non-empty sets: $\mathcal{O}$ containing objects (or transactions) and $\mathcal{A}$ containing all items (attributes) related to transactions $o \in \mathcal{O}$. Let $\mathcal{R}$ be a binary relation in $\mathcal{O} \times \mathcal{A}$. A triple $\mathcal{D}=(\mathcal{O}, \mathcal{A}, \mathcal{R})$ is called a transaction database or a binary database, or briefly a *database*. A set of items $A \subseteq \mathcal{A}$ and a set $O \subseteq \mathcal{O}$ are called an *itemset* and an *object-set*, respectively. Let $2^{\mathcal{O}}$ and $2^{\mathcal{A}}$ be the power sets of $\mathcal{O}$ and $\mathcal{A}$. Two set functions of λ: $2^{\mathcal{O}} \rightarrow 2^{\mathcal{A}}$ and ρ: $2^{\mathcal{A}} \rightarrow 2^{\mathcal{O}}$ are determined as follows: $\forall A \subseteq \mathcal{A}$, $O \subseteq \mathcal{O}$, $A \neq \varnothing$, $O \neq \varnothing$, $\lambda(O)=\{a \in \mathcal{A} | (o, a) \in \mathcal{R}, \forall o \in O\}$, $\rho(A)=\{o \in \mathcal{O} | (o, a) \in \mathcal{R}, \forall a \in A\}$, and as convention $\rho(\varnothing):=\mathcal{O}$, $\lambda(\varnothing):=\mathcal{A}$. The *closure* of an itemset A is defined by $h(A)=\lambda(\rho(A))$, and that of an object-set O is defined by $h'(O)=\rho(\lambda(O))$. Thus, itemset A is called *closed* iff it is identical to its closure, i.e., $A=h(A)$. Similarly, an object-set O is closed iff $O=h'(O)$ (for details, refer to the studies of Birkhoff (1967), Wille (1982), Wille (1992), Davey and Priestley (1994), and Ganter and Wille (1999)).

[2] For convenience, we write “if and only if” simply as “iff”.

[3] Object-set means set of objects.

The *support* of itemset A, written as supp(A), is the number of objects containing A, i.e., supp(A)=$|\rho(A)|$. For a given *minimum support*, *minsupp* $\in [1;\ |\mathcal{O}|]$, itemset A is *frequent* if supp(A) $\geq$ minsupp (Agrawal et al., 1993). From now on, we always consider all (non-trivial) itemsets A in $\mathcal{A}$: $\rho(A) \neq \varnothing$. Itemset A is called a *frequent closed* itemset if it is not only frequent but also closed. For every two itemsets G and A such that $\varnothing \neq G \subset A$, G is called a *generator* of A iff $h(G)=h(A)$ and $(h(G') \subset h(G), \forall G'\colon\ \varnothing \neq G' \subset G)$ (Pasquier et al., 1999). Let $\mathcal{G}(A)$ be the class of all generators of A. Since it is finite, its elements can be numbered from 1 to $|\mathcal{G}(A)|$ as follows: $\mathcal{G}(A)=\{Gi\colon\ i=1, 2, \ldots, |\mathcal{G}(A)|\}$. For convenience, we denote sets by simple concatenation. For example, we write an itemset $\{a_1, a_2, \ldots, a_n\}$ as $a_1a_2\ldots a_n$. We also write object-set $\{o_1, o_2, \ldots, o_n\}$ simply as $\{1, 2, \ldots, n\}$, where $1, 2, \ldots, n$ are the identifiers of the objects.

Example 1. Let us consider the example database[4] consisting of 6 objects, $\mathcal{O}1=\{1, 2, 3, 4, 5, 6\}$ and $\mathcal{A}_1=abcdefgh$, shown in Table 1. We have $\lambda(\{2, 3\})=adfh$ and $\rho(adfh)=\{2, 3\}$. Thus, $h(adfh)=\lambda(\rho(adfh))=adfh$, i.e., $adfh$ is a closed itemset. This implies that $\{2, 3\}$ is a closed object-set. This is obvious because $h'(\{2, 3\})=\rho(\lambda(\{2, 3\}))=\{2, 3\}$. With minsupp=1,[5] $adfh$ is a frequent itemset since supp($adfh$)=2. Hence, $adfh$ is a frequent closed itemset. Let us consider two subsets of $adfh$, namely d and af. Since $h(d)=adfh=h(adfh)$, d is a generator of $adfh$. Obviously, a sub-itemset of $adfh$ that contains d cannot be a generator of $adfh$. We also find that af is a generator of $adfh$ because $h(af)=h(adfh)$ and $h(a)=ah \subset h(adfh)$, $h(f)=fh \subset h(adfh)$. Hence, its proper subsets, a and f, cannot be generators of $adfh$. The itemsets ah, fh, and h are also not the generators of $adfh$ since their closures are proper subsets of $adfh$. Therefore, we have $(adfh)=\{d, af\}$.

Nineteen frequent closed itemsets are produced from this database. Those sets make the partition of all 145 frequent itemsets included in 35 generators, as shown in Fig. 1. Each class $[C]$, represented by a frequent closed itemset C, contains frequent itemsets of the same closure C. That means that they share the same object-set, and as a result, the same support. In the figure, the support and the object-set are written on the left and right of the corresponding closure, respectively, in superscript. An arc from class $[A]$ to class $[B]$ implies that B is an immediate closed subset of A.

2.1. Preliminaries

The following proposition shows some properties of the Galois operators of ρ, λ, h, and h'. Though they are basic, they are given here as a basis for propositions, consequences, and theorems. Most of them were proven in the studies by Birkhoff (1967), Wille (1992), Davey and Priestley (1994), and Zaki and Hsiao (2005).

Proposition 1. *(Some properties of Galois operators). For* $\forall A,\ A_1,\ A_2 \subseteq \mathcal{A}$ *and* $\forall O,\ O_1,\ O_2 \subseteq \mathcal{O}$, *we have*

1) *The anti-monotone property of* ρ, λ: $A_1 \subseteq A_2 \Rightarrow \rho(A_2) \subseteq \rho(A_1)$; $O_1 \subseteq O_2 \Rightarrow \lambda(O_2) \subseteq \lambda(O_1)$.
2) *The absorption law of* h, h': $A \subseteq h(A)$, $O \subseteq h'(O)$.
3) *The monotone property of* h, h': $A_1 \subseteq A_2 \Rightarrow h(A_1) \subseteq h(A_2)$; $O_1 \subseteq O_2 \Rightarrow h'(O_1) \subseteq h'(O_2)$.
4) *The idempotence property of* h, h': $\rho_o\lambda_o\rho=h'_o\rho=\rho_oh=\rho$, $\lambda_o\rho_o\lambda=h_o\lambda=\lambda_oh'=\lambda$ *and* $h_oh=h$, $h'_oh'=h'$. *As a consequence,* $\rho(A)$ *and* $\lambda(O)$ *are closed object-sets and itemsets for every* $A \subseteq A$, $O \subseteq \mathcal{O}$.

Table 1
An example database.

Object	Identifier	Items							
o_1	1	*a*	*b*	*c*		*e*		*g*	*h*
o_2	2	*a*		*c*	*d*		*f*		*h*
o_3	3	*a*			*d*	*e*	*f*	*g*	*h*
o_4	4		*b*	*c*		*e*	*f*	*g*	*h*
o_5	5		*b*	*c*		*e*			
o_6	6		*b*	*c*					

5) *(a)* $\rho(A_1)=\rho(A_2) \Leftrightarrow h(A_1)=h(A_2)$; *(b)* $\rho(A_1) \subset \rho(A_2) \Leftrightarrow h(A_1) \supset h(A_2)$; *(c)* $\rho(A_1) \subseteq \rho(A_2) \Rightarrow h(A_1)=h(A_1 \cup A_2)$.
6) *The intersection of closed itemsets is closed:* $\rho(\cup_{i \in I}A_i)=\cap_{i \in I}\rho(A_i)$.
7) *The union of closed itemsets may not be closed:* $h(A \cup B)=h(h(A) \cup h(B))$.

In studies concerned with the discovery of closed itemsets and generators, some definitions of closed itemsets and generators have been presented, such as those in the studies of Pasquier et al. (1999), Bastide et al. (2000), Boulicaut et al. (2003), and Szathmary et al. (2009). Their equivalence is shown in Consequences 1 and 2. We have first Remark 1.

Remark 1. For two finite sets $X \subseteq Y$, we have
$(X=Y \Leftrightarrow |X|=|Y|)and(X \subset Y \Leftrightarrow |X|<|Y|)$

Consequence 1. (*The equivalence of the definitions of closed itemsets*). *For* $\varnothing \neq A \subseteq \mathcal{A}$:

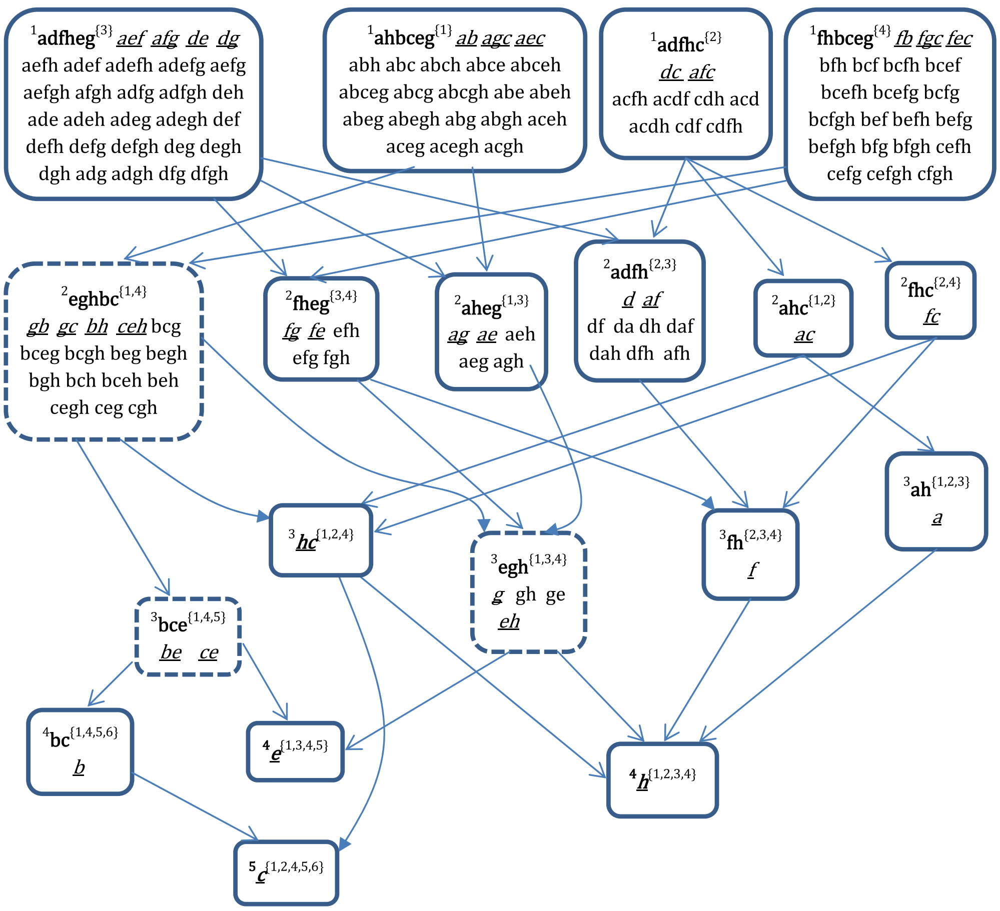
1) *The following three statements are equivalent:*
   (a) *A is a closed itemset (based on operator h),*
   (b) $[\text{supp}(P) < \text{supp}(A),\ \forall P\colon\ A \subset P \subseteq \mathcal{A}]$ *(based on the support of the proper supersets of A, Szathmary et al. (2009)),*
   (c) $[\rho(P) \subset \rho(A),\ \forall P\colon\ A \subset P \subseteq \mathcal{A}]$ *(based on operator* ρ*).*
2) (*a*) *The closure of A is the unique set P such that:* $P \supseteq A$, $h(A)=h(P)$ *(or* $\rho(A)=\rho(P)$ *or* supp(A)=supp(P)) *and for every proper superset Q of P:* $h(Q) \supset h(P)$ *(or* $\rho(Q) \subset \rho(P)$ *or* supp(Q) < supp(P)).
   (b) h(A) is the maximum in the supersets B of A such that:
$$[\text{supp}(A)=\text{supp}(B)\ \text{(Boulicaut et al., 2003) or}\ \rho(A)=\rho(B)\ \text{or}\ h(A)=h(B)] \tag{1}$$
3) *h(A) is the minimum in the class of closed sets containing A.*

Consequence 2. (*The equivalence of the definitions of generators*). *For* $\varnothing \neq G \subseteq A \subseteq \mathcal{A}$:

1) *The following statements are equivalent:*
   (a) $G \in \mathcal{G}(A)$ *(based on operator h),*
   (b) $[\rho(G)=\rho(A)$ *and* $(\rho(G') \supset \rho(G), \forall G'\colon\ \varnothing \neq G' \subset G)]$ *(based on operator* ρ*),*
   (c) [supp(G)=supp(A) *and* (supp(G') > supp(G) *(or* supp(G') $\neq$ supp(G)),$\forall G'\colon\ \varnothing \neq G' \subset G$)] *(based on the support).*
2) *The fact that G is a generator of h(G) is equivalent to the following statements:*
   - $h(G') \subset h(G), \forall G'\colon\ \varnothing \neq G' \subset G$ *(based on h)*
   - $\rho(G') \supset \rho(G), \forall G'\colon\ \varnothing \neq G' \subset G$ *(based on* ρ*)*
   - supp(G') > supp(G),$\forall G'\colon\ \varnothing \neq G' \subset G$ *(based on the support (Szathmary et al. (2009)) and the concepts of key patterns (Bastide et al., 2000) and free-sets (Boulicaut et al., 2003)).*

For non-empty itemset $A \subseteq \mathcal{A}$, the class of the subsets of A, which have the same closure as its own or share the same

[4] This database is used throughout the paper.
[5] From now on, we always consider minsupp=1.



1adfheg{3} aef afg de dg aefh adef adefh adefg aefg aefgh afgh adfg adfgh deh ade adeh adeg adegh def defh defg defgh deg degh dgh adg adgh dfg dfgh
1ahbceg{1} ab agc aec abh abc abch abce abceh abceg abcg abcgh abe abeh abeg abegh abg abgh aceh aceg acegh acgh
1adfhc{2} dc afc acfh acdf cdh acd acdh cdf cdfh
1fhbceg{4} fb fgc fec bfh bcf bcfh bcef bcefh bcefg bcfg bcfgh bef befh befg befgh bfg bfgh cefh cefg cefgh cfgh
2eghbc{1,4} gb gc bh ceh bcg bceg bcgh beg begh bgh bch bceh beh cegh ceg cgh
2fheg{3,4} fg fe efh efg fgh
2aheg{1,3} ag ae aeh aeg agh
2adfh{2,3} d af df da dh daf dah dfh afh
2ahc{1,2} ac
2fhc{2,4} fc
3ah{1,2,3} a
3hc{1,2,4}
3egh{1,3,4} g gh ge eh
3fh{2,3,4} f
3bce{1,4,5} be ce
4bc{1,4,5,6} b
4e{1,3,4,5}
4h{1,2,3,4}
5c{1,2,4,5,6}

Fig. 1. Frequent closed (bold) itemsets and generators (underlined, italicized) produced from example database.

object-set ($\rho(A)$), is written as $\lfloor A \rfloor$. Formally, $\lfloor A \rfloor = \{I\colon \varnothing \neq I \subseteq A, h(I) = h(A)\}$. Following the definition of generators, we have the following proposition.

Proposition 2. (Some features of generators). For every non-empty itemset $A \subseteq \mathcal{A}$, we have

1) *There exists a generator of A, i.e.,* $\mathcal{G}(A) \neq \varnothing$,
2) $\mathcal{G}(A) \subseteq \mathcal{G}(B)$, $\forall B \subseteq \mathcal{A}$ *such that* $A \subseteq B$ *and* $h(A) = h(B)$,
3) *G is a generator of A iff it is a minimal element in* $\lfloor A \rfloor$.

The Apriori property, which states that the subsets of a frequent itemset are also frequent, has been used in many mining algorithms. Surprisingly, the generators also have this property. In other words, every non-empty, proper subset G' of a generator G is also a generator of $h(G')$ (Pasquier et al., 2005). For instance, from Fig. 1, *ceh* is a generator (of closed itemset *bcegh*) and the generators of *ce*, *eh*, *hc*, *c*, *e*, and *h* (with respect to closed itemsets *bce*, *egh*, *hc*, *c*, *e*, and *h*) are just its non-empty, proper subsets.

2.2. Related work

2.2.1. Closed itemset mining

The algorithms for mining closed itemsets can be divided into three approaches, namely generate-and-test, divide-and-conquer, and hybrid.

The first includes Apriori-inspired algorithms. Close, proposed by Pasquier et al. (1999), executes a level-wise process. Each step has two phases. In the first phase, the candidates (their number is usually big) for generators are generated by joining generators found in the previous step. If the support of a candidate equals that of its subset, the candidate cannot be a generator. In the second phase, the closures of generators are computed. The ones whose occurrences exceed the minimum support are frequent closed itemsets. Unfortunately, because of the need to perform a large number of transaction intersection operations, the algorithm is very computationally expensive.

The divide-and-conquer strategy searches over tree structures similar to FP-tree. Closet (Pei et al., 2000) and Closet+ (Wang et al., 2003) are typical examples. Closet, an extension of FP-Growth (Han and Pei, 2000), applies single prefix path compression to dramatically reduce the search space for identifying frequent closed itemsets. The database projection approach, based on partitioning, is also used to improve mining. Closet+ integrates the advantages of the strategies proposed in Closet with the depth-first searching paradigm and top-down pseudo tree-projection. However, the maintenance of the global FP-tree for keeping track of frequent closed itemsets makes the search slow.

The hybrid approach combines the other two approaches. An example is Charm (Zaki and Hsiao, 2005). It executes an efficient hybrid search that skips many levels of the IT-tree to reach frequent closed itemsets, instead of having to enumerate many possible subsets. A hash-based approach is applied to speed up subsumption checking. To compute frequency quickly as well as to reduce the size of intermediate tidsets (sets of transaction identifiers or object-identifiers), the diffset technique is used. Based on Charm, Singh et al. (2005) proposed CloseMiner, which transforms the problem of mining closed itemsets into a problem of clustering the itemsets with closed tidsets. Experiments conducted by Zaki and Hsiao (2005) showed that, for mining closed itemsets, Charm outperforms the existing algorithms on many databases.

2.2.2. Generator mining

Very few studies have focused on mining the generators of frequent closed itemsets. SSMG-Miner, developed by Dong et al. (2005), is based on a depth-first search. While mining non-redundant generators, it does not output the closed itemsets. Further, the database is accessed to generate local generators. Boulicaut et al. (2003) proposed MineEX to generate frequent free-sets (generators). Unfortunately, the algorithm has to scan the database at each step for testing whether a candidate is free. Recently, based on Eclat, Szathmary et al. (2009) proposed Talky-*G*, an efficient algorithm for mining generators.

2.2.3. Hybrid approaches for mining closed itemsets and their generators

In a post-processing approach, Szathmary et al. (2009) applied Charm for mining closed itemsets according to generators mined by Talky-G, and then grouped the generators of a given closed itemset. This combination is called the Touch algorithm. Very recently, Hashem et al. (2014) used a modification of Charm-L for mining cross-level frequent closed itemsets. During the mining, MG-Charm (proposed by Vo and Le (2009)) is applied to identify generators. A different approach, called CharmLMG, includes two phases. In the first phase, the closed itemset lattice is determined using Charm-L (Zaki and Hsiao, 2005), whose execution time is greater than that of Charm but only by a negligible amount. In the second phase, MinimalGenerator (proposed by Zaki (2004)) is used for generating all generators of each frequent closed itemset based on only its immediate sub-itemsets. A previous version of this paper with the simultaneous mining of both frequent closed itemsets and their generators was also published (Anh et al., 2013).

3. GENCLOSE: background and algorithm

Based on the foundation of mining closed itemsets and their generators given in Section 2, we propose GENCLOSE, which executes a *breadth-first search* over an IOG-tree to find generators. Simultaneously, the closed itemsets (their closures) are gradually explored. In each step, the algorithm tests an *efficient necessary and sufficient condition* in order to produce new generators from the generators mined in the previous step. Unlike Close, which needs to scan the database to compute the closed itemsets, *three extension operators* are proposed to discover the closures during generator mining.

Itemset-Object-set-Generator tree. An IOG-tree node includes four fields. The first, called *generator set* (GS), contains generators. The second, called *pre-closed itemset* (or briefly *H*), is the itemset that shares the same object-set with the generators. In the execution of GENCLOSE, H is gradually enlarged to its closure, $h(H)$. The third, called *O*, contains the *objects* (in the implementation, their differences are stored). It is easy to see that $H \subseteq \lambda(O) = h(G)$ and $\rho(H) = O = \rho(G)$, $\forall G \in GS$. The last, called Supp, stores the *cardinality* of *O*, i.e., supp(*H*). Starting from the root, the IOG-tree is extended level by level: $L[1]$, $L[2]$, etc. The first level, $L[1]$, initialized from the database, contains the nodes of 1-generators. The *i*th level $L[i]$, with $i > 1$, contains the nodes of *i*-generators obtained from the previous level, $L[i-1]$.

Main steps. For a given transaction database $\mathcal{D}$ and minsupp, the task of GENCLOSE is to determine $\mathcal{LCG}$, the list of all frequent closed itemsets together with their generators and supports $\langle h(P) \rangle = (\text{supp}(P),\ h_P \equiv h(P),\ GS \equiv \mathcal{G}(h(P)))$ or $({}^{\text{supp}(P)}h_P \equiv h(P),\ GS \equiv \mathcal{G}(h(P)))$. We start by eliminating non-frequent items from $\mathcal{A}$ and using the remaining ones to initialize $L[1]$. More concretely, $L[1]$ includes the nodes with respect to frequent 1-generators in the form: $(x) = \langle |\rho(x)|, h_x := x, \{x\}, \rho(x) \rangle$ or $\langle {}^{|\rho(x)|}\{x\}^{\rho(x)}, h_x \rangle$. In the first step, we extend each h_x to its closure $h(x)$ for each (x) and initialize $\mathcal{LCG}$ by 1-generators together with their closures and supports. Starting with $i = 2$, the *i*th step of the algorithm, which includes producing new generators and extending pre-closed itemsets, is broken into three phases as follows:

- *At first, the* $(i\text{–}1)$*-generators in the nodes at* $L[i\text{–}1]$ *are joined to produce i-generators. The pre-closed itemsets with respect to new i-generators are initialized and extended.*
- *The second phase extends the pre-closed itemsets of the nodes at L* $[i]$ *and merges those from the same object-set.*
- *The last phase checks to see if the pre-closed itemsets with respect to i-generators can be extended to old closed itemsets, i.e., if they are in* $\mathcal{LCG}$*. If yes, we update their generator lists as well as the pre-closed itemsets of the nodes containing those generators. If no, i.e., they are new closed itemsets, then we insert them together with the corresponding generators into* $\mathcal{LCG}$*.*

To produce new generators, GENCLOSE uses the necessary and sufficient condition (5) whose correctness and efficiency is shown in Section 3.1. Three extension operators, proposed in Section 3.2, are used to implement the task of extending pre-closed itemsets toward their closures. Theorem 2 proves that using these operators produces the closed itemsets.

3.1. Necessary and sufficient conditions to produce generators

The special case of the Apriori property of generators implies that a generator can be created from its sub-generators. Thus, *i*-generators can be generated by joining $(i\text{–}1)$-generators. For example, we can join two generators, *ce* and *ch*, to obtain the generator *ceh* (appears in the works of Vo and Le (2009) and Hashem et al. (2014)). However, the union of two generators may not be a generator. For instance, although $G_1 = eb$ and $G_2 = eh$ are two generators (of *bce* and *egh*, respectively), $G = G_1 \cup G_2 = ebh$ is not a generator. The reason is that there exists a proper subset *bh* of *ebh* with the same closure *eghbc* (see classes [*bce*], [*egh*], and [*eghbc*], represented by dashed squares in Fig. 1). Hence, it is necessary to execute a check to determine whether two-generator union *G* is a generator.

In the execution of Talky-G (Szathmary et al., 2009), the union *G* of two generators G_1 andG_2 needs to pass a necessary test, which states that $\rho(G) \neq \rho(G_1)$ and $\rho(G) \neq \rho(G_2)$, to be a generator. This condition is satisfied for both *ceh* and *ebh*: $\rho(ce) = \{1, 4, 5\} \neq \rho(ceh) = \{1, 4\} \neq \rho(ch) = \{1, 2, 4\}$, $\rho(eh) = \{1, 3, 4\} \neq \rho(ebh) = \{1, 4\} \neq \rho(eb) = \{1, 4, 5\}$. However, as we known, *ceh* is a generator but *beh* is not. To ensure that *G* is a generator, Talky-G checks if it has a proper subset G' with the same support. This is a sufficient condition, which follows directly from Consequence 2.2. As all proper subsets of *G* were discovered using reverse pre-order (from right-to-left) depth-first traversal, the check can be executed. However, it can be a very computationally expensive operation since, in the worst case, all proper subsets of *G* need to be visited. For illustration, *beh* is not a generator as supp $(bh) = |\rho(bh)| = |\{1, 4\}| = 2 = \text{supp}(ebh)$. It would be a waste of time to access the supports of the remaining subsets of ebh: *b*, *e*, *h*, *eb*, and *eh*.

For doing this check, Close (proposed by Pasquier et al. (1999)) computes the closures of $(i\text{–}1)$-subsets of *G*. Based on the necessary condition for generators shown in Corollary 2 (of that paper), a test on those closures is required to decide whether *G* is a generator. More concretely, if there exists a $(i\text{–}1)$-subset G' of *G* such that $G \subseteq h(G')$, then *G* is not a generator (see Remark 2 below). For example, *beh* is not a generator since $beh \subset h(bh) =$ eghbc. It is important to note that using this corollary requires a lot of computation time. The computation of the closures requires at least a database pass, and the set inclusion operators used in the test are computationally expensive. More efficient necessary and sufficient conditions based on the supports of G and G' only ($\forall G'$: $\varnothing \subset G' \subset G$ and $|G'| = |G| - 1$) are given in Theorem 1.

Every 1-itemset $a \in \mathcal{A}$ is a 1-generator. For any $i \geq 2$, from two $(i-1)$-generators (in $L[i-1]$) of G_1 and G_2 with $i-2$ common items: $|G_1 \cap G_2| = i-2$, by joining them $G_1 \cup G_2$, we obtain a new candidate i-generator $G = G_1 \cup G_2$. When is G a generator? Theorem 1 gives the necessary and sufficient conditions to recognize if any itemset is a generator. Condition (5) only uses simple checks on the supports of immediate. Itemset G in this general theorem can be the result of joining two generators that satisfy the normal properties as well as the union of two different arbitrary generators.

Theorem 1. *(Necessary and sufficient conditions to recognize generators). For $\varnothing \neq G \subseteq \mathcal{A}$: $|G| \geq 2$, $G_g := G \backslash \{g\}$, $g \in G$. The following conditions are equivalent:*

$$.G \text{ is a generator of } h(G) \tag{2}$$

$$.\rho(G) \notin \cup_{g \in G} \rho(G_g) \tag{3}$$

$$.\text{not}(\rho(G_g) \subseteq \rho(\{g\})), \forall g \in G \tag{4}$$

$$.|\rho(G)| < |\rho(G_g)| \ (\text{i.e., } supp(G) < supp(G_g)), \forall g \in G \tag{5}$$

$$.h(G_g) \subset h(G), \forall g \in G \tag{6}$$

$$.|h(G_g)| < |h(G)|, \forall g \in G \tag{7}$$

$$.\text{not}(G \subseteq h(G_g)), \forall g \in G \tag{8}$$

$$.g \notin h(G_g), \forall g \in G \tag{9}$$

Proof. $+(2) \Leftrightarrow (3)$: "$\Rightarrow$": if G is a generator of $h(G)$, then for every $\varnothing \neq G' \subset G$, $\rho(G') \supset \rho(G)$. This implies that $\rho(G) \subset \rho(G_g)$ $\forall g \in G$. Hence, $\rho(G) \neq \rho(G_g)$, $\forall g \in G$.

"$\Leftarrow$": on the contrary, suppose that G is not a generator of $h(G)$. Then: $\exists G' \subset G$: $\rho(G') = \rho(G)$, $G' \neq \varnothing$. Based on the anti-monotone property of ρ, we can assume that $|G \backslash G'| = 1$ and $G' = G_g$, with $g \in G$. Thus, $\rho(G_g) = \rho(G)$. This contradicts (3).

$+(3) \Leftrightarrow (4)$: the fact we need to prove is equivalent to $[\rho(G) \in \cup_{g \in G} \rho(G_g) \Leftrightarrow \exists g \in G$: $\rho(G_g) \subseteq \rho(\{g\})] \equiv [\exists g \in G$: $\rho(G) = \rho(G_g) \Leftrightarrow \exists g \in G$: $\rho(G_g) \subseteq \rho(\{g\})]$. This is obvious from property 6 of Proposition 1:

$$\rho(G) = \rho(G_g) \cap \rho(\{g\}) = \rho(G_g) \Leftrightarrow \rho(G_g) \subseteq \rho(\{g\}).$$

$+(3) \Leftrightarrow (5) \Leftrightarrow (7)$: based on Proposition 1.1, we have $\rho(G) \subseteq \rho(G_g) \Leftrightarrow h(G) \supseteq h(G_g)$, $|\rho(G)| \leq |\rho(G_g)|$, $|h(G)| \geq |h(G_g)|$, $\forall g \in G$. Then $\rho(G) = \rho(G_g) \Leftrightarrow |\rho(G)| = |\rho(G_g)| \Leftrightarrow |h(G)| = |h(G_g)|$, i.e., not(3) $\Leftrightarrow$ not(5) $\Leftrightarrow$ not(7).

$+(3) \Leftrightarrow (6) \Leftrightarrow (8) \Leftrightarrow (9)$: indeed,

not(3) $\Leftrightarrow h(G) \in \cup_{g \in G} h(G_g)$ (Proposition 1.5) $\Leftrightarrow \exists g \in G$: $h(G) = h(G_g)$ (not(6))

$$\Leftrightarrow \exists g \in G : G \subseteq h(G_g)(\text{not}(8)) \Leftrightarrow \exists g \in G : g \in h(G_g)(\text{not}(9)) \tag{10}$$

The part "$\Rightarrow$" of (10) holds since $g \in G \subseteq h(G_g)$. Conversely, if $g \in h(G_g)$, then by Proposition 1.2, $G = G_g \cup \{g\} \subseteq h(G_g)$. Thus, (3) $\Leftrightarrow$ (6) $\Leftrightarrow$ (8) $\Leftrightarrow$ (9). □

Remark 2. Let us review Corollary 2 in Pasquier et al. (1999). We find that the conclusion $h(I) = h(s_a)$ is not true because of the assumption that I is an i-generator and $\varnothing \neq s_a \subset I$. It is thus corrected as follows:

Let I be an i-itemset and $S = \{s_1, s_2, \ldots, s_j\}$ a set of $(i-1)$-subsets of I where $\cup_{s \in S} s = I$.

If$\exists s_a \in S$ such that $I \subseteq h(s_a)$, i.e., $h(I) = h(s_a)$, then I is not a generator.

The contrapositive statement of Corollary 2 is a necessary condition to determine whether I is a generator. Using the same idea but more general, (6) is a necessary and sufficient condition. The conditions (6)–(9) are based on closed itemsets. However, at the time of discovering the generators, those closed itemsets have not been determined yet. It is not easy to compute them because accessing the database to take the intersection of transactions is time-consuming. Hence, it is suitable to use object-sets, which can be efficiently saved and computed using the diffset technique (Zaki and Gouda, 2003; Zaki and Hsiao, 2005). Among the remaining conditions, (5) seems to be the best choice because the task of checking the inclusions in (3) and (4) requires a lot of time compared to the cardinality computation (especially for databases in which object-sets can considerably grow).

Example 2. We find from Fig. 1 that $G = agc$ with $\rho(G) = \{1\}$ is a generator. Since $\rho(G_c) = \rho(ag) = \{1, 3\}$, $\rho(c) = \{1, 2, 4, 5, 6\}$, $\rho(G_g) = \rho(ac) = \{1, 2\}$, $\rho(g) = \{1, 3, 4\}$, $\rho(G_a) = \rho(gc) = \{1, 4\}$, and $\rho(a) = \{1, 2, 3\}$, all of (3)–(5) are satisfied. The conditions from (6)–(9) are also satisfied as $h(agc) = abcegh$, $h(ag) = aegh$, $h(ac) = ach$, and $h(gc) = bcegh$. In all those tests, it is clear that the one for (5) is the most simple: $\text{supp}(G) < \text{supp}(G_g)$, $\forall g \in G$. Let us consider $I = beh$. Although $\text{supp}(I) < \text{supp}(be)$ and $\text{supp}(I) < \text{supp}(eh)$, I is still not a generator since $\exists e \in I$, $I_e = bh$: $\text{supp}(I_e) = \text{supp}(I) = 2$ (see Fig. 2 for an illustration).

Condition (5) is applied in GENCLOSE to mine generators in a breadth first manner as follows. First, we get from the database frequent 1-generators together with the corresponding object-sets. Then, in the ith step ($i \geq 2$), we join (denoted by □) each $(i-1)$-generator G_1 with G_2 in the same level $L[i-1]$ to generate new i-generator candidate $G = G_1 □ G_2$. The corresponding object-set $\rho(G)$ is computed as follows: $\rho(G) = \rho(G_1) \cap \rho(G_2)$. The candidates that pass the frequency check are tested using condition (5). Fig. 3 shows an important part of the IOG-tree (which contains all 35 generators) used by GENCLOSE for mining generators, where generators are in enclosed regions. For convenience, pre-closed itemsets are omitted.

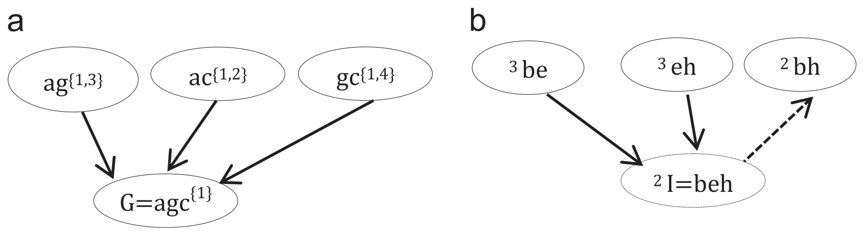


Fig. 2. Illustration of using condition (5).

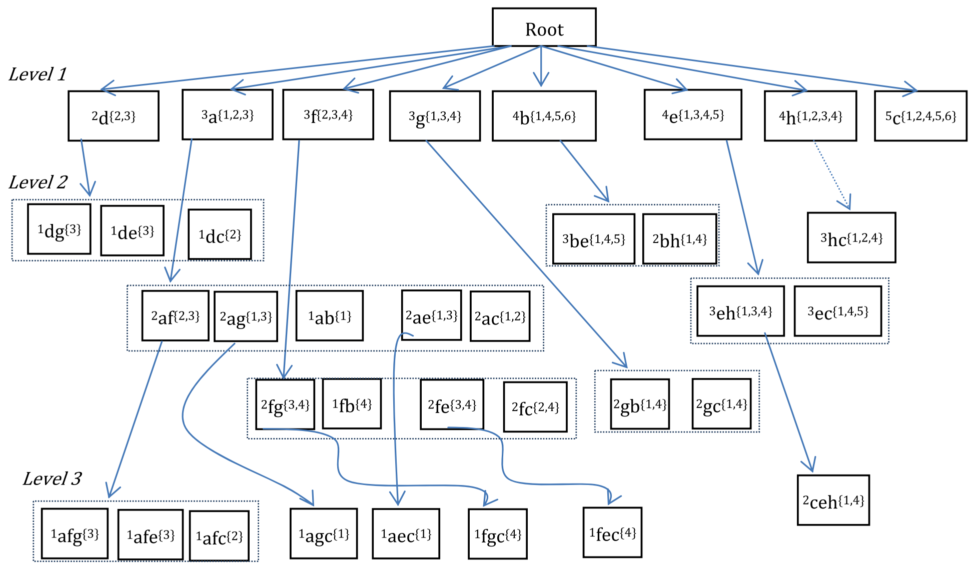


Fig. 3. Illustration of mining generators by GENCLOSE.

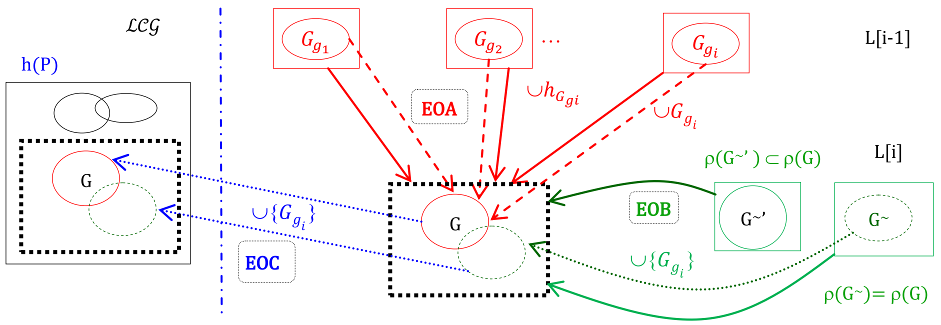


Fig. 4. Workflow of extension operators.

3.2. *Three operators for extending pre-closed itemsets toward their closures*

As mentioned in the beginning of this section, we attach pre-closed itemsets to generators for finding their closures during the generator mining process. Recall that pre-closed itemset hG of generator G shares the same object-set with that generator, i.e., $G \subseteq hG \subseteq h(G)$. During the process of finding generators G, we need to gradually extend h_G by subitemsets to discover the corresponding closure $h(G)$. The following two remarks are made:

First, for G, $G^\sim \subseteq \mathcal{A}$:

$$\text{if } \rho(G) \subseteq \rho(G^\sim), \quad \text{then } h(G) \supseteq h(G^\sim) \tag{11}$$

That is, we can add $h_{G^\sim}$ to h_G. Assume that the process of the extension, which uses (11), is currently at G. Thus, we need to consider the combinations of G with itemsets $G^{\sim 1}$ and $G^{\sim 2}$ so that $|G^{\sim 1}| \le |G| < |G^{\sim 2}|$. Without loss of generality, we only take care of itemsets $G^{\sim 1}$ as the combination of G with $G^{\sim 2}$ is considered when the process is at $G^{\sim 2}$. Recursively applying this remark on the two generators of G and $G^\sim$ which are in one or two adjoining levels ($i-1 \le |G|$, $|G^\sim| \le i$), we have the following extension operators, EOA and EOB, for h_G:

– Operator EOA is used to extend pre-closed itemsets h_G (at level i) based on $(i-1)$-generators $G_g = G \backslash \{g\}$, $\forall g \in G$ (used when joining two adjoining levels):

  $h_G \leftarrow h_G \cup (\cup_{g\in G} h_{G_g})$

– Operator EOB is used to extend pre-closed itemsets h_G based on i-generators $G^\sim$ of the same level i:

  $h_G \leftarrow h_G \cup (\cup_{G^\sim \in \mathcal{G}(h_{G^\sim}):\ |G^\sim| = |G|,\ O_G \subseteq O_{G^\sim}} h_{G^\sim})$

  At level 1, $L[1]$ contains 1-generators $a \in \mathcal{A}$. To identify the corresponding closed itemsets $h(a)$ from pre-closed itemsets h_a, we first assign $h_a \leftarrow \{a\}$ and then extend h_a by h_b: $h_a \leftarrow h_a \cup h_b$ for $\forall b \in \mathcal{A}$ such that $\rho(b) \supseteq \rho(a)$. Using extension operator EOB, we have $h_a \equiv h(a)$. In Fig. 4, at $L[1]$, after assigning: $h_a \leftarrow \{a\}$ with $a \in \mathcal{A}$, we have closed itemsets: *e*, *h*, and *c* (italicized). Then, EOB is applied to extend pre-closed itemsets *d*, *a*, *f*, *g*, and *b*. Thus, we have closed itemsets *dafh*, *ah*, *fh*, *geh*, and *bc*. For example, because $\rho(c) = \{1, 2, 4, 5, 6\} \supset \rho(b) = \{1, 4, 5, 6\}$, we reach closed itemset $h_b = bc = h(b)$ after the extension $h_b \leftarrow h_b \cup h_c$.

At level i ($i \ge 2$), $L[i]$ contains i-generators G, produced from the joining of $(i-1)$-generators G_g. First, we need to use EOA for extending h_G. For instance, with two nodes $(G_1) = \langle {}^{2}\{d\}^{\{2,3\}}, dafh\rangle$, $(G_2^\sim) = \langle {}^{3}\{g\}^{\{1,3,4\}}, ghe\rangle$, we have $(G^\sim) = (G^1 \square G_2^\sim) = \langle {}^{1}\{dg\}^{\{3\}}, dgafhe\rangle$ where $dgafhe = h(G^\sim)$ is closed. However, the output of this operator can be still a non-closed itemset. In other words, its extension is still not enough; we need the different extension operators. With $(G_1) = \langle {}^{2}\{d\}^{\{2,3\}}, dafh\rangle$, $(G_2) = \langle {}^{4}\{e\}^{\{1,3,4,5\}}, e\rangle$, we have $(G) = (G_1 \square G_2) = \langle {}^{1}\{de\}^{\{3\}}, deafh\rangle$ where $h_G = deafh$ $(\subset h(G) = deafhg)$ is not enough to be a closed itemset. Additionally using extension operator EOB, we can reach the closed itemset $h(G)$ from h_G. Let us consider the following case. Since $\rho(G)=\rho(G^{\sim})$, we get closed itemset deafhg from the extension $h_G \leftarrow h_G \cup h_{G\sim}$. Further, we merge node G to node $G^{\sim}$ (of the same closure) and add generator G to the current generator list of $G^{\sim}$. Thus, $(G^{\sim})=\langle{}^{1}\{dg, de\}^{\{3\}}, dgafhe\rangle$.

Second, the extension of h_G using both EOA and EOB may not be enough (i.e. after extension, h_G is still not equal to $h(G)$). In fact, for $(G_3)=\langle{}^{3}\{a\}^{\{1,2,3\}}, ah\rangle$, $(G_5)=\langle{}^{3}\{f\}^{\{2,3,4\}}, fh\rangle$, using EOA, pre-closed itemset *afh* of $(G_3\square G_5)=\langle{}^{2}\{af\}^{\{2,3\}}, afh\rangle$ is not closed. With this node, there is no 2-generator $G^{\sim}$ at the same level $L[2]$ that satisfies the condition of applying EOB. Note that $J=afhd$ has two generators of two different lengths: $\mathcal{G}(J)=\{af, d\}$. To extend the pre-closed itemset, we notice that closed itemset $h(d)=dafh$ of 1-generator d is in lattice $\mathcal{LCG}$. Hence, we need to check if the current $\mathcal{LCG}$ (which includes the nodes made by generators with sizes of 1, 2, …, i–1) contains closed node $\langle P\rangle$ with $P=h(P)$ such that $h(P)\supseteq h_G$ and $\text{supp}(P)=\text{supp}(h_G)$. If not, h_G will be a new closed itemset of $\mathcal{LCG}$. Otherwise, $h(G)=h(P)$. Thus, operator EOC is used in three steps: $h_G\leftarrow h(P)$, merge node $\langle h(G)\rangle$ to node $\langle P\rangle$, and add generator G to the generator list of P: $\mathcal{G}(P)\leftarrow\mathcal{G}(P)\cup\{G\}$. For the current example, since $\langle P\rangle=\langle h(d)\rangle$: $h(d)=dafh\supset afh=h_G$ and $\text{supp}(d)=\text{supp}(af)=2$, we assign $h_G\leftarrow h(d)=afhd$ and $\mathcal{G}(dafh)=\{d, af\}$. Thus, $\langle P\rangle=(2, dafh, \{d, af\})$.

Based on the above discussions, we propose three operators, EOA, EOB, and EOC, for gradually extending the pre-closed itemsets toward their closures (their workflow is shown in Fig. 4). *Let G be an i-generator created by joining two nodes at L[i–1] and $\mathcal{LCG}$ be the set of all closed nodes* $\langle P\rangle=(\text{supp}(P), h(P), \mathcal{G}(P))$, *where* $P=h(P)$ *and* $|G'|\le i-1$, $\forall G'\in\mathcal{G}(P)$:

EOA. $\forall i\ge 2$, h_G is formed by $(i-1)$-generators: $G_g=G\backslash\{g\}$, $\forall g\in G$: $h_G\leftarrow h_G\cup(\cup_{g\in G}h_{G_g})$.
EOB. $\forall i\ge 1$, h_G is extended by:
$h_G\leftarrow h_G\cup(\cup_{G\sim\in\mathcal{G}(h_{G\sim}):\ |G\sim|=i,\ O_G\subseteq O_{G\sim}}h_{G\sim})$.

EOC. $\forall i\ge 1$, for new entry $\langle h(G)\rangle=(\text{supp}(G), h_G, \mathcal{G}(h(G)))$, we check to see if there exists a closed node $\langle P\rangle$ in $\mathcal{LCG}$ such that:

$\text{supp}(P)=\text{supp}(h_G)$ and $P\supseteq h_G$.

1) *If yes, h_G is not a new closed itemset. Thus, we update old node $\langle P\rangle$ by adding i-generators in $\mathcal{G}(h(G))$ to $\mathcal{G}(P)$ of $\langle P\rangle$: $\mathcal{G}(P)\leftarrow\mathcal{G}(P)\cup\mathcal{G}(h(G))$.*
2) *Otherwise, we insert new closed itemset h_G together with its i-generators and support into $\mathcal{LCG}$.*
Note that for $i=1$, we only apply EOB and EOC.

Some cases of using three operators EOA, EOB, and EOC for finding closed itemsets were shown above. Fig. 5 illustrates all remaining cases, where dashed arrows, dashed dot arrows, and dashed doubledot arrows show the extensions obtained using EOB, EOA, and EOC, respectively. After we join (G_3) with (G_4) and apply operator EOA, we have (G') with $h_{G'}=ahbc$ is not closed. We need to additionally use EOB to extend $h_{G'}$ by $h_{G\sim\sim}=ge$ contained in $(G^{\sim\sim})$ (the result of merging the nodes $(G_3\square G_2^{\sim})$ and $(G_3\square G_2)$). Thus, we get closed itemset $h(G')=ahbcge$ (as $\rho(G^{\sim\sim})\supseteq\rho(G')$). When joining $(G^{\sim\sim})=\langle{}^{2}\{ag, ae\}^{\{1,3\}}, ahge\rangle$ with $(G''')=\langle{}^{2}ac^{\{1,2\}}, ahc\rangle$, we get new generator *agc* and corresponding pre-closed itemset *ahgec*. Since $gc=agc\backslash a$ is a generator in node $G''=\langle{}^{2}\{gb, gc, bh\}^{\{1,4\}}, gehbc\rangle$, operator EOA extends ahgec by *b*. The output is just closed itemset $h(agc)=$ahgecb. In addition, we show an example which illustrates the case where we meet closed node $[P]$ such that: $P=h_G$ (and $\text{supp}(P)=\text{supp}(h_G)$). For instance, after joining $(G_1')=\langle{}^{2}\{af, d\}^{\{2,3\}}, afhd\rangle$ with $(G^{\sim\sim})=\langle{}^{2}\{ag, ae\}^{\{1,3\}}, ageh\rangle$ and using EOA, we have $(G^{\sim\sim\sim})=(G_1'\square G^{\sim\sim})=\langle{}^{1}\{afg, afe\}^{\{3\}}, afgehd\rangle$ where $h_G^{\sim\sim\sim}=afgehd=h(G^{\sim\sim\sim})$ is closed. This closed itemset is identical to the old one in node $\langle P\rangle=\langle h(dg)\rangle=(1, edgfah, \{dg, de\})$ of the current lattice $\mathcal{LCG}$ (containing

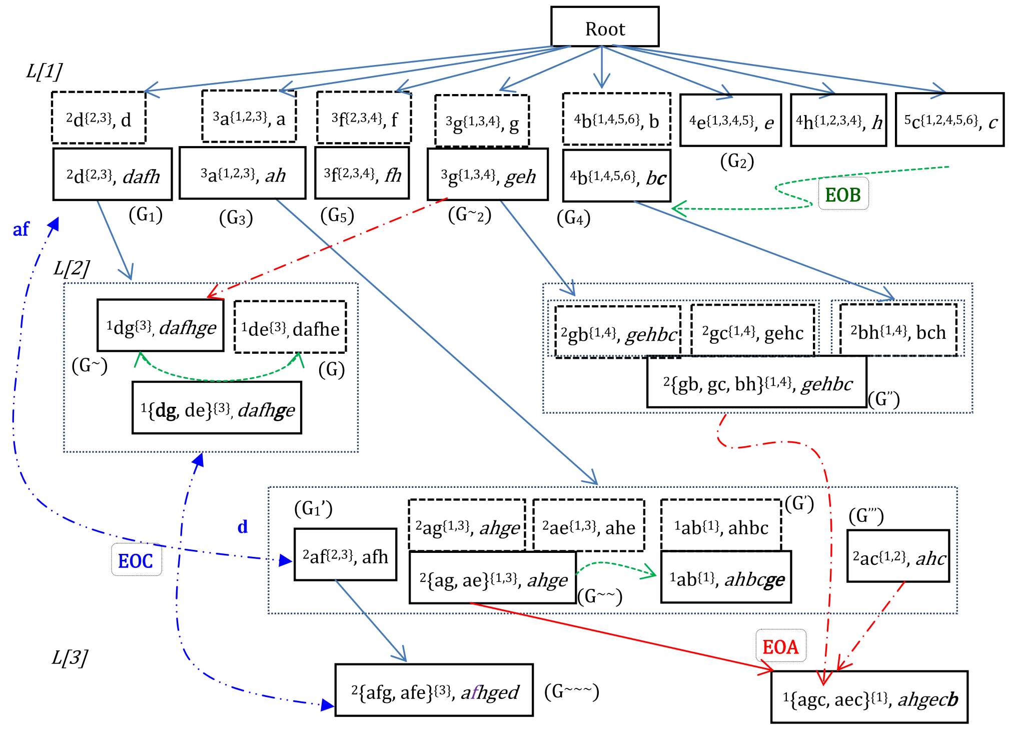


Fig. 5. Illustration of using extension operators EOA, EOB, and EOC in GENCLOSE.

the nodes with generators of sizes 1 and 2): $h(dg)=edafgh=h_G$ (and $\text{supp}(P)=\text{supp}(h_{\tilde{G}}{}^{\sim\sim})=1$). Thus, after updating the node, we have: $\langle P\rangle=(1, edgfah, \{dg, de, afg, afe\})$.

As above, we need the operators EOA, EOB, and EOC to extend pre-closed itemsets. However, it may be asked whether other operators are necessary. The answer is given in Theorem 2, which proves that the three operators produce the desired closed itemsets.

Theorem 2. *(Correctness of extension operators). After we use the operators on every node at level i containing i-generators, h_G is closed.*

Proof. *Case $i=1$:* First, $\forall b\in\mathcal{A}$, assign $h_b=b$. Then, $h(b)=\{a\in\mathcal{A}: a\in h(b)\}=\{a\in\mathcal{A}: h(a)\subseteq h(b)\}=\{a\in\mathcal{A}: O_a\equiv\rho(a)\supseteq O_b\equiv\rho(b)\}=h_b$, using EOB. Hence, $h(b)=h_b$.

Case $i>1$: Suppose that the conclusion is true for 1, 2, …, i–1. After EOA and EOB are used, we have

$$h_G=(\cup_{g\in G}h_{G_g})\cup(\cup_{G\sim\ \in\mathcal{G}(h_{G\sim}):\ |G\sim|=i,\ O_G\subseteq O_{G\sim}}h_{G\sim})$$

If h_G is closed and identical to a closed itemset $P=h(P)$ of node $\langle P\rangle$ on $\mathcal{LCG}$ (i.e., $h_G=P$), we just add i-generators in GS to $\mathcal{G}(P)$ of $\langle P\rangle$. If h_G is a new closed itemset, we insert it together with its i-generators and support into $\mathcal{LCG}$. Let us assume that h_G is not yet closed, i.e., $h_G\subset h(G)$. Consider any:

$$a\in h(G)\setminus h_G \tag{12}$$

First, we prove that aG_g is not an i-generator of $h(aG_g)$. Indeed, assume that the contrary happens. As $a\in h(G)$, $h(G)=h(aG)$ and $\rho(G)=\rho(aG)\subseteq\rho(aG_g)$. Using operator EOA for h_{aG_g} and operator EOB for the generators G and aG_g, we have $a\in h_a\subseteq h_{aG_g}\subseteq h_G$. This implies that $a\in h_G$, which contradicts (12).

Since aG_g is not an i-generator of $h(aG_g)$, there exists a generator G': $\varnothing\subset G'\subset aG_g$: $h(G')=h(aG_g)$.

(Case α). If $a\notin G'$, then $G'\subseteq G_g$, $h(G_g)=h_{G_g}\subseteq h_G$ (because of the inductive assumption with $|G_g|=i$–1 and applying EOA for G). Hence, $h(aG_g)=h(G')\subseteq h(G_g)\subseteq h(aG_g)$. It follows that $a\in h(aG_g)=h(G_g)=h_{G_g}\subseteq h_G$, which contradicts the fact that $a\notin h_G$.

(Case β). Therefore, $a\in G'$ and there exists $G'\equiv aG_{gB}$: $h(G')=h(aG_g)$, where $\varnothing\neq B\subseteq G_g$, $G_{gB}=G_g\backslash B=G\backslash(gB)$. Since $0\leq|G_{gB}|\leq i-2$, $1\leq|G'|\leq i-1$. In other words, generator G' of $h(aG_g)$ has at most i–1 items.

What is left to show is that $\exists g\in G$: $h(aG_g)=h(G)$. Assume that the conversion becomes: $\forall g\in G$:

$$h(aG_g)\subset h(G)\Leftrightarrow\rho(G)\subset\rho(aG_g) \tag{13}$$

and using case β above, there exists $G'=aG_{gB}$ that is a generator of $h(aG_g)=h(aG_{gB})$, where $\varnothing\neq B\subseteq G_g$ and $a\notin B$. Take any $g'\in B$. Clearly, $g\neq g'\notin G'$. Then, $G_{g'}\cup G_g=G$, $aG_{gB}\subseteq aG_{g'}$, and $\rho(aG_{g'})\subseteq\rho(aG_{gB})$. However, we also have $\rho(aG_{g'})\subseteq\rho(G_{g'})$, $\rho(aG_{gB})=\rho(aG_g)\subseteq\rho(G_g)$. Taking the intersection of two sides yields $\rho(aG_{g'})=\rho(aG_{g'})\cap\rho(aG_{gB})\subseteq\rho(G_{g'})\cap\rho(G_g)=\rho(G_{g'}\cup G_g)=\rho(G)\subset\rho(aG_{g'})$ (by (13)). This is a contradiction.

Finally, there exist $g\in G$, $\varnothing\neq B\subseteq G_g$, generator $G'\equiv aG_{gB}$: $|G'|\leq i$–1 and closure of G: $P\equiv h(G')=h(aG_g)=h(G)\supset h_G\supseteq G$, i.e., we will discover the node $\langle P\rangle$ in $\mathcal{LCG}$. Thus, $\rho(P)=\rho(G)$ and $\text{supp}(P)=\text{supp}(G)=\text{supp}(h_G)$. □

3.3. GENCLOSE algorithm

We now describe GENCLOSE for simultaneously mining generators and their closures. Its pseudo-code is shown in Fig. 6. After non-frequent items have been eliminated from the database, the remaining items are sorted in ascending order first according to their supports and then according to their weights where the weight of a frequent item a is the sum of the supports of frequent 2-itemsets containing a (see Zaki and Hsiao (2005) for details). The sorted frequent items construct $L[1]$ on Line 3. The main loop of GENCLOSE is on Line 4. For $i=1$, $L[1]$ contains the 1-generators

```
GENCLOSE (D, minsup):
1.  A^F = SelectFreqItems-Sort (D, minsup);
2.  LCG = ∅; L[1] = ∅; HasNewLevel = true; i = 1;
3.  for each a ∈ A^F do L[1] = L[1] ∪ {<|ρ(a)|, a, {a}, ρ(a)>};
4.  while ( HasNewLevel ) do
5.     ExtendMerge ( L[i] );          // using EOB
6.     InsertLevel ( L[i], LCG );     // using EOC
7.     L[i+1] = ∅;          // produce (i+1)-generators and extend corresponding pre-closed itemsets by EOC
8.     for each Left in L[i] do
9.       for each Right in L[i] such that: Left < Right; Left, Right are in the same folder do
10.        NewO = Left.O ∩ Right.O; NewSupp = |NewO|;
11.        if ( NewSupp ≠ Left.Supp and NewSupp ≠ Right.Supp and NewSupp ≥ minsup ) then
12.             NewH = Left.H ∪ Right.H;                // using EOA
13.             JoinGenerators ( Left, Right, NewO, NewSupp, NewH, L[i+1] ); // using condition (5)
14.    if ( L[i+1] = ∅) then HasNewLevel = false;
15.    i = i+1;
16. return LCG;  // all frequent closed itemsets and their generators and supports
```

```
JoinGenerators (Left, Right, NewO, NewSupp, NewH, L[i+1]):
17. for each (G_l ∈ Left.GS, G_r ∈ Right.GS: |G_l|=i, |G_r|=i and |G_l∩G_r|=i-1) do
18.    G = G_l∪G_r; G_0 = G_l∩G_r; G_is_Generator = true;
19.    for each g ∈ G_0 do
20.       G_g = G\{g};
21.       Node_g = SearchNodeWithGenerator (G_g);
22.       if ( Node_g is null or NewSupp = Node_g.Supp ) then
23.          G_is_Generator = false;
24.          break;       // for each g ∈ G_0
25.       else            // G can be a generator
26.          NewH = NewH ∪ Node_g.H;     // using EOA
27.    if ( G_is_Generator ) then NewGS = NewGS ∪ {G};
28. if ( |NewGS| ≥ 1 ) then          // new (i+1)-generators
29.    L[i+1] = L[i+1] ∪ {<NewSupp, NewH, NewGS, NewO>};
```

Fig. 6. GENCLOSE algorithm.

together with their pre-closed itemsets. For $i > 1$, $L[i]$ contains the nodes whose i-generators are obtained from $(i–1)$-generators in $L[i–1]$ and the corresponding pre-closed itemsets are initialized and extended by operator EOA. For each $L[i]$ with $i \geq 1$, three phases are executed.

First phase. The ExtendMerge procedure (Line 5) is called to extend pre-closed itemsets at $L[i]$ by EOB. Let X and Y be two nodes at $L[i]$, where X comes before Y. It is determined which of the three following cases is satisfied. In the case that $X.O \subset Y.O$, we extend $X.H$ by $Y.H$: $X.H = X.H \cup Y.H$. If $X.O \supset Y.O$, we add $X.H$ to $Y.H$. In the remaining case, this procedure merges Y to X. It pushes all generators in $Y.GS$ to $X.GS$, adds $Y.H$ to $X.H$, and discards Y. Recall that we join only i-generators of G_1 and G_2 (in the nodes at $L[i]$) with common $i-1$ first items, called the common prefix. To avoid considering the node pairs that do not contain the generators with the same prefix, we split each level into folders such that the nodes in each folder share at least one common prefix. In the execution of the second phase, we only join the nodes of the same folder. Hence, if Y is not in the same folder with X, we move all nodes that are in the folder containing Y to the folder containing X. Thus, X also has the prefixes of Y.

Second phase. If $i = 1$, after the execution of EOB, all pre-closed itemsets in $L[1]$ are closed. Then, GENCLOSE inserts the 1-generators together with their closures and supports to $\mathscr{LCG}$. Otherwise, pre-closed itemsets in $L[i]$ $(i > 1)$, initialized and extended by EOA and EOB, are extended to be completed by EOC in InsertLevel (Line 6). For a given node X, it is checked whether $X.H$ is closed. If there exists itemset P in $\mathscr{LCG}$ such that supp$(P) = X.Supp$ and $P \supseteq X.H$, P is the closure that $X.H$ wants to reach. Thus, the new generators in $X.GS$ are pushed into the generator list of P and $X.H$ becomes P. In contrast, since $X.H$ is a new closed itemset, $X.H$ together with its generators and support are inserted into $\mathscr{LCG}$.

Third phase. This phase produces the nodes at $L[i+1]$ (initialized as empty on Line 7) by considering each node Left with the other nodes that come after it in the same folder. Let (Left, Right) be such a pair (Lines 8, 9). Obviously, $NewO$, computed on Line 10, is a closed object-set. If its cardinality is identical to that of either $Left.O$ or $Right.O$, or less than minsupp, we jump to the next pair. Otherwise, GENCLOSE initializes the corresponding pre-closed itemset by the union of $Left.H$ with $Right.H$ (using EOA) and calls JoinGenerators to mine the generators of $NewH$ from i-generators containing in Left and Right as well as to extend $NewH$ (using EOA) if required. The procedure only considers the pairs (G_l, G_r) of i-generators sharing $i–1$ common items (Line 17). Further, the inequality in (5) is checked only for $g \in G_0$ because for $g = g_i$ and $g = g_{i+1}$ it is always satisfied (Line 19). *For* $i = 1$, we have immediately 2-generators since G_0 is empty. *For* $i > 1$, if there exists a value of $g \in G_0$ such that $G_g = G \backslash \{g\}$ is not an i-generator or supp(G_g) is equal to supp(G), it can immediately be concluded that G is not an $(i+1)$-generator of $NewH$ (Line 24). Otherwise, G can be a generator. Thus, $NewH$ is extended by $Node_g.H$ (applying EOA on Line 26). Once G_is_Generator is true, GENCLOSE discovers new generator G. The joining of (Left, Right) creates a new node at $L[i+1]$ only when there exists at least a new respective $(i+1)$-generator (Line 29). The procedure ends and the algorithm returns to Lines 8 and 9 for considering the remaining node pairs.

Example 3. The execution of GENCLOSE on the example database with minsupp=1 is described graphically in Fig. 7. At the initialization, we have $L[1] = \{D = \langle ^2\{d\}^{\{2,3\}}, d\rangle, A = \langle ^3\{a\}^{\{1,2,3\}}, a\rangle, F = \langle ^3\{f\}^{\{2,3,4\}}, f\rangle, G = \langle ^3\{g\}^{\{1,3,4\}}, g\rangle, B = \langle ^4\{b\}^{\{1,4,5,6\}}, b\rangle, E = \langle ^4\{e\}^{\{1,3,4,5\}}, e\rangle, H = \langle ^4\{h\}^{\{1,2,3,4\}}, h\rangle$, and $C = \langle ^5\{c\}^{\{1,2,4,5,6\}}, c\rangle\}$.

For $i = 1$:

– We call ExtendMerge($L[1]$), which applies EOB. Since $D.O$ is contained in $A.O$, $F.O$ and $H.O$, $D.H$ becomes *dafh*. Similarly, the pre-closed itemsets in A, F, G and B are also extended. After the procedure ends, we have $L[1]$ such that all the pre-closed itemsets are closed.

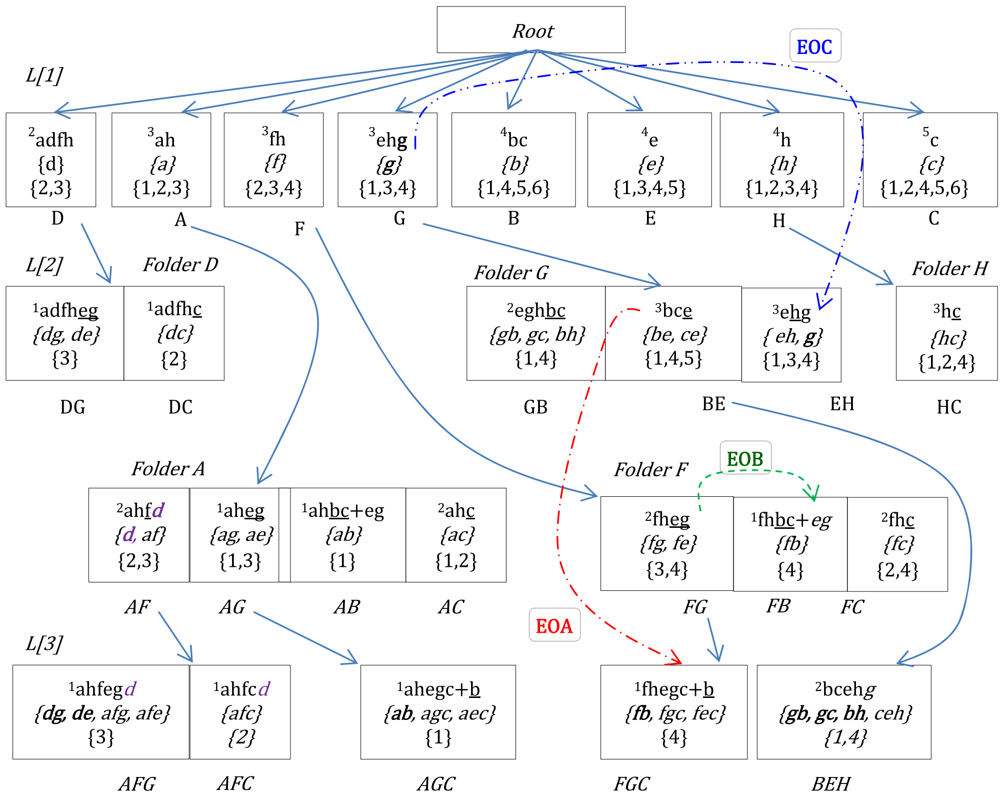


Fig. 7. IOG-tree used in GENCLOSE.

– In the second phase, InsertLevel inserts 1-generators together with their closures and supports into $\mathscr{LCG}$: $\mathscr{LCG}=\{({}^{2}adfh, \{d\}), ({}^{3}ah, \{a\}), ({}^{3}fh, \{f\}), ({}^{3}egh, \{g\}), ({}^{4}bc, \{b\}), ({}^{4}e, \{e\}), ({}^{4}h, \{h\}), ({}^{5}c, \{c\})\}$.
– In the last phase, we skip the pairs (D, A), (D, F), (D, H), (A, H), (F, H), (G, E), (G, H) and (B, C) without executing JoinGenerators as they do not pass the check on Line 11. For Left$=D$, we consider (D, G). Because $NewO=\{3\}$ and $NewSupp=1$, Line 12 outputs $NewH=D.H\cup G.H=adfheg$. In JoinGenerators, new 2-generator dg is discovered. Hence, we get a new node $\langle {}^{1}\{dg\}^{\{3\}}, adfheg\rangle$, called DG, at $L[2]$. Looking at the pairs (D, E) and (D, C), we discover $DE:=\langle {}^{1}\{de\}^{\{3\}}, adfhe\rangle$ and DC. These three nodes are in folder D. After finishing, this phase outputs DG, DE, DC (contained in folder D); $AF:=\langle {}^{2}\{af\}^{\{2,3\}}, ahf\rangle$, $AG:=\langle {}^{2}\{ag\}^{\{1,3\}}, aheg\rangle$, AB, $AE:=\langle {}^{2}\{ae\}^{\{1,3\}}, ahe\rangle$, AC (folder A); $FG:=\langle {}^{2}\{fg\}^{\{3,4\}}, fheg\rangle$, FB, $FE:=\langle {}^{2}\{fe\}^{\{3,4\}}, fhe\rangle$, FC (folder F); $GB:=\langle {}^{2}\{gb\}^{\{1,4\}}, eghbc\rangle$, $GC:=\langle {}^{2}\{gc\}^{\{1,4\}}, eghc\rangle$ (folder G); $BE:=\langle {}^{3}\{be\}^{\{1,4,5\}}, bce\rangle$, $BH:=\langle {}^{2}\{bh\}^{\{1,4\}}, bch\rangle$ (folder B); EH$:=\langle {}^{3}\{eh\}^{\{1,3,4\}}, eh\rangle$, $EC:=\langle {}^{3}\{ec\}^{\{1,4,5\}}, ec\rangle$; and HC (contained in folders E and H, respectively).
For i$=2$:
– Applying ExtendMerge on $L[2]$, we merge DE to DG, AE to AG, and FE to FG. The pre-closed itemsets of AG and FG enlarge the ones of AB and FB (illustrated by "$+eg$" and dashed arrow). Then, GC and BH are merged to GB, and CE is merged to BE. Thus, we also unite the folders G, B and E. Using similar computations, we get the new value of $L[2]$.
– Then, GENCLOSE executes InsertLevel to insert $L[2]$ into $\mathscr{LCG}$. Let us consider AF. Since $AF.H=ahf\subset adfh$ and supp$(ahf)=$sup$(adfh)=2$, using EOC.1, we conclude that af is a 2-generator of $adfh$. Similarly, we also discover generator eh of mined closed itemset egh. Hence, we update the two generator lists of the two nodes $(adfh)$ and (egh) in $\mathscr{LCG}$ (shown in bold and dashed doubledot arrow). Thus, the nodes AF and EH become $\langle {}^{2}\{d, af\}^{\{2,3\}}, ahfd\rangle$ and $\langle {}^{3}\{g, eh\}^{\{1,3,4\}}, ehg\rangle$, respectively. Then, we push new closed itemsets together with their generators to $\mathscr{LCG}$ (using EOC.2). Now, $\mathscr{LCG}=\{({}^{2}adfh, \{d, af\}), ({}^{3}ah, \{a\}), ({}^{3}fh, \{f\}), ({}^{3}egh, \{g, eh\}), ({}^{4}bc, \{b\}), ({}^{4}e, \{e\}), ({}^{4}h, \{h\}), ({}^{5}c, \{c\}), ({}^{1}adfheg, \{dg, de\}), ({}^{1}adfhc, \{dc\}), ({}^{1}aheg, \{ag, ae\}), ({}^{1}ahbceg, \{ab\}), ({}^{2}ahc, \{ac\}), ({}^{2}fheg, \{fg, fe\}), ({}^{1}fhbceg, \{fb\}), ({}^{2}fhc, \{fc\}), ({}^{2}eghbc, \{gb, gc, bh\}), ({}^{3}bce, \{be, ce\}), ({}^{3}hc, \{hc\})\}$.
– Initializing $L[3]$ as empty, GENCLOSE considers the joining of the nodes at $L[2]$ for each folder. At folder A, considering (AG, AC), we have: $NewO=\{1\}$, $NewSupp=1$ and $NewH=ahegc$. In JoinGenerators, for generator pair $(G_l=$ag, $G_r=$ac$)$: $G=$agc, $G_0=$a. We search and find that GB is the node containing $G_g=$gc as a generator. Since $GB.Supp>NewSupp$, agc is a new generator. The operator EOA extends $NewH$ by $GB.H$. Next, we consider the generator pair (ae, ac) and discover new generator aec of $NewH$. Thus, $AGC:=\langle {}^{1}\{agc, aec\}^{\{1\}}, aghecb\rangle$ is a new node at $L[3]$. Similarly, we get $AFG:=\langle {}^{1}\{afg, afe\}^{\{3\}}, ahefgd\rangle$, AFC and $FGC:=\langle {}^{1}\{fgc, fec\}^{\{4\}}, fghecb\rangle$ (see dashed arrow for the extension of $fgehc$ to $fgehcb$ by EOA) when considering the node pairs (AF, AG), (AF, AC) and (FG, FC), respectively. Now, BE is combined with EH. For generator pair (be, eh): $G=beh$, $G_0=e$. Since $Gg=bh$ is contained in $Node_g=GB$ and $GB.Supp=NewSupp$, beh is not a generator. For generator pair (ce, eh), we discover new generator ceh of $bceh$ and create node $BEH:=\langle {}^{2}\{ceh\}^{\{1,4\}}, bcehg\rangle$.
For i$=3$:
– First, we extend $AGC.H$ and $FGC.H$ by $BEH.H$.
– Then, GENCLOSE adds the generators afg, afe, afc, agc, aec, fgc, fec and ceh to $\mathscr{LCG}$: $\mathscr{LCG}=\{({}^{2}adfh, \{d, af\}), ({}^{3}ah, \{a\}), ({}^{3}fh, \{f\}), ({}^{3}egh, \{g, eh\}), ({}^{4}bc, \{b\}), ({}^{4}e, \{e\}), ({}^{4}h, \{h\}), ({}^{5}c, \{c\}), ({}^{1}adfheg, \{dg, de, \underline{afg}, \underline{afe}\}), ({}^{1}adfhc, \{dc\}), ({}^{1}aheg, \{ag, ae\}), ({}^{1}ahbceg, \{ab, \underline{agc}, \underline{aec}\}), ({}^{2}ahc, \{ac\}), ({}^{2}fheg, \{fg, fe\}), ({}^{1}fhbceg, \{fb, \underline{fgc}, \underline{fec}\}), ({}^{2}fhc, \{fc\}), ({}^{2}eghbc, \{gb, gc, bh, \underline{ceh}\}), ({}^{3}bce, \{be, ce\}), ({}^{3}hc, \{hc\}), ({}^{1}ahfcd, \{afc\})\}$.
– There is no joining at $L[3]$ and the algorithm ends.

Correctness and completeness. GENCLOSE enumerates all frequent closed itemsets together with their generators. First, it correctly lists all and only the generators. Its search is based on a complete bottom-up joining process. From the database, we get all frequent 1-generators. Then, at level i of the search, i-generator candidates are completely determined from mined $(i-1)$-generators. Necessary and sufficient condition (5), whose correctness was proven to be reliable in Theorem 1, is used to keep only the generators. The search branches with respect to the candidates that are not generators are pruned. That implies that the process of mining generators is complete and correct. Second, since we completely have all generators and their pre-closed itemsets will have been extended to their closures exactly (from Theorem 2), GENCLOSE completely discovers all frequent closed itemsets.

Implemented techniques. For fast implementation, we apply the diffset technique and a double-hash table, which are described in Appendix B.

4. Experimental study

Experiments were carried out on a personal computer with an Intel i5-2400 3-GHz CPU and 3.16 GB of RAM running under Linux and Cygwin. To test the correctness and performance of GENCLOSE, we compared it to CharmLMG and DTouch, two well-known algorithms for finding closed itemsets and generators. The source code of CharmLMG (includes Charm-L and MinimalGenerator) in C^{++} is publicly available from the website of the author himself (http://www.cs.rpi.edu/~zaki). DTouch is a fast implementation of Touch based on diffset (given at http://coron.wikidot.com/). Eight benchmark databases, namely C20d10k, C73d10k, Connect, Pumsb, Pumsbstar, Accidents, T25i4d10k, and T20i6d10k were used in the experiments. C20d10k, C73d10k, Pumsb, and Pumsbstar are census databases from the PUMS sample file. Connect was generated from game steps and Accident

Table 2
Database characteristics.

Database (DB)	Size (bytes)	# Transactions	# Items	Average length
Pumsb (Pum)	16,691,152	49,046	7117	74.0
Pumsbstar (Pum*)	11,293,562	49,046	7117	50.5
Accidents (Acc)	35,509,823	340,183	468	33.8
C73d10k (C73)	3,205,868	100,000	2177	73.0
Connect (Con)	9,255,309	67,557	129	43.0
C20d10k (C20)	800,000	100,000	385	20.0
T25i4d10k (T25)	970,890	10,000	1000	24.7
T20i6d10k (T20)	650,155	10,000	1000	16.0

Table 3
Minimum support thresholds.

DB	Minimum support thresholds (%)									
	1	2	3	4	5	6	7	8	9	10
Pum	95	92	89	87	85	82	80	79	75	70
Pum*	55	45	40	37	35	32	30	27	25	20
Acc	80	70	55	50	45	40	35	30	25	20
C73	80	75	70	68	65	62	60	57	55	52
Con	90	80	70	60	55	50	40	35	30	20
C20	10	5	1	0.5	0.4	0.3	0.2	0.1	0.08	0.06
T25	0.5	0.4	0.3	0.26	0.23	0.2	0.18	0.16	0.13	0.1
T20	0.2	0.18	0.16	0.14	0.12	0.1	0.09	0.08	0.07	0.06

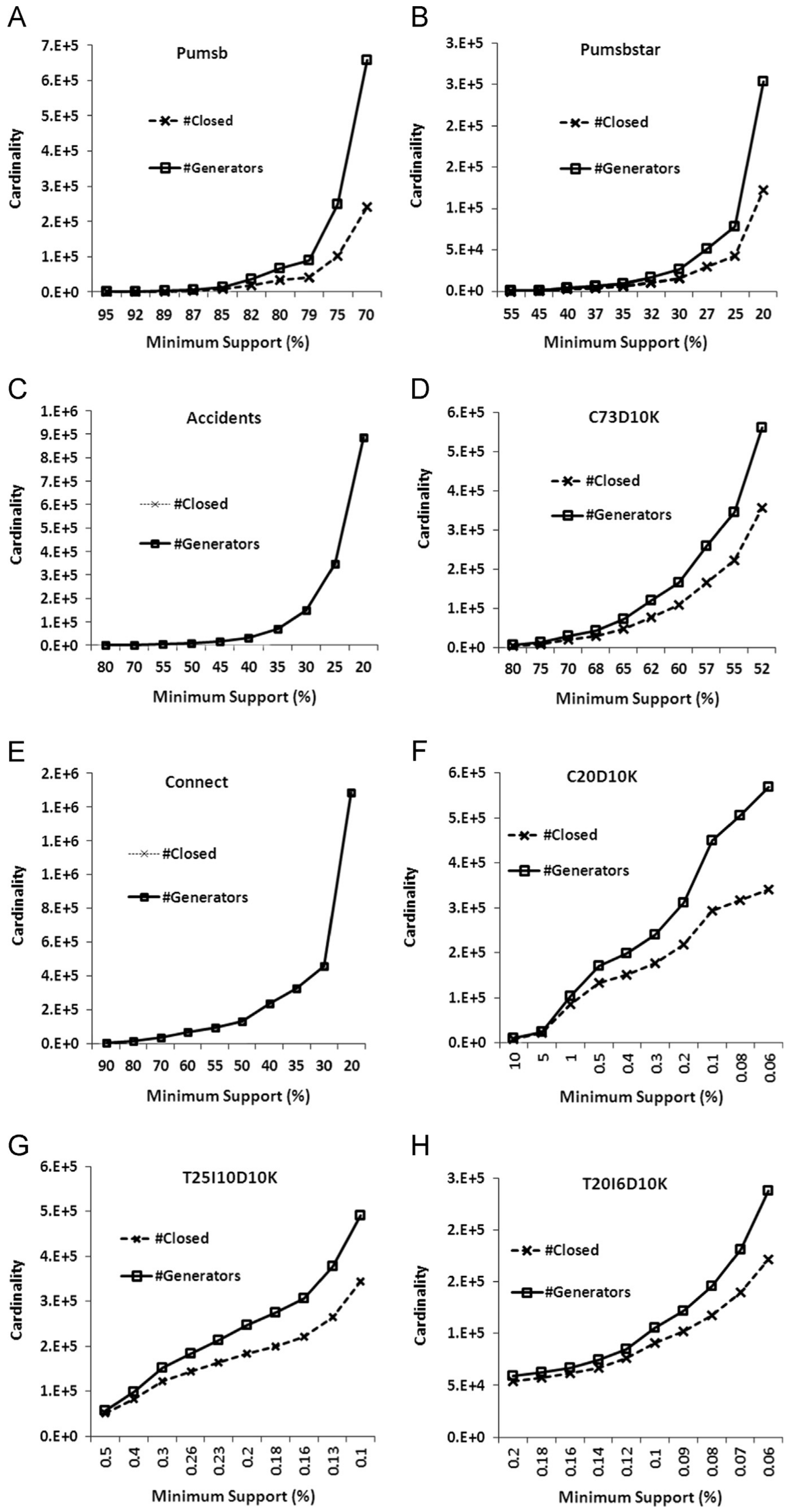


Fig. 8. Numbers of frequent closed itemsets and generators.

Table 4
Numbers of frequent closed itemsets.

DB	Minimum support thresholds (%)									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Pum	110	610	2186	4508	8509	19,934	33,282	42,363	101,003	241,197
Pum*	116	713	2610	4325	6133	10,649	16,154	29,279	42,756	122,262
Acc	149	529	4051	8057	16,124	32,529	68,224	149,529	346,389	887,388
C73	4262	9367	19,501	29,465	47,491	77,920	108,428	166,060	222,253	357,243
Con	3486	15,107	35,875	68,349	94,916	130,101	239,372	328,344	460,411	1,483,199
C20	8777	21,213	85,608	132,952	151,394	177,195	218,455	292,523	316,412	340,468
T25	52,033	83,062	122,581	144,177	163,625	184,300	200,666	221,074	221,074	343,733
T20	54,099	57,063	61,185	66,899	75,743	90,501	101,895	117,344	139,469	171,796

Table 5
Numbers of generators.

DB	Minimum support thresholds (%)									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Pum	141	811	3085	6763	13,789	37,023	67,810	90,252	248,170	658,380
Pum*	128	777	3587	6153	9039	16,763	26,971	51,800	78,219	253,240
Acc	149	529	4051	8057	16,124	32,529	68,224	149,529	346,389	887,388
C73	6281	13,918	29,008	44,065	71,875	119,560	166,918	258,145	346,029	346,029
Con	3486	15,107	35,875	68,349	94,916	130,101	239,372	328,344	460,411	1,483,199
C20	9332	23,052	102,316	170,261	170,261	240,382	310,117	310,117	310,117	568,254
T25	57,857	98,229	153,245	153,245	153,245	248,196	274,452	306,580	306,580	489,812
T20	58,548	62,035	67,084	74,018	85,052	105,281	121,872	145,644	180,840	237,345

contains traffic accident data. These databases, taken from the Frequent Itemset Mining Database Repository (http://fimi.cs.helsinki.fi/data/), are highly correlated, real and dense, i.e., they produce many long frequent itemsets, only a small fraction of which is closed. T25i4d10k and T20i6d10k are synthetic and sparse databases, randomly constructed according to the properties of market basket data using the IBM generator (available from IBM Almaden). Table 2 shows the experimental database characteristics, namely database size in bytes, number of lines (transactions), number of attributes (items) and average transaction length. Many values of minsupp thresholds for each database (computed on percentages), ranging from high to low, were used in the experiments (see Table 3).

4.1. Verification of correctness of GENCLOSE

In the previous section, we proved the correctness of GENCLOSE. Here, the correctness is verified. For all experiments (except for Con with minsupp 20%), CharmLMG always finishes its execution, DTouch fails on Pumsb with minsupps of 75% and 70%, on C73 with a minsupp of 52%, and on Pum⁎ with a minsupp of 20%, and GENCLOSE fails on Acc with a minsupp of 20%. Except for cases where one or both of CharmLMG and DTouch fail, the output of GENCLOSE is identical to theirs. The numbers of mined frequent closed itemsets and generators are shown graphically in Fig. 8. To verify the correctness of the cardinalities of mined frequent closed itemsets as well as generators with respect to the existing algorithms, we also numerically show them in Tables 4 and 5. For Pum, Pum*, C73, C20, T25, and T20, the numbers of generators are about 1.8, 1.5, 1.5, 1.4, 1.3 and 1.2 times bigger than those of frequent closed itemsets, respectively. However, for Con and Acc, those numbers are nearly identical.

4.2. Performance of GENCLOSE

The runtimes of the three algorithms are shown in Fig. 9. First, for Pum, CharmLMG runs faster than DTouch at high and medium values of minsupp (ranging from minsupps of 95% to 79%). At the lower values (for minsupps of 75% and 70%), DTouch fails. Between GENCLOSE and CharmLMG, there are negligible differences in their runtimes at higher minsupps (from threshold 1 to threshold 5); however, for smaller values of minsupp, those differences become huge. GENCLOSE can be up to 6 times faster at the two lowest minsupps and about 3 times faster on average than CharmLMG. This also holds for Pum⁎, but the difference between CharmLMG and GENCLOSE is more considerable. On Acc and T20, GENCLOSE is many times faster than CharmLMG, which always outperforms DTouch, especially at low thresholds of minimum support. The run times of DTouch and CharmLMG for Con, C73 and C20 show that DTouch outperforms CharmLMG at almost all thresholds. Their average runtime ratios are 2, 4 and 6 for Con, C73 and C20, respectively. With decreasing minsupp, the performance gap between them widens. For T25, there are negligible differences in the runtimes of DTouch and CharmLMG. For four of these databases, GENCLOSE is faster than DTouch, especially for Con and T25, for which it is 2–4 times faster on average.

We next analyze how GENCLOSE improves the performance of mining frequent closed itemsets and their generators for three groups of databases. Let T_{GC}, T_{CM}, and T_{DT} be the runtimes of GENCLOSE, CharmLMG, and DTouch, respectively. We call $RT_{CM/GC}$ the ratio of T_{CM} to T_{GC} ($RT_{\mathrm{CM/GC}}:=T_{CM}/T_{GC}$) and $\mathrm{RT}_{\mathrm{DT/GC}}$ the ratio of T_{DT} to T_{GC} ($RT_{DT/GC}:=T_{DT}/T_{GC}$). The average length of closed itemsets is denoted by $|C|$. For each closed itemset, we get the average length of its generators and compute the ratio as a percentage of that number to its length. The average number of all closed itemsets is $|G|_{|I|/|C|}$ (%). Table 6 shows the average values of $RT_{\mathrm{CM/GC}}$, $RT_{DT/GC}$, $|C|$ and $|G|_{|I|/|C|}$ for various minsupps for each database.

Group (A), which includes Pum, Pum*, and Acc, has large values of Avg-$|G|_{|I|/|C|}$, ranging from 74.6% to 100%. For Acc, Avg-$|G|_{|I|/|C|}$ can be up to 100%, i.e., the only generator of a frequent closed itemset is nearly itself. Those values imply that the generators mined from those databases are very close to their closures. Hence, it is easy for GENCLOSE to obtain frequent closed itemsets from their generators using the proposed extension operators. Since DTouch discovers them in a stand-alone phase, it takes a lot of time. GENCLOSE can be faster than DTouch by up to 11 times, as shown in column Avg-$\mathrm{RT}_{\mathrm{DT/GC}}$ on rows 1, 2 and 3. For group (B),

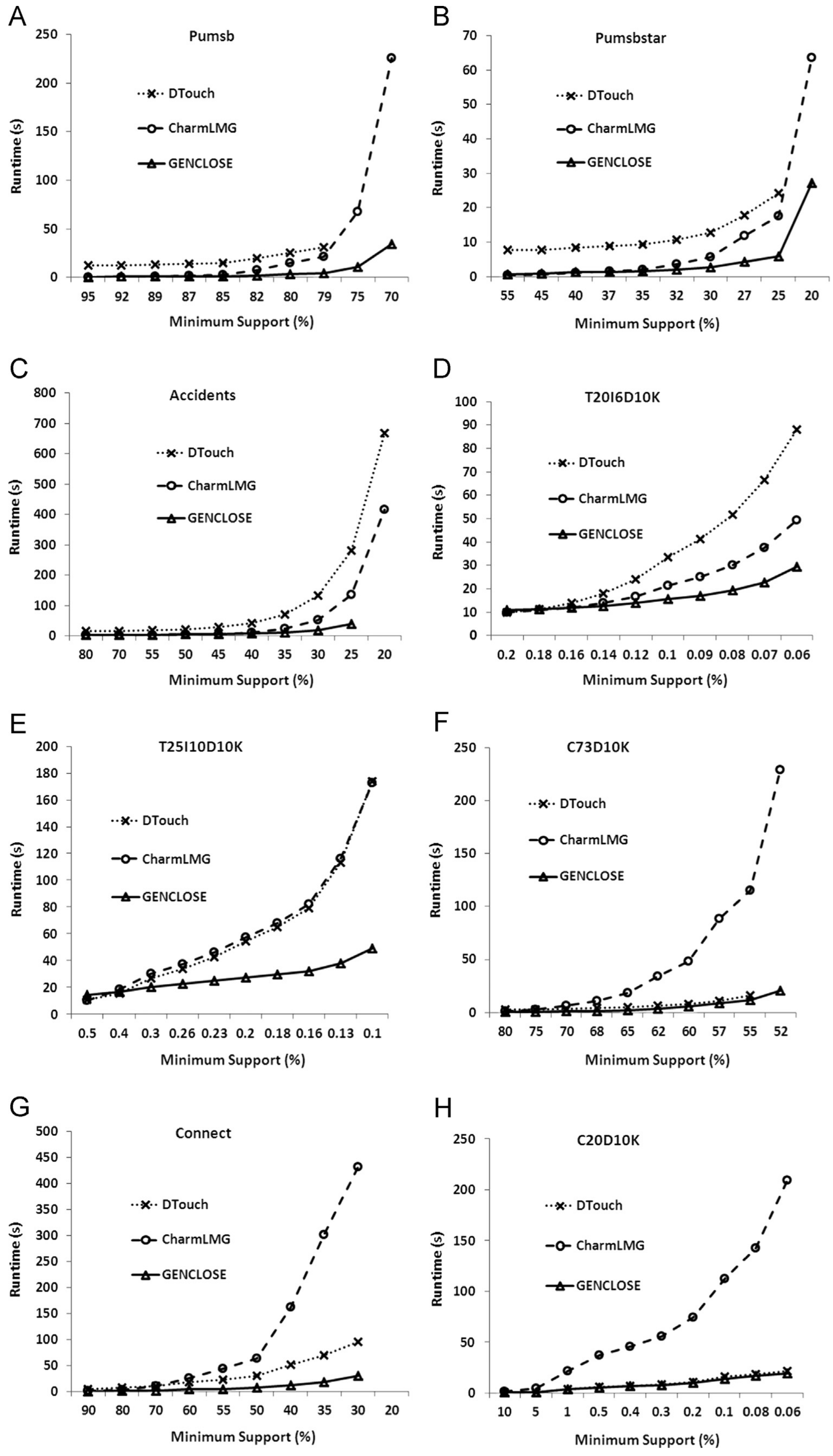


Fig. 9. Runtimes of GENCLOSE, CharmLMG, and DTouch.

Table 6
Performance of GENCLOSE compared to CharmLMG and DTouch.

Group	DB	Avg-|C|	Avg-$\|G\|_{\|I\|/\|C\|}$ (%)	Avg-$RT_{CM/GC}$	Avg-$RT_{DT/GC}$
(A)	Pum	6.0	90.0	3.2	11.7
	Pum*	6.7	74.6	1.6	6.5
	Acc	5.3	100.0	1.6	6.2
(B)	C73	10.7	59.9	7.9	2.9
	Con	12.1	47.3	8.3	4.3
	C20	10.4	49.6	7.2	1.2
(C)	T25	5.0	85.8	2.0	1.9
	T20	4.1	92.8	1.3	1.9

GENCLOSE has difficulty when trying to reach long frequent closed itemsets that are much longer than their generators. However, it is still about 3 or 4 times faster than DTouch (except on C20, where their performance difference is only 1.2 times).

Now, let us compare GENCLOSE to CharmLMG. For group (A), which produces short frequent closed itemsets, the mining of generators with short lengths by MinimalGenerator is fast. Hence, the total runtimes of CharmLMG are similar to those of GENCLOSE (see first three rows in column Avg-$RT_{CM/GC}$). For group (B), the time for checking the sufficient condition for generators based on long frequent closed itemsets is longer. CharmLMG is up to 7.8 times (on average) slower than GENCLOSE.

The synthetic databases T25i4d10k and T20i6d10k are sparse and thus produce very short frequent closed itemsets (with average lengths of 5.0 and 4.1, respectively, as shown in Table 6). For this group, the performances of three algorithms are similar. However, GENCLOSE is still about 2 times faster.

5. Conclusions and future works

This study described the properties of closed sets and generators as well as their relations. The necessary and sufficient conditions to produce generators and operators to determine closed itemsets were derived. The GENCLOSE algorithm was proposed for simultaneously mining frequent closed itemsets and their generators. Using the IOG-tree framework, it uses the itemset object-set and generator space, which allow not only quick identification of new generators without accessing the database or testing mined generators, but also extension of pre-closed itemsets toward their closures. Experiments on real and synthetic benchmark datasets confirm that GENCLOSE significantly outperforms existing methods in mining frequent closed itemsets and generators.

For mining either closed itemsets or generators separately, depth-first algorithms usually outperform level-wise ones. An interesting extension is to develop a depth-first miner based on our approach. Further, to deal with large databases, it is necessary to convert GENCLOSE for parallel and distributed environments as well as to execute different mining tasks, such as sequence mining.

Acknowledgment

This work was funded by the National Foundation for Science and Technology Development (NAFOSTED) under Grant no. 102.05-2013.20.

Appendix A. Proofs

Proposition 1. *Proof.* Here, we only prove property 5.(b).

"$\Rightarrow$" Since $\rho(A_1)\subset\rho(A_2)$, $h(A_1)=\lambda(\rho(A_1))\supseteq\lambda(\rho(A_2))=h(A_2)$. If $h(A_1)=h(A_2)$, using 5.(a), $\rho(A_1)=\rho(A_2)$. This is a contradiction. Hence, $h(A_1)\supset h(A_2)$.

"$\Leftarrow$" Since $h(A_1)\supset h(A_2)$, using the anti-monotone law of ρ, $\rho(h(A_1))=\rho(A_1)\subseteq\rho(h(A_2))=\rho(A_2)$. If $\rho(A_2)=\rho(A_1)$, then $h(A_1)=h(A_2)$ (from 5.(a)). This implies that $\rho(A_2)\supset\rho(A_1)$. □

Consequence 1. *Proof.*

1) (a) "$\Rightarrow$" (b): For every $P\supset A=h(A)$, we always have $\rho(A)\supseteq\rho(P)$ and $\text{supp}(A)\geq\ \text{supp}(P)$. Now, let us assume that $\text{supp}(A)=\text{supp}(P)$. Thus, $\rho(A)=\rho(P)$ and $P\supset A=h(A)=h(P)\supseteq P$. This implies a contradiction. Hence, $\text{supp}(A)>\text{supp}(P)$.
(b) "$\Rightarrow$" (c): For every $P\supset A$, $\rho(A)\supseteq\rho(P)$. If $\rho(A)=\rho(P)$, then $\text{supp}(A)=\text{supp}(P)$. It is a contradiction. Thus, $\rho(A)\supset\rho(P)$.
(a) "$\Leftarrow$" (c): We have $A\subseteq h(A)$ and $\rho(A)=\rho(h(A))$. If $A\subset h(A)$, then we take $P\equiv h(A)$, $\rho(A)\supset\rho(P)=\rho(h(A))$: a contradiction. Hence, $A=h(A)$.
2) For closed itemset $h(A)$, we have $h(A)\supseteq A$, $\rho(A)=\rho(h(A))$, $\text{supp}(A)=\text{supp}(h(A))$, $h(A)=h(h(A))$.
(a) As follows from property 5 of Proposition 1 and Remark 1, we have for every $Q\supset P$: $h(Q)\supset h(P)\ \Leftrightarrow\ \rho(Q)\subset\rho(P)\ \Leftrightarrow\ \text{supp}(Q)<\text{supp}(P)$ and if $P\supseteq$ A, then $h(A)=h(P)\ \Leftrightarrow\ \rho(A)=\rho(P)\ \Leftrightarrow\ \text{supp}(A)=\text{supp}(P)$. What is left is to prove that the closure of A is the unique set P such that

$$P\supseteq A, h(P)=h(A) \text{ and } (h(Q)\supset h(P), \forall Q\supset P) \tag{14}$$

+ Clearly, $h(A)$ satisfies (14): $h(A)=h(h(A))\supseteq A$ and ($\forall\ Q\supset h(A)$, $h(Q)\supseteq h(A)$). If $h(A)=h(Q)$, $Q\supset h(A)=h(Q)\supseteq Q$. This is a contradiction. Therefore, $h(Q)\supset h(A)=h(h(A))$.
+ We will show the fact that for P satisfying (14), $P=h(A)$. Since $h(A)=h(P)\supseteq P\supseteq A$, if $A\subseteq P\subset h(A)$, we take $Q=h(A)\supset P\supseteq A$. Thus,

$$h(A)=h(Q)\supset h(P) \tag{15}$$

Otherwise, $h(A)=h(h(A))\supseteq h(P)\supseteq h(A)$ implies that $h(A)=h(P)$. This contradicts (15). Hence, $P=h(A)$ is unique.
(b) Obviously, the closure $h(A)$ is the superset of A which satisfies (1). For every $B\supseteq A$, from properties of 2 and 5 of Proposition 1 and Remark 1, the conditions in (1) are equivalent, i.e., $\text{supp}(A)=\text{supp}(B)\ \Leftrightarrow\ \rho(A)=\rho(B)\Leftrightarrow h(A)=h(B)$. Then, we only need to prove that $h(A)$ is the maximum superset of A such that $h(A)=h(B)$. Conversely, assume that there exists $B\supset h(A)$ such that B is a superset of A with $h(A)=h(B)$. Thus, $B\supset h(A)=h(B)\supseteq B$. This is a contradiction. Thus, the closed set $h(A)$ is the maximum superset of A satisfying (1).
3) It is easy to see that $h(A)$ is a closed itemset containing A. Let B be a closed itemset containing A, i.e., $B=h(B)\supseteq A$. By the monotone and idempotence properties, we have $B=h(B)=h(h(B))\supseteq h(A)$. Therefore, $h(A)$ is a minimum closed itemset containing A with the set containment order "$\subseteq$". □

Consequence 2. Proof.

The equivalence (a)$\Leftrightarrow$(b) comes naturally from property 5 of Proposition 1. Based on Remark 1 and the fact that $[G\subseteq A\Rightarrow\rho(G)\supseteq\rho(A)]$, we have (a)$\Leftrightarrow$(c).

This proof is deduced from the above statement since $h(G)=h(h(G))$, $\rho(G)=\rho(h(G))$ and $\text{supp}(G)=\text{supp}(h(G))$. □

Proposition 2. Proof.

Assuming that $|A|=m$. Let us finitely consider the subsets of A, each of which is created by deleting an item of A: $A_{i_1}=A\backslash\{a_{i_1}\}, a_{i_1}\in A, \forall i_1=1\ldots m$.

Case 1: If $\rho(A_{i_1}) \supset \rho(A)$, $\forall i_1 = 1 \ldots m$, then A is a generator of A.
Case 2: Otherwise, there exists $i_1 = 1 \ldots m$: $\rho(A_{i_1}) = \rho(A)$. The above steps are repeated for A_{i_j} $(j = 1 \ldots m-1)$ until:
case 1 happens thus, we get the generator $A_{i_{j-1}}$ of A; or
case 2 happens when $|A_{i_{m-1}}| = 1$, i.e., $r(A_{i_1}) = r(A_{i_2}) = \cdots = \rho(A_{i_{m-1}}) = \rho(A)$. Hence, $A_{i_{m-1}}$ is a generator of A. Since A is finite, we always discover a generator of A.

$G \in \mathcal{G}(A) \Leftrightarrow [G \subseteq A,\ h(G) = h(A)$ and $(h(G') \subset h(G), \forall G'$: $\varnothing \neq G' \subset G)]$. Furthermore, for every $B \supseteq A$ such that $h(A) = h(B)$. Then, $G \in \mathcal{G}(A)$ implies that $[G \subseteq B,\ h(G) = h(B)$ and $(h(G') \subset h(G), \forall G'$: $\varnothing \neq G' \subset G)]$, i.e., $G \in \mathcal{G}(B)$. Thus, $\mathcal{G}(A) \subseteq \mathcal{G}(B)$.

"$\Rightarrow$": If $G \in \lfloor A \rfloor$ is a generator of A but not minimal in $\lfloor A \rfloor$, there exists $G' \in \lfloor A \rfloor$ such that $\varnothing \neq G' \subset G$. Hence, $h(G') = h(A)$. This contradicts the fact that G is a generator.

"$\Leftarrow$": Let us consider a minimal element G of $\lfloor A \rfloor$. We have $\varnothing \neq G \subseteq A$, $h(G) = h(A)$. Further, for every proper subset G' of G, we have: $h(G') \subset h(G)$, $\forall G'$: $\varnothing \neq G' \subset G$. Since $G' \subset G$, then $h(G') \subseteq h(G)$. If $h(G') = h(G)$, i.e., G' is in $\lfloor A \rfloor$. Therefore, G is not minimal. This is a contradiction. Hence, G is a generator of A. □

Appendix B. Implemented techniques

Fast implementation based on diffsets. In order to reduce memory, the diffset technique is used in many algorithms, such as Charm, Eclat, and Touch. It is thus applied in GENCLOSE. Instead of storing whole object-sets, we only store the corresponding diffsets. A diffset at each node keeps track of its object-set from the diffsets of its left-most ancestors. The diffset of node N^k at $L[k]$ (with $k \geq 1$) is defined as the difference between its object-set and the object-set of its left-parent Left at $L[k-1]$, i.e.:

$$d(N^k, Left) := \rho(Left.H) \setminus \rho(N^k.H)$$

Since this difference is insignificant in comparison with the object-set size, the memory required is considerably reduced. As convention, the unique left-parent of all nodes in $L[1]$ is Root with $\rho(\text{Root}) = \{a \in \mathcal{A}: \sup(\{a\}) \geq \text{minsupp}\}$. For example, the differences for nodes G, B, E, H, and C are {2, 5, 6}, {2, 3}, {2, 6}, {5, 6}, and {3}, respectively (see Fig. 11). For $k > 1$, let (*Left, Right*) be the pair that produces N^k and LP be their common left-parent. It is easy to find that (see Fig. 10):

$$d(N^k, Left) = d(Right, LP) \setminus d(Left, LP) \tag{16}$$

$$N^k.Supp = Left.Supp - |d(N^k, Left)| \tag{17}$$

Observing folder G in Fig. 7, we notice that GB comes from not only left-parent G but also from B (since we merged BH to **GB**). Hence, GB contains two diffsets, {3} and {5, 6}. In general, a node N with a diffset is obtained in the form $(^{\text{Supp}}H$, {[Left-parent$_j$, d(N, Left-parent$_j$), GS_j]}), where GS_j is the set of generators generated from Left-parent$_j$ (jth left-parent), $j = 1, 2, \ldots$ For instance, $GB = (^2eghbc$, {[G, {3}, {gb, gc}], [B, {5,6}, {bh}]}) (see Fig. 11).

Let us consider the execution of ExtendMerge with diffsets. If two nodes *Left* and *Right* have the same left-parent *LP*, the relation of the two respective diffsets implies that of two object-sets:

$$d(Left, LP) \subseteq d(Right, LP) \Leftrightarrow \rho(Left.H) \supseteq \rho(Right.H)$$

Otherwise, their pre-closed itemsets share the same object-set only for:

$$(Left.H \subseteq Right.H \text{ or } Right.H \subseteq Left.H) \text{ and } Left.Supp = Right.Supp$$

For example, we merge GC to GB since $d(GC, G) = d(GB, G)$, but, BH to GB since ($BH.H \subset GB.H$ and $BH.Supp = GB.Supp$). In the second phase, for the combination of the two nodes Left and Right, we need to locate their shared left-parent. If it does not exist, these nodes cannot be joined together. Otherwise, let LP be that left-parent. We compute *NewDiff* (instead of *NewO*) based on the two corresponding diffsets and then *NewSupp* using (16) and (17). In JoinForGenerators, we consider only the generator pairs (G_l, G_r) such that G_l and G_r have the same left-parent. For instance, for (GB, BE): $LP = B$, $NewDiff = \varnothing$. Thus, $NewSupp = 2$ does not pass the test on Line 11. For (BE, EH), we consider the generator pair of *ce* and *eh*

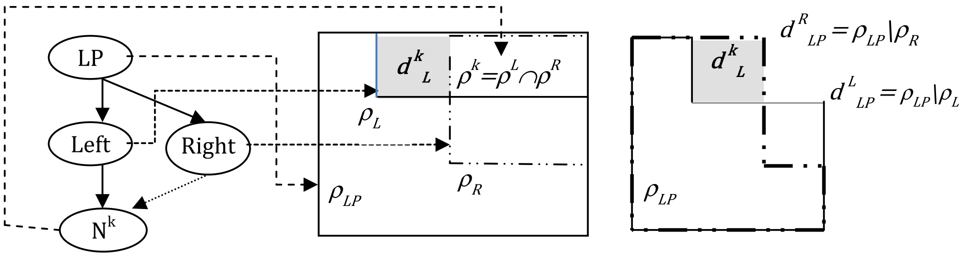


Fig. 10. Diffset computation.

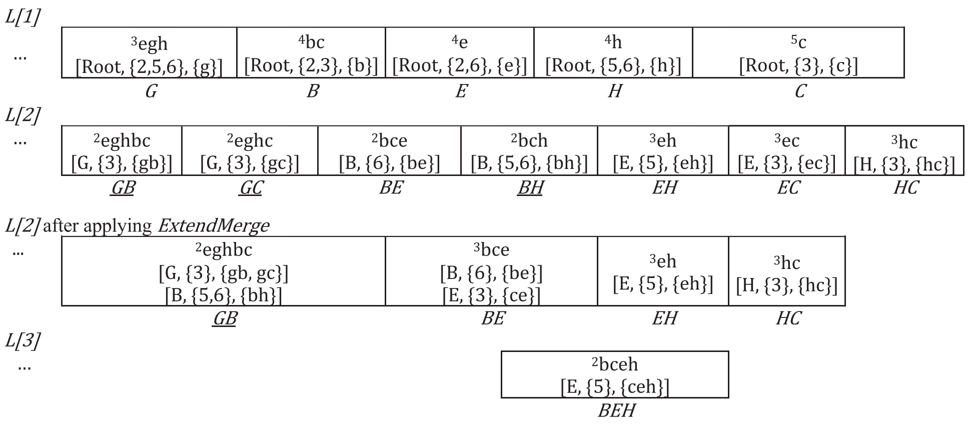


Fig. 11. Execution of GENCLOSE with diffsets.

since their common left-parent is E. Since there exists $G = ceh$, it passes the tests on Lines 27 and 28 and new node BEH is created. See Fig. 11 for the execution of GENCLOSE with diffsets.

Fast searching on $\mathscr{LCG}$ using a doublehash. In order to reach the closures using operator EOC, we need to quickly search $\mathscr{LCG}$ for closure P of itemset h_G. Obviously, a search based only on the support (supp(h_G)) or/and the itemset (h_G) is not efficient. As the object-set shared an itemset is identical to the one shared its closure, using the information about the known object-set (O_G) to determine the closure of an itemset helps reduce the search time considerably. We first apply the hash technique based on the sum of object-sets (proposed by Zaki and Hsiao (2005)). Since we store only diffsets and not object-sets, the sum should be computed through diffsets. More formally, let N^k be a new node with left-parent *Left*. Its object-set sum, called N^k.*SumO*, is computed by

$$N^k.SumO = Left.SumO - \sum_{m \in d(N^k, LP)} m$$

Since many closures can have the same hash value (based on the sum of objects), we hash them later by the support. Once an itemset h_G passes the tests on the hash values based on the object-set sum and support, we check the support and set containment relation to locate its closure.

References

Agrawal, R., Imielinski, T., Swami, N., 1993. Mining association rules between sets of items in large databases. In: Proceedings of the ACM SIGMOID 1993, pp. 207–216.

Agrawal, R., Srikant, R., 1994. Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Data Bases, pp. 478–499.

Anh, T., Hai, D., Tin, T., Bac, L., 2011. Efficient algorithms for mining frequent itemsets with constraint. In: Proceedings of the 3rd International Conference on Knowledge and System Engineering KSE 2011, pp. 19–25.

Anh, T., Tin, T., Bac, L., 2012a. Structures of association rule set. In: Proceedings of ACIIDS 2012, LNAI 7197, Part II. Springer-Verlag, pp. 361–370.

Anh, T., Hai, D., Tin, T., Bac, L., 2012b. Mining frequent itemsets with dualistic constraints. In: Proceedings of PRICAI 2012, LNAI 7458. Springer-Verlag, pp. 807–813.

Anh, T., Tin, T., Bac, L., 2013. An approach for mining concurrently closed itemsets and generators. Adv. Comput. Methods Knowl. Eng. SCI 479, 355–366.

Anh, T., Tin, T., Bac, L., 2014. An approach for mining association rules intersected with constraint itemsets. Adv. Intell. Syst. Comput. 245, 351–363.

Balcazar, J.L., 2010. Redundancy, deduction schemes, and minimum-size base for association rules. Log. Methods Comput. Sci. 6 (2:3), 1–33.

Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., Lakhal, L., 2000. Mining frequent patterns with counting inference. SIGKDD Explor. 2 (2), 66–75.

Bay, V., Hong, T.P., Bac, L., 2012. Mining most generalization association rules based on frequent closed itemset. Int. J. Innov. Comput., Inform. Control 8 (10), 1–17.

Bayardo, R.J., 1998. Efficiently mining long patterns from databases. In: Proceedings of the SIGMOD Conference 1998, pp. 85–93.

Birkhoff, G., 1967. Lattice Theory, 3rd edition American Mathematical Society, Providence, RI.

Boulicaut, J., Bykowski, A., Rigotti, C., 2003. Free-Sets: a condensed representation of boolean data for the approximation of frequency queries. Data Min. Knowl. Discov. 7, 5–22.

Burdick, D., Calimlim, M., Gehrke, J., 2001. MAFIA: a maximal frequent itemset algorithm for transactional databases. In: Proceedings of ICDE'01, pp. 443–452.

Davey, B.A., Priestley, H.A., 1994. Introduction to Lattices and Order, fourth edition Cambridge University Press, Cambridge.

Dong, G., Jiang, C., Pei, J., Li, J., Wong, L., 2005. Mining succinct systems of minimal generators of formal concepts. In: Proceedings of DASFAA 2005, LNCS 3453, pp. 175–187.

Ganter, B., Wille, R., 1999. Formal Concept Analysis: mathematical foundations. Springer-Verlag.

Hai, D., Tin, T., Bac, L., 2013. An efficient algorithm for mining frequent itemsets with single constraint. Adv. Comput. Methods Knowl. Eng. SCI 479, 367–378.

Hai, D., Tin, T., Bac, L., 2014. An efficient method for mining frequent itemsets with double constraints. J. Eng. Appl. Artif. Intell. 27, 148–154.

Han, J., Pei, J., 2000. Mining frequent patterns by pattern-growth: methodology and implications. ACM SIGKDD Explor. 2 (2), 14–20.

Han, J., Pei, J., Yin, J., Mao, R., 2004. Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min. Knowl. Discov. 8 (1), 53–87.

Hashem, T., Ahmed, C.F., Samiullah, M., Akther, S., Jeong, B.-S., Jeon, S., 2014. An efficient approach for mining cross-level closed itemsets and minimal association rules using closed itemset lattices. Expert Syst. Appl. 41 (6), 2914–2938.

Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L., 1999. Efficient mining of association rules using closed item set lattices. Inf. Syst. 24 (1), 25–46.

Pasquier, N., Taouil, R., Bastide, Y., Stumme, G., Lakhal, L., 2005. Generating a condensed representation for association rules. J. Intell.t Inf. Syst. 24 (1), 29–60.

Pei, J., Han, J., Mao, R., 2000. CLOSET: an efficient algorithm for mining frequent closed itemsets. In: Proceedings of the DMKDWorkshop on Research Issues in Data Mining and Knowledge Discovery 2000, pp. 21–30.

Singh, N.G., Singh, S.R., Mahanta, A.K., 2005. CloseMiner: discovering frequent closed itemsets using frequent closed tidsets. In: Proceedings. of the 5th ICDM, Washington DC, USA, pp. 633–636.

Szathmary, L., Valtchev, P., Napoli, A., 2009. Efficient vertical mining of frequent closed itemsets and generators. In: Proceedings of IDA 2009, pp. 393–404.

Tin, T., Anh, T., 2010a. Structure of set of association rules based on concept lattice. In: Proceedings of ACIIDS 2010, Advanced in Intelligent Information and Database Systems, SCI 283, Springer, pp. 217–227.

Tin, T., Anh, T., Thong, T., 2010b. Structure of association rule set based on min-min basic rules. In: Proceedings of the International Conference on Computing and Communication Technologies RIVF 2010, pp. 83–88.

Zaki, M.J., 2000. Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng. 12 (3), 372–390.

Zaki, M.J., Gouda, K., 2003. Fast vertical mining using diffsets. In: Proceedings of the 9th ACM SIGKDD.

Zaki, M.J., 2004. Mining non-redundant association rules. Data Min. Knowl. Discov. 9, 223–248.

Zaki, M.J., Hsiao, C.J., 2005. Efficient algorithms for mining closed itemsets and their lattice structure. IEEE Trans. Knowl. Data Eng. 17 (4), 462–478.

Vo, B., Le, B., 2009. Fast algorithm for mining minimal generators of frequent closed itemsets and their applications. In: Proceedings of 39th International Conference on Computers and Industrial Engineering, pp. 1407–1411.

Wang, J., Han, J., Pei, J., 2003. Closet+: searching for the best strategies for mining frequent closed itemsets. In: Proceedings of ACM SIGKDD'03.

Wille, R., 1982. Restructuring lattices theory: an approach based on hierarchies of concepts. In Ordered Sets, pp. 445–470.

Wille, R., 1992. Concept lattices and conceptual knowledge systems. Comput. Math. Appl. 23, 493–515.