Mining Succinct Systems of Minimal Generators of Formal Concepts

Guozhu Dong[†], Chunyu Jiang[†], Jian Pei[‡], Jinyan Li[¶], Limsoon Wong[¶]

[†] Wright State University, U.S.A. {gdong,cjiang}@cs.wright.edu
[‡] Simon Fraser University, Canada. jpei@cs.sfu.ca
[¶] Institute for Infocomm Research, Singapore. {jinyan,limsoon}@i2r.a-star.edu.sg

Abstract. Formal concept analysis has become an active field of study for data analysis and knowledge discovery. A formal concept C is determined by its extent (the set of objects that fall under C) and its intent (the set of properties or attributes covered by C). The intent for C, also called a closed itemset, is the maximum set of attributes that characterize C. The minimal generators for C are the minimal subsets of C's intent which can similarly characterize C. This paper introduces the *succinct system of minimal generators* (SSMG) as a minimal representation of the minimal generators of all concepts, and gives an efficient algorithm for mining SSMGs. The SSMGs are useful for revealing the equivalence relationship among the minimal generators, which may be important for medical and other scientific discovery; and for revealing the extent-based semantic equivalence among associations. The SSMGs are also useful for losslessly reducing the size of the representation of all minimal generators, similar to the way that closed itemsets are useful for losslessly reducing the size of the representation of all frequent itemsets. The removal of redudancies will help human users to grasp the structure and information in the concepts.

Keywords: Minimal generators, formal concepts, closed itemsets, succinctness

1 Introduction

Formal concept analysis (FCA) [7] is an important tool for data analysis and knowledge discovery. A formal concept C is determined by its extent (the set of objects or transactions that fall under C) and its intent (the set of properties, attributes, or items covered by C). Take the transaction database TDB in Figure 1 as an example. Each transaction has an identity Tid and a set of items; the set of items is written as a list of items alphabetically and the set brackets are omitted. Itemset $bcghi$ and transaction set $\{T_1, T_3, T_5\}$ form a formal concept, where itemset $bcghi$ is its intent and transaction set $\{T_1, T_3, T_5\}$ is its extent. Intuitively, $bcghi$ is the largest itemset that is contained in transactions T_1, T_3 and T_5. No other transactions contains $bcghi$. The formal concepts in the transaction database are listed in Figure 2.

In general, the intent of a formal concept C is the closure of the properties, attributes, or items that form a maximum characterization for C: Every object satisfying the intent is in C. The closure (or a closed itemset[1]) serves as the upper bound of

[1] Since formal concepts and closed itemsets are in one-to-one correspondence, we henceforth treat a closed itemset and its corresponding formal concept as the same thing.

Tid	Items
T_1	$abcdeghi$
T_2	$acdg$
T_3	$bcdghi$
T_4	$abdhi$
T_5	$bceghi$

Fig. 1. A transaction database TDB.

Closure	Minimal generators	Sup	SuccMinGen
ad	a	3	a
bhi	b, h, i	4	b, h, i
cg	c, g	4	c, g
d	d	4	d
$bceghi$	e	2	e
$abdhi$	ab, ah, ai	2	ab
$acdg$	ac, ag	2	ac
$abcdeghi$	$ae, de, abc, abg,$ ach, aci, agh, agi	1	ae, de, abc
$bcghi$	bc, bg, ch, ci, gh, gi	3	bc
$bdhi$	bd, dh, di	3	bd
cdg	cd, dg	3	cd
$bcdghi$	$bcd, bdg, cdh, cdi, dgh, dgi$	2	bcd

Fig. 2. The formal concepts and their closures, minimal generators and succinct system of minimal generators in TDB of Figure 1.

the attributes covered by the formal concept. Mining the intents of concepts or closed itemsets has attracted a lot of attention (e.g., [9, 10, 8, 12, 15, 14]) for their importance in knowledge discovery, and for the significant reduction in the number of necessary frequent itemsets achieved by removing redundant (recoverable) ones.

Each formal concept actually corresponds to a set of itemsets, which are all equivalent since they capture the same intent. While the closures are the maximal sets of attributes/items presenting the concept, it is often interesting to ask, "*What are the critical combinations of attributes that manifest the concept?*" That is, for a concept, we want to identify the minimal combinations of attributes—the so-called minimal generators—that distinguish the objects in this concept from the others. Such minimal generators can offer a complementary, perhaps simpler way to understand the concept, because they may contain far fewer attributes than closed itemsets.

Technically, the *minimal generators* of a formal concept C are the minimal subsets of C's intent that can characterize C, and are the lower bounds of the itemsets characterizing C [10, 12]. For the running example, itemsets bc, bg, ch, ci, gh, gi are the minimal generators of formal concept $bcghi$, since any transaction containing any of those minimal generators must also contain the other items in the closure.

Complementary to closures, minimal generators provide an important way to characterize formal concepts. However, very little has been done on understanding and mining the minimal generators. Some previous studies (e.g., [10, 15]) use minimal generators only as a means to achieve other goals such as mining closed itemsets. [12] considers the mining of *all* minimal generators, but its algorithm leaves considerable room for improvement.

Interestingly, the minimal generators still may contain a lot of redundant information. Consider the formal concepts in Figure 2. Closed itemset $bcghi$ has six minimal generators: bc, bg, ch, hg, ic and ig. From any one of them we can derive all the others, since b, h and i always appear together in transactions and are thus equivalent, and similarly for c and g. Those facts are indicated by formal concepts bhi and cg, respectively.

Can we remove the redundant information and achieve a succinct representation of the minimal generators? In this paper, we propose a novel concept of *succinct system of* *minimal generators* (SSMG for short). The idea is to remove the redundant information by choosing one (e.g., the lexically smallest) minimal generator of a formal concept as its representative minimal generator, and exclude non-representative minimal generators of the concept to occur as parts of minimal generators of any other concepts.

For example, we can choose b as the representative minimal generator for the formal concept bhi, and c for cg. For the concept $bcghi$ only the minimal generator bc will be included in the SSMG; all the other five (i.e. bg, ch, hg, ic and ig) are excluded and can be derived. Using SSMG there are a total of 17 minimal generators (Figure 2), compared with a total of 38 standard minimal generators. This big reduction in size causes no loss of information, as all minimal generators can be inferred from the SSMG.

Using the SSMG, the same equivalence information between the minimal generators of a concept will not occur redundantly. This helps reduce the result of mining and make it easier to browse, understand and manage, and reduce the need for the user to digest the same information multiple times and hence helps the user to concentrate on the new equivalence among minimal generators. Since the results on the equivalence among the minimal generators also reveal the minimal equivalence relation among associations and itemsets, results on SSMG are also useful for association mining.

In this paper, we give an efficient algorithm for mining SSMGs. Our algorithm is substantially more effective and efficient than the algorithm in [12], which mines all minimal generators. While the problem of mining SSMGs is computationally expensive, our experiments demonstrate that our algorithm can deal with high dimensional and large real data sets. We will also illustrate the power of our method on real data sets in terms of both effectiveness and efficiency. It should be noted that the SSMG mining is significantly more involved than the closed itemset mining, since it provides information on all the minimal generators in addition to the closed itemsets. We applied the algorithm on some real data sets and obtained some some interesting findings. But the details are omitted due to space limit.

Section 2 provides definitions of SSMG. Section 3 describes the algorithm. Section 4 reports experimental results on effectiveness and efficiency. Section 5 discusses related works and potential extensions.

2 Definition of SSMG

After revisiting the preliminaries of formal concepts, this section introduces the notion of succinct system of minimal generators (SSMG).

2.1 Preliminaries. Let $I = \{i_1, \ldots, i_n\}$ be a set of *items*. An *itemset* is a subset of I. A *transaction* is a tuple $\langle tid, X\rangle$, where tid is a transaction identity and X is an itemset. A *transaction database* TDB is a set of transactions. A transaction $\langle tid, X\rangle$ is said to *contain* itemset Y if $Y \subseteq X$. Let TDB be a given transaction database. The *support* of an itemset X, denoted as $sup(X)$, is the number of transactions in TDB that contain X. Given a *minimum support threshold* min_sup, X is *frequent* if $sup(X) \geq min\_sup$.

The *transaction set* of an itemset X, denoted as $\mathcal{T}(X)$, is the set of all transactions in TDB that contain X. For our running example (Figure 1), $\mathcal{T}(bc) = \{T_1, T_3, T_5\}$. Two itemsets X and Y are called *equivalent*, denoted as $X \sim Y$, if $\mathcal{T}(X) = \mathcal{T}(Y)$. The *equivalence class* of an itemset X is the set of all itemsets that are equivalent to X. For Figure 1, itemsets bc and gi are equivalent since $\mathcal{T}(bc) = \{T_1, T_3, T_5\} = \mathcal{T}(gi)$;

the equivalence class of b is $\{b,\ h,\ i,\ bh,\ bi,\ hi,\ bhi\}$. Symmetrically, the *itemset* of a set of transactions $D \subseteq TDB$, denoted as $\mathcal{I}(D)$, is the set of items that appear in every transaction in D, i.e., $\mathcal{I}(D) = \cap_{(tid,X)\in D} X$. In Figure 1, $\mathcal{I}(\{T_1, T_3, T_5\}) = bcghi$.

An itemset X is call *closed* if there exists no proper superset $X' \supset X$ such that $sup(X) = sup(X')$. Easily, we can show that an itemset X is closed iff $\mathcal{I}(\mathcal{T}(X)) = X$. Symmetrically, a set of transaction D is *closed* if and only if $\mathcal{T}(\mathcal{I}(D)) = D$.

Definition 1. A *formal concept* is a pair $C = (X, D)$ where X and D are a closed itemset and a closed transaction set, respectively, such that $D = \mathcal{T}(X)$ and $X = \mathcal{I}(D)$. Given two concepts $C = (X, D)$ and $C' = (X', D')$, C is said to be *more general* than C' if $X \subset X'$.

Under the set containment order, the itemsets form a lattice L. Moreover, under the same order on the closed itemsets, the formal concepts form a lattice $\mathcal{L}_{\mathcal{C}}$, which is a *Galois lattice* (see Figure 3). Apparently, the lattice of the formal concepts is a quotient lattice with respect to $\mathcal{L}$, i.e., $\mathcal{L}_{\mathcal{C}} = \mathcal{L}/\sim$.

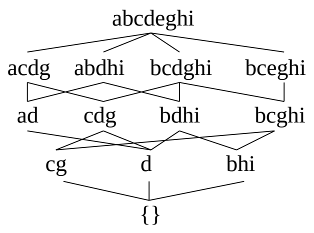


Fig. 3. Galois lattice for TDB of Figure 1.

Observe that each equivalence class of itemsets contains a unique closed itemset, which serves as the upper bound for the equivalence class. Also, each class contains one or more lower bounds, which are the minimal generators. For example in Figure 1, b, h and i are the minimal generators of formal concept $(bhi, \{T_1, T_3, T_4, T_5\})$ (Figure 2).

Definition 2. An itemset Y is called a *minimal generator* for a formal concept (X, D) if $\mathcal{T}(Y) = D$ but for every proper subset $Y' \subset Y$, $\mathcal{T}(Y') \neq D$.

2.2 Succinct System of Minimal Generators. As discussed earlier, the minimal generators may still contain a lot of redundant information. Consider again the formal concept $C = (bcghi, \{T_1, T_3, T_5\})$ in our running example. In the same database, there is another concept $C_1 = (cg, \{T_1, T_2, T_3, T_5\})$. C_1 is more general than C, and C_1 has c and g as minimal generators. We can observe the following: If itemset cX is a minimal generator for C such that $g \notin X$, then gX is also be a minimal generator for C. This can be verified from Figure 2. Since we have bc, ch and ci as the minimal generators containing c but no g, we also have minimal generators bg, gh and gi. So, we only need to keep either the minimal generators containing c or those containing g, but not both; the rest can be inferred.

Can we have a *non-redundant* representation of the minimal generators? The answer is yes. The idea is intuitive though not straightforward. To illustrate the general idea, suppose a user wishes to browse the minimal generators of the formal concepts in the coarse-to-fine (or general-to-specific) order[2]. For each new concept C to be browsed, we would like to present to the user a minimal but complete set of all *new* minimal generators that cannot be inferred from the others for this concept C and from the minimal generators of the more general formal concepts already browsed.

Formally, for each formal concept C, we need to define a new equivalence relation: Two itemsets X and Y are C*-equivalent*, denoted $X \approx_C Y$ if (i) both X and Y are minimal generators of formal concept C' such that C' is more general than C, or (ii) X can be obtained from Y by replacing a subset $Z \subset X$ with $Z' \subset Y$ such that $Z \approx_C Z'$.

In our running example, if C is the formal concept whose closed itemset is $bcghi$, then $b \approx_C h$, $b \approx_C i$, $c \approx_C g$, $bc \approx_C bg$, $bc \approx_C hg$, etc. (Note that C has two more general formal concepts.) If C is the formal concept whose closed itemset is bhi, then $X \approx_C Y$ if and only if $X = Y$ since there are no concept more general than C.

The $\approx_C$ equivalence relation partitions C's minimal generators into equivalence classes. We can achieve the goal of deriving a minimal non-redundant subset of minimal generators by presenting one minimal generator for each of the equivalence classes. For example, if C is the formal concept whose closed itemset is $bcghi$, then all of its six minimal generators, namely bc, bg, ch, ci, gh, gi, belong to one equivalence class; if C is the formal concept whose closed itemset is $abcdeghi$, then its minimal generators can be partitioned into three equivalence classes: $\{ae\}$, $\{de\}$, $\{abc, abg, ach, aci, agh, agi\}$. Then, we can choose one representative for each class of minimal generators.

Which member of an equivalence class should be shown to the user, in order to minimize the overall overhead on the user? We choose one minimal generator of each formal concept as its *representative* minimal generator. This can be done freely for most basic formal concepts such as bhi, cg and d in Figure 3. To be succinct, for other concepts, we should choose one of those *canonical* minimal generators X such that X does not contain any non-representative minimal generators of more general formal concepts. For example, if C is the formal concept whose closed itemset is $bcghi$, if b and c are respectively the representative minimal generators for the concepts for bhi and cg, then bc should be the representative of C.

Definition 3. A *succinct system of minimal generators*, or SSMG for short, consists of, for each formal concept $C = (X, D)$, a representative minimal generator and a set of canonical minimal generators.

An SSMG will remove the redundancy of minimal generators, give the users a consistent handle on each class using the representative minimal generators, and also can be used to derive all the minimal generators. The last column of Figure 2 gives an SSMG for our running example TDB, where the first minimal generators are the representatives for the concepts of the corresponding rows. Given an SSMG, clearly we can reconstruct all of the minimal generators. Also, the SSMG is not unique, even though different SSMGs have the same number of minimal generators.
Problem definition. Given a transaction database TDB and a support threshold min_sup, the problem of *mining the succinct system of minimal generators* is to find a succinct system of minimal generators for all formal concepts $C = (X, D)$ that $sup(X) \geq min\_sup$.

[2] In fact, our definitions can deal with any order of browsing.

3 The SSMG-Miner Algorithm

This section introduces our algorithm for mining SSMGs. It includes several novel techniques for computing local minimal generators and closed itemsets in a depth-first manner, and for using them to derive the SSMGs. While the high-level structure of the algorithm is similar to many existing DFS based algorithms, the new algorithmic contributions lie with the efficient techniques for producing representative minimal generators and removing the non-representative ones.

3.1 Depth-First Search Framework. The SSMG-Miner algorithm follows the general depth-first search framework that can be described using a depth-first search tree (e.g. *set-enumeration tree* (or SE-tree) [13]). The SE-tree enumerates all possible itemsets for a given set of items, with a global order on the items. For each node v in the tree we have a head H (consisting of items considered so far), and a tail T (consisting of items to be considered among descendant nodes). The search space associated with v consists of all itemsets of the form $Z = H \cup T'$, where T' is a nonempty subset of T. For the node labelled by ab in the SE-tree for $\{a, b, c, d\}$, we have $H = ab$ and $T = cd$, and its search space consists of abc, abd, and $abcd$. The algorithm will remove useless branches of the SE-tree, as discussed later.

3.2 Computing Local Minimals/Closures. The SSMG-Miner will efficiently compute "local minimal generators and closed itemsets" for each visited node in the depth-first search. Later we will show that some local minimal generators and closed itemsets may not be true minimal generators and closed itemsets for formal concepts, and consider efficient techniques to remove such itemsets.

In our DFS computation, for each node v with head H and tail T, those items x in T such that $\mathcal{T}(Hx) = \mathcal{T}(H)$ (or equivalently, $sup(Hx) = sup(H)$, in other words, item x appears in every transaction that contains H) are in the local closed itemsets, and are removed from T. Let the *local closure* of H be $LC(H) = \{x \in H \cup T \mid \mathcal{T}(H) = \mathcal{T}(Hx)\}$. The removal of items from T as described above will ensure that, for all ancestor nodes v' of v with head H' and tail T', $LC(H')$ is a proper subset of $LC(H)$. Hence H is considered as the *local minimal generator* for $LC(H)$.

We now illustrate the local minimal generators and closures computed for 6 nodes, using our running example (Figure 1). (1) At the root node, $H = \emptyset$, $T = abcdeghi$, $LC(H) = \emptyset$. (2) For the first child of the root, $H = a$ and $T = bcdeghi$; since $\mathcal{T}(a) = \mathcal{T}(ad)$, we remove d from T (so T becomes $bceghi$; this node now has 6 children instead of the original 7); $LC(H) = ad$; a is the local minimal generator for ad and ad is the local closure for a. (3) For the node with $(H, T) = (ab, ceghi)$, $LC(ab) = abdhi$ with $sup(ab) = 2$. (4) For $(H, T) = (abc, eg)$, $LC(abc) = abcdeghi$ with $sup(abc) = 1$. (5) For $(H, T) = (abe, g)$, $LC(abe) = abdeghi$ with $sup(abe) = 1$. (6) For $(H, T) = (ae, \emptyset)$, $LC(ae) = aeghi$ with $sup(ae) = 1$.

The SSMG-Miner algorithm keeps a tuple of the form $(MinList : Max, Count)$ for each formal concept, where $MinList$ is the list of minimal generators, Max is the closed itemset, and $Count$ is the support count. The first minimal generator in $MinList$ is the representative minimal generator. For Figures 1 and 2, the tuple $(b, h, i : bhi, 4)$ is for the formal concept $(bhi, \{T_1, T_3, T_4, T_5\})$.

3.2 Determining Equivalence. The local minimal generators and closures computed at different nodes may belong to the same formal concept. The SSMG-Miner will check on this and remove any redundancy.

Lemma 1. *Let v be a node with head H and tail T. Then $LC(H)$ belongs to an existing formal concept at the time v is visited if and only if there is a node v' visited before, with head H' and tail T', such that $LC(H) \subset LC(H')$ and $sup(H) = sup(H')$.*

If one does the equivalence check based on the above lemma, the check will be inefficient. The reason is that, for each new node v with head H and tail T, we will need to go through all existing formal concepts and conduct the subset checking based on the support equivalence. In general, checking whether an itemset is a subset of another itemset in collection of itemset is very expensive.

Lemma 2. *Let v be a node with head H and tail T. Then $LC(H)$ belongs to an existing formal concept at the time v is visited if and only if there is a node v' with head H' and tail T' such that v' is visited before v and v' satisfies the following three conditions: (1) $sup(LC(H)) = sup(LC(H'))$; (2) $LC(H)$ and $LC(H')$ share a common suffix starting from x, where x is the last item of H; (3) the prefix of $LC(H))$ before x is a subset of the prefix of $LC(H')$ before x, where x is as above. Here, the values of T and T' are those when the nodes are created.*
Rationale. Clearly the "if" holds, since conditions (1–3) imply that $LC(H) \subset LC(H')$ and $sup(H) = sup(H')$, which in turn imply that $LC(H)$ and $LC(H')$ are subsets of some common closed itemset. "Only if": Suppose $LC(H)$ belongs to an existing formal concept at the time v is visited. Let v' be the node when $LC(H)$'s formal concept is first inserted; let H' be its head and T' its tail. Since $LC(H)$ and $LC(H')$ belong to the same concept, condition (1) holds. Since v' is the first node when $LC(H)$'s formal concept is inserted, by the nature of DFS computation, we have $\{y \mid y \in H$ and y is before $x\} \subseteq \{y \mid y \in H'$ and y is before $x\}$, and so (3) holds. This implies (2) holds.

This lemma allows us to efficiently implement the check using some comparison on the support counts, and certain suffixes and prefixes of itemsets and the local closure.

Example 1. We illustrate by considering these three formal concepts for example in Figures 1 and 2: $(a : ad, 3)$, $(ab : abdhi, 2)$ and $(abc : abcdeghi, 1)$. For the node with $H = abe$ and $T = g$, $LC(abe) = abdeghi$ and $sup(abe) = 1$. We need to decide if this is a new formal concept and, if not new, which existing concept is that of abe. We do this as follows: We look for (1) concepts with the same support count as abe, and we compare their closed itemsets against $LC(abe) = abdeghi$. The concept C of abc is the only such concept. We note the following: (2) The closed itemset of C, namely $abcdeghi$, and $LC(abe) = abdeghi$ share the common suffix of $eghi$, starting at the item e (the last item of abe). (3) The prefix of $LC(abe) = abdeghi$ before e, namely abd, is a subset of the prefix of $abcdeghi$ before e. Lemma 2 ensures that, when this happens, abe is not generating a new formal concept, but abe is another potential generator for the concept of C.

On the other hand, if there is no concept C satisfying the conditions, then the new local minimal generator and closure form a new formal concept. Consider the formal concept $(bd : bdhi, 3)$ and we compute $LC(cd) = cdg$ and $sup(cd) = 3$ at the node with $H = cd$. We conclude that cd and bd do not belong to any previously found formal concept, since $LC(cd)$ is not a subset of any closed itemsets of existing concepts with the same support.

More specifically, Lemma 2 implies that equivalence checking can be accomplished efficiently by using a search tree structure. In such a tree, the items are ordered under the *reverse of the original order* on the items. We have one such tree for each support count. The trees will be built in a lazy manner. For each formal concept C, we use the closed itemset for C to search and insert. This is done similarly for local closures computed at nodes. For example, if the original order of items is the alphabetical order, for the closed itemset $abcdeghi$, we have a branch of $i \rightarrow h \rightarrow g \rightarrow e \rightarrow$ ... For the search involving $LC(abe) = abdeghi$, we follow the branch $i \rightarrow h \rightarrow g \rightarrow e$. Then we go through the formal concepts stored below this branch to check for containment of the prefixes. Since the search of the suffix only needs to continue if exact match is found and can be terminated as soon as a mismatch is found, it is very efficient.

3.3 Removing Non-Minimal Generators and Clutters. Some local minimal generators computed in the DFS process may turn out to be not minimal generators for their formal concepts. Also, the clutters caused by redundant minimal generators need to be removed. We now discuss how SSMG-Miner handles these issues.

To show the removal of non-minimal generators, let us examine the running example again (Figures 1 and 2). For node $H = ae$, we have four formal concepts computed: $(a : ad, 3)$, $(ab : abdhi, 2)$, $(abc, abe, abg, ace, ach, aci : abcdeghi, 1)$, $(ac : acdg, 2)$. We find that ae is a new minimal generator for the concept of $abcdeghi$. Since abe and ace are supersets of ae, they are not true minimal generators for their formal concept, and should be removed. So the third formal concept becomes $(abc, abg, ach, aci : abcdeghi, 1)$, before ae is inserted. Since ae is earlier than abc in the "cognitive-order", we select ae to replace abc as the representative minimal generator.

To exemplify the removal of clutters, let us consider the running example (Figures 1 and 2). Suppose our current set of formal concepts are as given above, and we next consider the node $H = ag$. We find that $LC(ag) = adg$. We see that adg and $acdg$ are equivalent, hence ag is the second minimal generator of $acdg$. We then remove all minimal generators of other concepts which contain ag (the redundant generators). For example, abg is removed from the set of minimal generators of $abcdeghi$. So we get the following formal concepts: $(a : ad, 3)$, $(ab : abdhi, 2)$, $(ae, ach, aci, abc : abcdeghi, 1)$, $(ac, ag : acdg, 2)$.

Regarding implementation, for each formal concept we have a concept identifier CID. For each item x, we have an inverted list consisting of all those formal concepts that have one or more minimal generators containing x. These inverted lists will be used to locate formal concepts that may contain a given itemset (minimal generator).

3.4 The Pseudo-Code of SSMG-Miner. The SSMG-Miner (Figure 4) calls the DFS function for the root node, with these three arguments: $H = \emptyset$, $T = I - LC$, and $LC = \emptyset$, where I is the set of all items.

The DFS function first determines if H meets the minimal support threshold. If the answer is yes, it will move all those items x such that $sup(H) = sup(H \cup \{x\})$ from T to LC. At this time, $(H, LC, sup(H))$ becomes a candidate new concept. If LC is not equivalent to any current concept, then it inserts $(H, LC, sup(H))$ as a new concept. Otherwise it inserts H as a new minimal generator of its concept, and removes the clutters. The check regarding equivalence, the insertion of new minimal generators and the removal of clutters are discussed in the previous subsections. Limited by space, their pseudo-code is omitted. DFS calls itself for each child node of the current node.

```
ALGORITHM SSMG_MINER:
INPUT: A transaction database $TDB$, support threshold min_sup.
OUTPUT: Succinct system of minimal generators for formal concepts in $TDB$.
METHOD:
  let $SSMG = \emptyset$; // SSMG is a global variable;
  let LC = {items occurring in all transactions};
  call DFS($H = \emptyset, T = I - LC, LC$);
  return SSMG;

Function DFS(H,T,LC) // $H$: head, $T$: tail
// $LC$: local closure, with value of parent node initially
  if $sup(H) < min\_sup$ return;
  for each $x \in T$
    if $sup(H \cup \{x\}) = sup(H)$
        let $T = T - \{x\}, LC = LC \cup \{x\}$;
    if $(H : LC, sup(H))$ is a new concept
        add $(H : LC, sup(H))$ to SSMG;
    else add $H$ as minimal generator for $LC$ and remove clutter
  for each $x$ in $T$
    let $H_x = H \cup \{x\}$ and $T_x = \{y \in T \mid y > x\}$;
    call $DFS(H_x, T_x, LC)$;
```

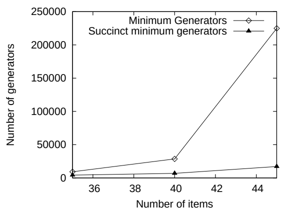
Fig. 4. Algorithm SSMG-Miner.

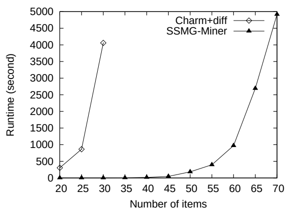
4 Performance Study

We now report experiments on the performance of the SSMG-Miner algorithm and its effect in reducing the amount of redundant information. Experiments show that the algorithm can deal with fairly high dimensional data sets within a short time. We also provide comparison with previous work as much as we could, and with a post-processing approach. All experiments (unless indicated otherwise) were performed on a PC with P4 2.4G CPU and 512M main memory, running on Windows XP.

We used two data sets in our efficiency experiments. (1) The Mushroom data set has been frequently used for evaluating data mining algorithms and is obtained from the UCI Machine Learning Repository. It includes 22 attributes and $8,124$ tuples. There are a total of 121 attribute-value pairs (items). (2) The Colon tumor gene expression data set is from [2]. It consists of micro-array gene expression data for 62 sample tissues, with 22 being normal tissues and 40 colon tumor tissues. Microarrays are a technology for simultaneously profiling the expression levels of tens of thousands of genes in a patient sample. It is increasingly clear that better diagnosis methods and better understanding of disease mechanisms can be derived from a careful analysis of microarray measurements of gene expression profiles. As with most association-type data mining, we discretized each gene into two intervals: low and high. We also used the entropy method to select the top 45 most "relevant" genes from total of 2000 genes. These data sets are typical examples of data that might be used in scientific discovery process by data mining techniques. Please also note that these two data sets are quite dense and thus challenging to mine, as indicated by many previous studies.

4.1 Redundant Information Reduction. Figure 5 shows the succinct minimal generator concept leads to a huge reduction in the number of minimal generators in the result





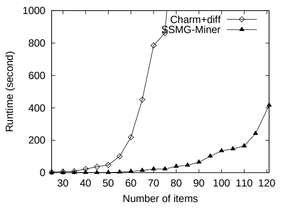


Fig. 5. Reduction of # of generators on Colon Cancer data set (support threshold = 1%).

Fig. 6. Scalability on number of items: Colon Cancer data set (support threshold=1%).

Fig. 7. Scalability on number of items: Mushroom data set (support threshold=1%).

of mining. The Colon data set is used. For the case of 40 items, 76% of the minimal generators are redundant, and for 45 items 93% of the minimal generators are redundant. The reduction on Mushroom is similar (the details are omitted due to space).

4.2 Comparison with Postprocessing Approach. Post processing seems to be much worse than the SSMG-Miner, even though we do not ask the postprocessing algorithm to remove the reduntant minimal generators. For example, we compared with the postprocessing approach which combines the Charm algorithm [15] for closed itemset mining, and the Border-Diff algorithm [6] for mining the minimal generators from the closed itemsets. First, the Charm algorithm is used to compute the closed itemsets satisfying given support threshold. (We used our own implementation of the Charm algorithm.) Then, for each closed itemset X, let $S_X = \{Y \mid Y \subset X$ and Y is a closed itemset$\}$; then the Border-Diff algorithm is called to mine the minimal itemsets which occur in X but not in any itemset in S_X. Let $M_1, ..., M_k$ be the result of this operation. It can be verified that $M_1, ..., M_k$ are the minimal generators for the class represented by X. It turns out that this algorithm is very expensive: On the Colon data, on 20, 25 and 30 projected columns of the data, SSMG-Miner used 1, 1 and 2 seconds respectively, whereas post-processing used 305, 964, and 4090 seconds respectively. The main cost of the Charm+Border-Diff algorithm is due to the large number of calls fo Border-Diff.

4.3 Comparison with a Previous Algorithm. No previous work has considered the mining of succinct minimal generators. Some prior work considered the mining of all minimal generators [12]. (Several papers considered the mining of closed itemsets, and perhaps with one minimal generator for each closed itemset.) We contacted the authors of [12], but we cannot obtain either executable or source code. We are able to provide a rough comparison as follows: For the Mushroom data when all attributes are considered and the minimal support is set at 1 (so that all itemsets are frequent), our algorithm finished in about 2 hours on our machine. On the other hand, the algorithm of [12] used about one day and a half. Although the configuration of the test platform is not given in [12], we believe that our method is substantially faster. We should also note that the algorithm of [12] does not remove clutters.

4.4 Scalability. Figures 6 and 7 show the computation time of SSMG **as the number of items varies**. The minimum support is set at 1% and the different number of items is obtained by projecting the original data set over the first k items. Although the processing time increases exponentially with the number of items, it is encouraging to know that our algorithm finishes in a reasonable amount of time. We use random subsets of the Mushroom data to test the **scalability on size of database**. Figure 8 shows the compu-

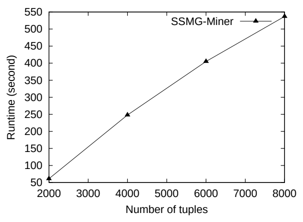


Fig. 8. Scalability on database size: Mushroom data set (support threshold=1%).

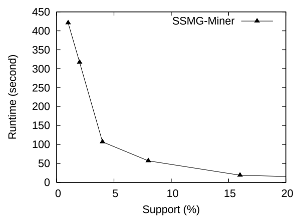


Fig. 9. Scalability on support threshold: Mushroom data set.

tation time vs. the number of instances for the Mushroom data. The computation time is roughly linear as the number of instances (tuples) increases. Figure 9 shows how computation time varies **as the support threshold varies** on the Mushroom data. As the support threshold decreases, the number of minimal generators increases, leading to increased computation time.

Summary. From the extensive experiments on the two real data sets, the effectiveness and efficiency of SSMG-Miner are verified. Our results show that SSMG-Miner is feasible for mining real data sets.

5 Related Work and Discussion

5.1 Related Work. Formal Concept Analysis (FCA) was first pioneered by Wille in 1982 [7], and has grown into an active field for data analysis and knowledge discovery. Other previous research most related to our work can be divided into two categories: mining closed itemsets and mining minimal generators.

Closed itemset mining is one of the major classes of research addressing the frequent itemset mining problem [1]. This class of research aims at mining a concise subset of the frequent itemsets that can be used to derive all other frequent itemsets and their support counts. A major approach considers the closed itemset mining problem, initially proposed in [10], where one mines only those frequent itemsets having no proper superset with the same support. Mining closed itemsets can lead to orders of magnitude smaller result set [16] (than mining all frequent itemsets) while retaining the completeness, i.e., the concise result set can be used to generate all the frequent itemsets with correct support counts in a straightforward manner. In the last several years, extensive studies have proposed fast algorithms for mining frequent closed itemsets, such as Aclose [10], CLOSET [11], MAFIA [4], CHARM [15], and CLOSET+ [14].

While prior research considered closed itemsets, they paid little or no attention to **mining minimal generators**. Minimal generators were only used as a means to achieve other goals if they were considered. The algorithm of [10] focused on the mining of closed itemsets, but in the computation process it produces one minimal generator as a by-product. The non-derivable and free itemsets [5, 3] are related to minimal generators.

Reference [12] gave an algorithm to compute the closed itemsets and their minimal generators incrementally (by inserting tuples one at a time). However, it does not consider the removal of the redundant minimal generators.

5.2 Further Extensions. Our method can be extended in several aspects, including these three: (1) We can analyze the SSMGs and their relationship with their corresponding closed itemsets. We can also analyze the SSMGs for data with multiple classes, such as normal tissues and cancer tissues for colon cancer discussed above. (2) We can consider SSMGs for approximate formal concepts, as a generalization of "exact" equivalent classes. We can view itemsets as approximately equivalent if their transaction sets are approximately equal. This will help reduce the number of formal concepts significantly. We can also analyze the SSMGs of approximately identical formal concepts. (3) We conjecture that the following numbers can be used as indicators of the structure of the data set under consideration: the number of formal concepts, the number of formal concepts with multiple minimal generators, and the reduction ratio from number of minimal generators to succinct minimal generators.

Acknolwedgement: We thank Ravi Janga who helped with the coding of the Charm-BDiff algorithm, and the reviewers of a previous version of this paper.

References

1. R. Agrawal, et al. Mining association rules between sets of items in large databases. In *SIGMOD'93*.
2. U. Alon, et al. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissue s probed by oligonucleotide arrays. *Proc. Nat. Academy of Sciences of the United States of American*, 96:6745–675, 1999.
3. J-F. Boulicaut, et al. Free-sets: A condensed representation of boolean data for the approximation of frequency queries. *Data Mininig and Knowledge Discovery*, 7(1):5–22, 2003.
4. D. Burdick, et al. MAFIA: A maximal frequent itemset algorithm for transactional databases. In *ICDE'01*.
5. T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. In *PKDD'02*.
6. G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *KDD'99*.
7. B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, Heidelberg, 1999.
8. J. Hereth, et al. Conceptual knowledge discovery and data analysis. In *Int. Conf. on Conceptual Structures*, pages 421–437, 2000.
9. G. Mineau and B. Ganter, editors. *Proc. Int. Conf. on Conceptual Structures*. LNCS 1867, Springer, 2000.
10. N. Pasquier, et al. Discovering frequent closed itemsets for association rules. In *ICDT'99*.
11. J. Pei, et al. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD DMKD'00*.
12. J.L. Pfaltz and C.M. Taylor. Closed set mining of biological data. In *BIOKDD'02*.
13. R. Rymon. Search through systematic set enumeration. In *Proc. of Int'l Conf. on Principles of Knowledge Representation and Reasoning, Cambridge MA*, pages 539–550, 1992.
14. J. Wang, et al. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *KDD'03*.
15. M. Zaki and C. Hsiao. CHARM: An efficient algorithm for closed itemset mining. In *SDM'02*.
16. M. J. Zaki. Generating non-redundant association rules. In *KDD'00*.