NUCLEAR: AN EFFICIENT METHOD FOR MINING FREQUENT ITEMSETS BASED ON KERNELS AND EXTENDABLE SETS

Huy Quang Pham[1, 2], Duc Tran[3], Ninh Bao Duong[2], Philippe Fournier-Viger[4], Alioune Ngom[1]

[1]University of Windsor, Windsor, Ontario, Canada
[2]University of Dalat, Dalat, Vietnam
[3]Faculty of Information Technology, Ho Chi Minh City University of Food Industry, Vietnam
[4]School of Natural Sciences and Humanities, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, 518055, China

ABSTRACT

Frequent itemset (FI) mining is an interesting data mining task. Directly mining the FIs from data often requires lots of time and memory, and should be avoided in many cases. A more preferred approach is to mine only the frequent closed itemsets (FCIs) first and then extract the FIs for each FCI because the number of FCIs is usually much less than that of the FIs. However, some algorithms require the generators for each FCI to extract the FIs, leading to an extra cost. In this paper, based on the concepts of "kernel set" and "extendable set", we introduce the NUCLEAR algorithm which easily and quickly induces the FIs from the lattice of FCIs without the need of the generators. Experimental results showed that NUCLEAR is effective as compared to previous studies, especially, the time for extracting the FIs is usually much smaller than that for mining the FCIs.

KEYWORDS

Association Rule, Kernel and Extendable Set, Frequent Itemset, Frequent Closed Itemset, Nuclear.

1. INTRODUCTION

Mining association rules (ARs) [1] is one of the most interesting and popular problems in data mining. It is widely used for decision making in retail, e-commerce, medicine, and many other domains. Mining frequent itemsets (FIs) is the first and the main step in the discovery of ARs. Since its first introduction in 1993 [1] it has attracted a lot of attention and has been extended and applied in various ways. For instance, some popular variations of the FI mining problem are to discover high utility patterns [2, 3], uncertain frequent patterns [2] and high utility association rules [2, 4]. Most algorithms for mining FIs partition the search space into subclasses in order to apply the parallel approaches to improve their performance. However, the performance of many parallel FI mining algorithms is limited by the speed of disk accesses, as they repeatedly scan the input database, which can still lead to long execution [5, 6]. To address this issue, some researchers proposed more efficient parallel algorithms, which compress the database in a frequent pattern tree and perform tree projections [7, 8]. Another approach to speed up the FI mining process is to first mine all the frequent closed itemsets (FCIs) and then derive the FIs from them without the need of rescanning the data file. This approach is more efficient than mining FIs directly because the number of FCIs is usually much less than that of FIs (see Table 4). Charm [9], FPClose [7], DCI_PLUS [10] and NAFCP [6] are among the best algorithms for mining FCIs. In 2010, a parallel algorithm (PLCMQS) for mining FCIs has been proposed [11]. The authors of Charm proposed the CharmL algorithm [8], which builds the lattice of FCIs. Formal concept (i.e., lattice of FCIs) analysis is also another way of mining FIs as well as ARs [6, 12, 13].

Mining FIs from the lattice of FCIs has several advantages over mining FIs directly from data. First, the number of FCIs is often much smaller than the number of FIs; therefore, this requires less memory. Second, each FCI can stand for an equivalence class of FIs having the same closure (i.e., these FIs shares the same set of transactions containing them); thus, we can develop parallel algorithms or "divide-and-conquer" approaches to facilitate the process of deriving FIs from FCIs. Third, when users want to try different minimum support (*minsup*) thresholds to find an optimal set of FIs for a certain downstream procedure, the cost of updating the set of FCIs can be much lower than mining them from scratch in a database. However, to extract to FIs from FCIs, the current algorithms requires the generators for each FCI, and mining them might take a significant amount of time.

In this paper, we present a method to mine the FIs from a lattice of FCIs without the need of the generators. We introduce the concepts of "kernels" and "extendable sets" which further partition the equivalence classes represented by the FCIs into smaller subclasses. Then, each pair of kernel and extendable set stands for a subclass of FIs which are supersets of the kernel and subsets of the union of the kernel and the extendable set. Thus, once a pair of kernel and extendable set are identified, enumerating FIs in the subclass is straightforward. Our proposed algorithm, called NUCLEAR, to generate kernels and extendable sets for each FCI is simple and efficient. Its inputs are just the largest FCIs those are subsets of that FCI, and the time for it to induce all FIs from the lattice of FCIs is significantly shorter than the time to construct the lattice.

The fact that NUCLEAR does not require the generators makes it more efficient than the approach infer FIs from FCIs and generators. In the comparison of NUC, the approach using NUCLEAR, against dEclat [8, 30], a well-known algorithm which mines FIs directly from data, NUC is faster when the number of FIs is much larger than the number of FCIs. When NUC is slower, the reason is that CharmL, the algorithm used to construct the lattice of FCIs from data before applying NUCLEAR, is already slower than dEclat.

The rest of this paper is organized as follows. Section 2 introduces some related concepts. Section 3 reviews the related works. Section 4 presents novel theoretical results that are the basis of the proposed algorithm, including a recurrent formula for generating kernels and extendable sets. Section 5 presents the proposed NUCLEAR algorithm. Section 6 reports experimental results that show the efficiency of the proposed algorithm. Finally, a conclusion is drawn and future work is discussed in section 7.

2. PRELIMINARIES

Consider a context (T, I, R) where I is a set of items (or attributes), T is a set of transactions (or objects) and R is a binary relation on $T \times I$. For each non-empty subset A of I and non-empty subset O of T, the two functions λ and ρ below define a Galois connection between 2^T and 2^I (reader can refer to [28] for more details):

$$\lambda: 2^T \rightarrow 2^I: \lambda(O) = \{a \in I \mid (o, a) \in R,\ \forall o \in O\},\ \lambda(\emptyset) = I$$
$$\rho: 2^I \rightarrow 2^T: \rho(A) = \{o \in T \mid (o, a) \in R,\ \forall a \in A\},\ \rho(\emptyset) = T.$$

Then, $h = \lambda_o\rho$ in 2^I is called the closure operator [33], and $h(A)$ is said to be the closure of A.
A set $A \subseteq I$ is called an itemset. An itemset A containing k items is called a k-itemset. Given a user-specified minimum support threshold *minsup*, such that $0 < minsup \leq 1$, the support of an itemset A is denoted and defined as $supp(A) = |\rho(A)| / |T|$. A is said to be "frequent" if $supp(A) \geq minsup$.
Itemset A is "closed" if and only if $h(A) = A$. Let $[A] = \{A' \subseteq I\text{: } h(A') = h(A)\}$ be the set of all itemsets having the same closure, which is $h(A)$, then, the itemsets in $[A]$ share the same set of transactions, which is $\rho(A)$, i.e., they have the same support.

Let *CS* and *FCS* denote the set of all closed itemsets and the set of all FCIs, respectively. Then, $L \equiv (FCS, \preccurlyeq_A)$ is a lattice of FCIs, where $\preccurlyeq_A$ is an order relation based on the $\subseteq$ operator between subsets of I.

An itemset G is called a "generator" of a closed itemset A if and only if $h(G) = h(A)$ and $\forall G'\text{: } \varnothing \neq G' \subset G \Rightarrow h(G') \subset h(G)$.

For any itemset $A \subseteq I$, the equivalence class $[A]$ has only one closed itemset, and one or more generators.

Example 1. Given $I = \{1, 2, 3, 4, 5, 6\}$, $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$, and R as in the Table 1. Let itemset $X = \{1, 4, 5\}$, then $\rho(X) = \{t_1, t_2\}$, $supp(X) = 2/4$, and $\lambda(\{t_1, t_2\}) = \{1, 4, 5, 6\}$. Then, X is not a FCI since $h(X) = \lambda_o\rho\ (X) = \{1, 4, 5, 6\} \neq X$. Now, let $C = \{1, 4, 5, 6\}$, we have $h(C) = \underline{C}$. Thus, C is a FCI. And, we have, $[X] = [C] = \{\{1, 4\}, \{1, 5\}, \{1, 6\}, \{1, 4, 5\}, \{1, 4, 6\}, \{1, 4, 5, 6\}\}$, in which, the sets $\{1, 4\}$, $\{1, 5\}$, $\{1, 6\}$ are the generators of C.

The lattice of FCIs mined from R for *minsup* = 0.25 (i.e., absolute minimum support = 1) is shown in Figure 1. In this lattice, each node (rectangle) represents a FCI and its absolute support, separated by a colon (":") and each edge links a FCI to the FCIs which are its largest subsets.

Table 1. The relation $R = T \times I$, where $I = \{1, 2, 3, 4, 5, 6\}$, $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$.

Transactions	Items					
t_1	1	2	3	4	5	6
t_2	1			4	5	6
t_3	1		3			
t_4		2	3	4	5	6

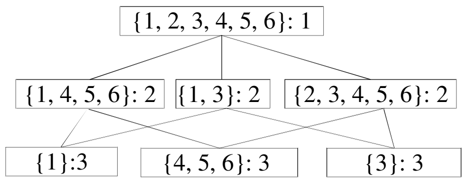

Figure 1. The lattice of FCIs mined from R with *minsup* = 0.25.

3. Related Work

3.1. Mining Frequent (Closed) Itemsets

In recent years, several algorithms have been proposed for FI mining such as dEclat, and Node-list-based algorithms [5, 14, 15, 16]. The dEclat was one of the most effective algorithms according to [17]. It scans a database once to generate the transaction sets (tidsets) for all itemsets of 1 item (1-itemset). Then, it applies the "diffset" strategy to enumerate all FIs without repeatedly scanning the database. Deng et al. [14] proposed a novel structure named N-list for mining FIs. The proposed algorithm first compresses the dataset into a PPC-tree structure, and then, using that tree, the algorithm generates N-lists for each 1-itemset. Finally, the algorithm applies a divide-and-conquer approach to mine FIs using these lists. Experimental evaluation has shown that the N-list-based algorithm outperforms state-of-the-art FI mining algorithms on a variety of real and synthetic datasets. Recently, Deng and Lv [15] proposed an improved N-list based frequent itemset mining algorithm named PrePost+, which applies a novel pruning strategy called children-parent equivalence pruning to reduce the search space. Subsequently, Vo et al. [16] combined the N-list structure with the subsume concept to further increase the performance of FI mining. Recently, Deng [14] proposed an efficient algorithm relying on an improved Nodeset structure, named DiffNodesets.

As defined in section II, the closure of an itemset is the set of items that appear in all transactions containing the itemset. FCIs have attracted a lot of studies as they can be used to partition FIs into equivalence classes. This inspires the development of parallel or “divide-and-conquer” approaches to mine FIs from FCIs without scanning the database for the support. However, few approaches have been proposed to perform this efficiently. Several researchers have studied retrieving FIs using the generator itemsets and eliminable itemsets in the equivalence classes of their closures [10, 17, 18-20]. For this purpose, algorithms were proposed that efficiently discover FIs using the lattice of FCIs, without performing duplicate checks, and by processing only one FCI at a time, that is, without considering its relationship to other FCIs. Generator itemsets can be mined independently or at the same time as FCIs. Zaki et al. [9] proposed the Minimal Generators algorithm to mine generators from the lattice of FCIs using a level-wise approach inspired by the Apriori algorithm. However, to identify the generators of a FCI, the algorithm had to scan all its subsets. Thus, the algorithm can be very slow. Szathmary et al. [21] proposed the Talky-G algorithm to mine generators from data, using an IT-tree structure. The algorithm uses the Charm algorithm [8, 9] to separately mine the FCIs and then, matches the generators with each FCI. Talky-G guarantees that when an itemset X is visited during the search, all its subsets have been already visited, and thus all generators that are subsets of X have already been found. Consequently, an itemset X is a generator if no already found generator is subset X and has the same support as X’s support. To quickly select the generators for that check Talky-G stores the support of visited generators in a hash table using the number of transactions containing each itemset as the hash function. This hash function is also used to match each FCI to its generators. The algorithm is effective when minsup is high. However, the time required for finding generators is similar to the time for mining FCIs. GENCLOSE [22] is an algorithm that concurrently mines FCIs and generators. The authors introduced necessary and sufficient conditions to generate generators (k + 1)-itemsets using generators k-itemsets. Using these conditions, the closure of each generator can be extended gradually to find generators. In 2005, the CHARM [9] and dCHARM [8] algorithms have been proposed for mining FCIs using the “diffset” structure introduced in the dEclat algorithm. In 2012, the DBV-Miner [23] algorithm improved this approach by compressing the tidsets of 1-itemsets using dynamic bit vectors. It was shown that this can greatly reduce the memory required for storing tidsets and compute the support of itemsets efficiently. Then, Sahoo et al. [10] proposed the DCI_PLUS algorithm for mining FCIs and their generators. The algorithm compresses the database using a BitTable structure, which is built using a single database scan. In [20], Tran et al. proposed the GEN_ITEMSETS algorithm to generate all itemsets from a lattice of FCIs and generators without repetitions. More recently, Le and Vo [12] proposed an N-list-based algorithm for mining FCIs, named NAFCP. The experimental evaluation of this work has shown that NAFCP outperforms state-of-art FCI mining algorithms in terms of runtime and memory usage in most cases.

B. Lattice-based Approaches for Mining Association Rules

In general, two types of lattices are considered for mining ARs, which are the frequent itemset lattice (FIL) and the frequent closed itemset lattice (FCIL) [12]. Vo and Le [13] have presented a lattice-based approach for building the FIL for mining ARs, here called FIL-2009. Each node of the structure used in FIL-2009 represents an itemset X and stores a tuple (*X, Tidset, Children*) where *Tidset* is the list of transactions containing *X* and *Children* are pointers to nodes representing supersets of *X*. Although, mining ARs using FIL-2009 is very effective, it is not designed for efficiently mining minimal non-redundant association rules. To address this issue, Vo and Le [24] extended the structure used by FIL-2009 (here called FIL-2011) by adding two fields in each node indicating if a node is a minimal generator or a frequent closed itemset, respectively. These values are determined during lattice construction. The structure is then used by FIL-2011 to effectively mine minimal non-redundant association rules. Thereafter, an efficient approach named PFIL was proposed, which supports incremental mining using the pre-large concept. It was shown that this approach is especially efficient for huge databases containing a large number of FIs [25]. The PFIL algorithm uses the diffset structure to quickly build a FIL. Then it uses the pre-large concept and diffset structure for maintaining the pre-large FIL.

For a given dataset and a *minsup* threshold, building the FCIL is generally much faster than building the FIL because the number of FCIs is usually much less than that of FIs. CharmL [8] is an effective algorithm to build the lattice of FCIs. To update the lattice, researchers have proposed an algorithm [5] that runs efficiently in the case of large databases with a small number of inserted transactions.

For parallel algorithms for mining FIs and ARs we refer readers to [26-29], and for the survey on algorithms FIs and ARs, we refer readers to [30, 31].

4. THEORETICAL RESULTS

In this section, we introduce theoretical results that are the basis of our proposed algorithm.
From now on, for convenience, whenever we use the variables *C*, and/or *G* without condition, it implicitly means that: $C \in$ CS, $\varnothing \neq G \subseteq C$ respectively. And, let ":" stand for "such that", and ","stand for the logical operator "$\wedge$" in the logical propositions.

Definition 1 (the immediately closed subsets of a closed itemset). *Let* $S_C = \{Y \in CS: (Y \subset C) \wedge (\nexists Z \in CS: Y \subset Z \subset C)\}$ *be the set of the largest closed itemsets that are subset of a given closed itemset C. These itemsets in* S_C *are called the immediately closed subsets of C.*

Proposition 1. $\forall G \subseteq C,\ \forall Y \in S_C\ (G \in [C] \Leftrightarrow \exists x \in G: x \notin Y)$.

Proposition 1 points out a way to find [*C*] by searching every itemset G satisfying the right-hand side of Proposition 1. However, it might be not efficient if we have to scan every subset of C.

Proof. For all $G \subseteq C$ and $Y \in S_C$,

"$\Rightarrow$": Since $G \in [C]$, we have $h(G) = C$. Now, assume that $\forall x \in G\ (x \in Y)$. Thus, $G \subseteq Y \subset C \Rightarrow h(G) \subseteq \text{h}(Y) = Y \subset C$. This leads to a contradiction: $h(G) \subset C$.

"$\Leftarrow$": Assume that $\forall Y \in S_C\ (\exists x \in G: x \notin Y) \wedge (G \notin [C])$. We have: $\forall Y \in S_C\ ((\text{h}(G) = \text{h}(Y) = Y) \vee (\text{h}(G) \neq Y)) \wedge (h(G) \subset C)$. By the definition of S_C, we have: $\forall X \subset C\ (X \in \text{CS} \Rightarrow \exists Y \in S_C: X \subseteq Y)$. Then, $\forall Y \in S_C\ (\text{h}(G) \subseteq Y)$. Thus, we have: $\forall Y \in S_C, \forall x \in G\ (x \in Y)$, which is a contradiction to the hypothesis.

Corollary 1. *$\forall Y_k \in S_C$, let $M_k = C \setminus Y_k$, $NS = |S_C|$, $1 \le k \le NS$, and $M = \{M_1, \ldots, M_{NS}\}$. We have $G \in [C] \Leftrightarrow \forall M_k, \exists x \in G: x \in M_k$.*

Proof. We have $G \in [C] \Leftrightarrow \forall Y \in S_C, \exists x \in G: x \notin Y \Leftrightarrow \forall M_k, \exists x \in G: x \in M_k$. Therefore, this corollary is proven.

Definition 2 (the kernel and extendable set). *Given two itemsets G and E those are disjoint. Let the notation [G, E] denote the class of all itemsets those are supersets of G and subsets of G + E, i.e., $[G, E] = \{X: G \subseteq X \subseteq G + E\}$.*

G is called the kernel and E is called the extendable set of the class.

The following results provide an easy way to find the pairs of kernel and extendable set that can help partition [*C*] into equivalence classes.

Definition 3. *Let $M_k = C \setminus Y_k$, $\forall Y_k \in S_C$, $M = \{M_1, \ldots, M_{NS}\}$. For $1 \le k \le |S_C|$, let $S_k = \{M_1, \ldots, M_k\}$ denotes the set of k first elements of M, and $S_0 = \emptyset$.*

An itemset G is said to "satisfy S_k" if $\forall M_i \in S_k, \exists x \in G: x \in M_i$. Let $[S_k]$ denote the set of all FIs that satisfies S_k, and $[S_k] = 2^C$ if $k = 0$.

The following lemmas are obtained by Definition 3 and Corollary 1.

Lemma 1. *$\forall G \subseteq C$, $E \subseteq C$, $G \cap E = \emptyset$, $1 \le k \le |S_C|$, we have:*

a) $[S_k] \subseteq [S_{k-1}]$, *(i.e., if G satisfies S_k, it also satisfies S_{k-1}).*
b) $\forall G \in [S_k] \Rightarrow \forall X \in [G, E], X \in [S_k]$.
c) $[C] = [S_{NS}]$.

Lemma 2. *For $1 \le k \le |S_C|$, $G^* \in [S_{k-1}]$, and let $G = G^* + \{x\}$, $x \in C$, we have: $(M_k \cap G^* \ne \emptyset) \vee (x \in M_k) \Rightarrow G \in [S_k]$*

In the first condition case (i.e., $M_k \cap G^* \ne \emptyset$), *x* is not necessary for *G* to satisfy S_k since G^* already satisfy S_k, meanwhile, in latter it is.

From now on, we denote "+" (and "$\sum$") the union operator for two (and many, respectively) disjoint sets. Definition 4 below lead to the idea of generating the kernel sets.

Definition 4 (k-minimal set). *Given $G = G^* \cup X$, and $X \subseteq M_k$. G is "k-minimal" if one of the following conditions is satified:*

a) *G^* is "(k-1)-minimal", $M_k \cap G^* \ne \emptyset$, $X = \emptyset$ (i.e., $G = G^*$, and no more item are needed for G^* to satisfy S_k).*

b) *G^* is "(k-1)-minimal" $M_k \cap G^* = \emptyset$, and $X = \{x\}$, $x \in M_k\}$ (i.e., x is the new item needed for G^* to satisfy S_k).*

c) *$\emptyset$ is "0-minimal".*

We can see that if *G* is "*k*-minimal" then *G* satisfies S_k. Cases (*a*), and (*b*) are based on Lemma 2.
It is worthy to note that *G* is "*k*-minimal" does not imply that there is no subset of *G* satisfying S_k. It just implies that no prefix of *G* satisfies S_k if *G* is treated as a sequence of items.

From now on, we assume that there exists an order over the items in C (e.g., alphabetic order), and every M_i is sorted in the increasing order.

Definition 5 (the partition of $[S_k]$ by kernel and extendable set). *Given C, and S_C. Let the set Q_k contain the pairs of (kernel set G, extendable set E) defined recurrently as follows:*

a) $Q_0 = \{(\varnothing, C)\}$

b) $Q_1 = \{(G_i, E_i): G_i = \{x_i\}, x_i \in M_1, E_i = C \setminus \{y \in M_1: y \leq x\}\}$

c) $\forall k > 1, Q_k = B_k + C_k$, *where:*

$B_k = \{(G, E): (G, E) \in Q_{k-1}, G \cap M_k \neq \varnothing\}$,

$C_k = \{(G + \{x_i\}, E \setminus E_i): (G, E) \in Q_{k-1}, G \cap M_k = \varnothing, N_k = M_k \cap E, x_i \in N_k, E_i = \{y \in N_k: y \leq x_i\}$.

In details, B_k contains pairs (G, E) in Q_{k-1} where G is "(k-1)-minimal" and is also "k-minimal", according to Definition 4.*a*. Thus, (G, E) belongs to Q_k also. Meanwhile, C_k contains the pairs (G, E) such that $G = G' + \{x_i\}$, where G' is "(k-1)-minimal" but not "k-minimal", and x_i is necessary for G to become "k-minimal" (as Definition 4.b).

Lemma 3. *For all $(G_i, E_i), (G_j, E_j) \in Q_k$, $i \neq j$, $1 \leq k \leq |S_C|$, we have:*

a) $X \in [G_i, E_i] \Rightarrow X \in [S_k]$.

b) $G_i \cap E_i = \varnothing$.

c) $[G_i, E_i] \cap [G_j, E_j] = \varnothing$.

In other words, Q_k induces a partition of all FIs in $[S_k]$, where each equivalence classes is defined by $[Q, E]$ as in Definition 2, with $(G, E) \in Q_k$. Furthermore, for every $(G, E) \in Q_k$, G is *"k-minimal"*.

Theorem 1. *For $0 \leq k \leq |S_C|$, $\{[G, E]: (G, E) \in Q_k\}$ is a partition of $[S_k]$, where each $[G, E]$ is an equivalence class, and G is "k-minimal".*

Proof.

A. We'll first prove that Theorem 1 hold with $k = 0$.

Since k = 0, $Q_0 = \{(\varnothing, C)\}$ by Definition 5.a, and $[S_k] = 2^C$ by Definition 3. We have: $[\varnothing, C] = \{X: X \subseteq C\} = 2^C$. Since $\{[\varnothing, C]\}$ is a partition of 2^C and $\varnothing$ is "*0-minimal*", let G = $\varnothing$ and E = C then Theorem 1 is proven for k = 0.

B. Assume that Theorem 1 holds for any k-1, $0 < k \leq |S_C|$ (i.e., the set $\{[G, E]: (G, E) \in Q_{k-1}\}$ is a partition of $[S_{k-1}]$, where each $[G, E]$ is an equivalence class, and G is *"(k-1)-minimal"*), we will prove that Theorem 1 holds for k.

By assumption, we have $[S_{k-1}] = \sum [G_i, E_i]$, where $(G_i, E_i) \in Q_{k-1}$. By Lemma 1.a, for an itemset X to be in $[S_k]$, it must be in $[S_{k-1}]$ $^{(a)}$. By Lemma 3.c, for all (G_i, E_i) and $(G_j, E_j) \in Q_{k-1}$, $i \neq j$, we have: $[G_i, E_i] \cap [G_j, E_j] = \varnothing^{(b)}$. From $^{(a)}$ and $^{(b)}$, we only need to prove that given a pair $(G, E) \in Q_{k-1}$, we can partition $[G, E]$ into disjoint subclasses, where each subclass either is in the form of $[G', E']$ and G' is a *"k-minimal"* (i.e., $(G', E') \in Q_k$), or contains only the itemsets which do not satisfy S_k $^{(*)}$.

If $M_k \cap G \neq \varnothing$, then G satisfies S_k. Then, $(G, E) \in Q_k$. Then $^{(*)}$ is proved. In this case, (G, E) belongs to B_k as in Definition 5.b, thus, it belongs to Q_k $^{(i)}$. Now, let assume that $M_k \cap G = \varnothing$. If $M_k \cap E = \varnothing$, then for all $Y \in [G, E]$, Y does not satisfy S_k. Then $^{(*)}$ is proved $^{(ii)}$.

Now, let assume that $M_k \cap G = \emptyset$ and $M_k \cap E \neq \emptyset$. Denote $N_k = M_k \cap E = \{x_1, .., x_n\}$, and for $0 < i \leq n$, denote $G_i = G + \{x_i\}$ $(x_i \in N_k)$, $E_i = E \backslash \{x_j: x_j \in N_k, j \leq i\}$. Let $U_1 = \{G_1 + T: T \subseteq E_1\} = [G_1, E_1]$, and $V_1 = \{G + Y: Y \subseteq E \setminus \{x_1\}\} = [G, E_1]$. Then, $\forall X \in U_1$ $(x_1 \in X)$ and $\forall Y \in V_1$ $(x_1 \notin Y)$. This means: U_1 and V_1 are disjoint. We can further divide V_1 into two disjoint sets: U_2 – the set of itemsets containing x_2, and V_2 - the set of itemsets not containing x_2. One can see that $E_i = E_{i-1} \setminus \{x_i\}$, then, we have: $U_2 = \{G_2 + T: T \subseteq E_2\} = [G_2, E_1 \setminus \{x_2\}] = [G_2, E_2]$, $V_2 = [G, E_2]$, and U_2 and $V_2 = \emptyset$. By the same way, the division process continues until we divide V_{n-1} into two disjoint sets: $U_n = [G_n, E_n]$ and $V_n = [G, E_n]$. One can see that, $[G, E] = V_n + \sum [G_i, E_i]$.

Since $M_k \cap G = \emptyset$ and $E_n = \emptyset$, for all $Y \in V_n$, Y does not satisfy S_k. Meanwhile, since $N_k \cap G_i = \{x_i\} \neq \emptyset$, for all $i \leq n$. This means $M_k \cap G_i \neq \emptyset$. Thus, G_i is "k-minimal" by Definition 4.b and for all $X \in [G_i, E_i]$, X satisfies S_k. This mean (G_i, E_i) is generated as Definition 5.c then $(G_i, E_i) \in Q_k$. Then (*) is proved. $^{(iii)}$

By $^{(i)}$, $^{(ii)}$, and $^{(iii)}$, $^{(*)}$ is proved, and Theorem 1 is proved.

Corollary 2: $\{[G, E]$*: (G, E) ∈ $Q_{NS}\}$ is a partition of* $[C]$, *where* each *pair* $[G, E]$ *is an equivalence class, and G is "NS-minimal".*

Proof: This is result of Theorem 1, where $k = NS$, $[C] = [S_k]$.

Example 2: Let consider the relation R shown in Table 1 and the lattice of FCIs mined from R with *minsup* = 0.25 (i.e., absolute minimum support = 1) shown in Figure 1. The following paragraphs explain how to find all FIs in $[C]$ for $C = \{1, 2, 3, 4, 5, 6\}$.

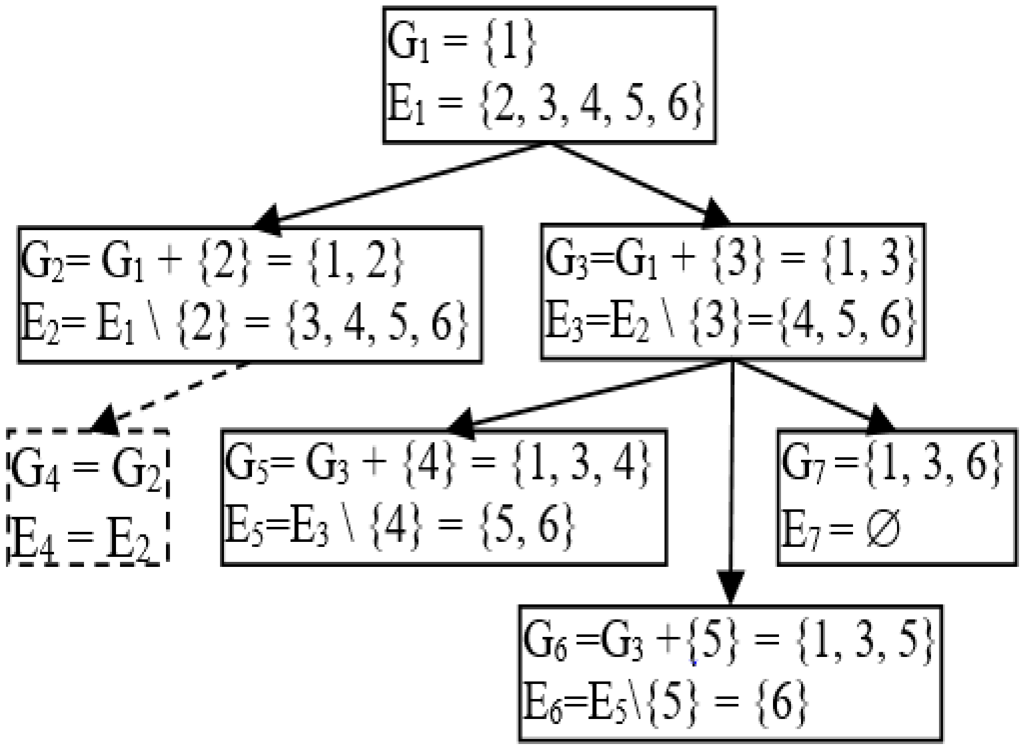

Figure 2. Search tree for generating the kernels and extendable sets of Q_3, where $Q_3 = \{(G_2, E_2), (G_5, E_5), (G_6, E_6), \{(G_6, E_7)\}$.

Table 2. The partition of 23 FIs in [{1, 2, 3, 4, 5, 6}] based on Q_3 as in Figure 2.

	$[G_2, E_2]$	$[G_5, E_5]$	$[G_6, E_6]$	$[G_7, E_7]$
1	{{1, 2},	{{1, 3, 4},	{{1, 3, 5},	{{1, 3, 6}}
2	{1, 2, 3},	{1, 3, 4, 5},	{1, 3, 5, 6}}	
3	{1, 2, 3, 4},	{1, 3, 4, 5, 6},		
4	{1, 2, 3, 4, 5},	{1, 3, 4, 6}}		
5	{1, 2, 3, 4, 5, 6},			
6	{1, 2, 3, 4, 6},			
7	{1, 2, 3, 5},			
8	{1, 2, 3, 5, 6},			
9	{1, 2, 3, 6},			
10	{1, 2, 4},			
11	{1, 2, 4, 5},			
12	{1, 2, 4, 5, 6},			
13	{1, 2, 4, 6},			
14	{1, 2, 5},			
15	{1, 2, 5, 6},			
16	{1, 2, 6}}			

According to the lattice, the immediately closed subsets of C are $Y_1 = \{2, 3, 4, 5, 6\}$, $Y_2 = \{1, 4, 5, 6\}$ and $Y_3 = \{1, 3\}$. In other words, $S_C = \{Y_1, Y_2, Y_3\}$. Figure 2 presents the search tree that can be built, implicitly, during the process of generating the kernels and extendable sets using a breadth-first search. Here, $Q_1 = \{(G_1, E_1)\}$, $Q_2 = \{(G_2, E_2), (G_3, E_3)\}$, and $Q_3 = \{(G_2, E_2), (G_5, E_5), (G_6, E_6), \{(G_6, E_7)\}$, where G_i is a kernel and E_i is its corresponding extendable set. (We do not have to compute the pair (G_4, E_4), the rectangle with dash-lined border.) They are found by the following steps:
With $M_1 = C \setminus Y_1 = \{1\}$, by Definition 5.b, let $G_1 = \{1\}$, $E_1 = C \setminus \{1\} = \{2, 3, 4, 5, 6\}$) we have: $Q_1 = \{(G_1, E_1)\}$. (Then, $[S_1] = [G_1, E_1]$, but we do not need to compute it!)

Now, we will find Q_2 based on (G_1, E_1) and M_2, where $M_2 = C \setminus Y_2 = \{2, 3\}$). Let $N_2 = M_2 \cap E = \{2, 3\}$. Using item 2 in N_2 we have $G_2 = G_1 + \{2\} = \{1, 2\}$, and $E_2 = E_1 \setminus \{2\} = \{3, 4, 5, 6\}$. Using item 3 in N_2 we have $G_3 = G_1 + \{3\} = \{1, 3\}$, and $E_3 = E_2 \setminus \{3\} = \{4, 5, 6\}$. Then, $Q_2 = \{(G_2, E_2), (G_3, E_3)\}$ as nodes of level 2 in Figure 2.

Now, we will find Q_3 based on $\{(G_2, E_2), (G_3, E_3)\}$ and M_3, where $M_3 = C \setminus Y_3 = \{2, 4, 5, 6\}$. With (G_2, E_2), one can see that G_2 satisfies S_3 since $M_3 \cap E_2 = \{2\} \neq \emptyset$. Thus, (G_2, E_2) belongs to Q_3. With (G_3, E_3), we have $N_3 = M_3 \cap E_3 = \{4, 5, 6\}$. With item 4 in N_3, we have $G_5 = G_3 + \{4\} = \{1, 3, 4\}$, and $E_5 = E_3 \setminus \{4\} = \{5, 6\}$. With item 5 in N_3, we have $G_6 = G_3 + \{5\} = \{1, 3, 5\}$, and $E_6 = E_5 \setminus \{5\} = \{6\}$. With item 6 in N_3, we have $G_7 = G_3 + \{6\} = \{1, 3, 6\}$, and $E_7 = E_6 \setminus \{6\} = \emptyset$. Then, $Q_3 = \{(G_2, E_2), (G_5, E_5), (G_6, E_6), \{(G_6, E_7)\}$ as nodes of level 3 in Figure 2.

5. Nuclear Algorithm

In this section, we present a breadth-first search algorithm for generating FIs from a lattice of FCIs: based on Definition 4 and Theorem 1 of the previous section.

Input: L = the lattice of FCIs mined from R for a given *minsup*

Output: all FIs satisfying *minsup*.

```
1   $\forall C \in L$,
2   $G = \varnothing$;  $E = C$;
3   if ($S_C = \varnothing$)        return $2^C \backslash \varnothing$;
4   else
5   $\forall Y_i \in S_C$, $M_i = C \backslash Y_i$.
6   TRY_BFS(1, G, E);
```

Figure 3. The NUCLEAR algorithm.

For each immediate frequent closed subset C in lattice L, the variables G and E are initialized as $G = \varnothing$ and $E = C$. If $S_C = \varnothing$, the class [C] is all non-empty subsets of C. Otherwise, the recursive TRY_BFS algorithm is called (Figure 4).

Input: k: $k \le NS$; G: itemset (k-1)-minimal; E: extendable set of G.

Output: all FIs satisfying S_k.

```
1   if ($k > |Sc|$)
2   $\forall$(G, E) $\in Q_{k-1}$,
3   Save [G, E];
4   Stop;
5   $Q_k = \varnothing$;
6   $\forall$(G, E) $\in Q_{k-1}$
7   if($M_k \cap G \neq \varnothing$) $Q_k = Q_k + \{(G, E)\}$;
8   else
9   $E_i = E$;
10  $N_k = M_k \cap E$ ;
11  $\forall x \in N_k$
12  $E_i = E_i \backslash \{x\}$;
13  $Q_k = Q_k + \{(G + \{x\}, Ei)\}$;
14  TRY_BFS(k+1, $Q_k$);
```

Figure 4. The TRY_BFS algorithm.

6. Experimental Results and Discussion

In this section, we compare the running time (in seconds) for mining all FIs from data of three frameworks: dEclat, GenIT, and NUC. dEclat mines FIs directly from data. GenIT is not an algorithm but a combination of algorithms of some previous studies, which is slightly different to our proposed approach. In GenIT, we first use CharmL algorithm to mine the lattice of FCIs from data, then, use Minimal Generator algorithm to mine the generators from the FCIs before applying GEN_ITEMSETS to extract FIs from the FCIs and the generators by FIs. In NUC, we also use CharmL algorithm to mine the lattice of FCIs from data before applying NUCLEAR to generate the kernels and extendable sets. The FIs are inferred during this process. Thus, to be fair, the total times reported for NUC includes the time of CharmL and the time of NUCLEAR;

whereas, that for GenIT includes the time of CharmL, the time of Minimal Generator, and the time of GEN_ITEMSETS. These algorithms have been executed on a Pentium (R) Dual-Core CPU E6500 @ 2.93GHz, equipped with 1.94GB of RAM, running the Microsoft Windows XP Version 2002 operating system.

Seven datasets (available at [32, 33] have been used to compare the frameworks under different *minsup* threshold values. Information about the datasets is given in Table 3. The number of patterns can be mined from each dataset are shown in Table 4. Table 5 shows the overall runtime for NUC, GenIT, and dEclat in the columns of the corresponding names, and other details. The visual comparisons of the three approaches are also given in Figures 5-11.

In our experiment, NUC is faster than dEclat when testing on the Mushroom and Connect datasets (Figure 8, 9). The reasons are: (1) the time for CharmL (column tCS) is smaller than that of dEclat because the number of FCIs in a dataset is much less than that of FIs, and (2) the time for NUCLEAR (column tNNI) is also smaller than that of dEclat. On the other datasets, NUC is slower than dEclat. However, the main reason is just CharmL is slower dEclat, whereas, the time for NUCLEAR is significantly small as compared to those of CharmL and dEclat.

From Table 5, we can estimate that NUC is about 1.25 time faster than GenIT. To clearer see the advantages of NUCLEAR, we break down the runtime of GenIT and NUC into the times for different stages including: (1) constructing the lattice of FCIs by CharmL (column tCS), (2) mining generators by Minimal Generator (column tG), (3) extracting FIs from the FCIs and the generators by GEN_ITEMSETS (column tGI), (4) generating kernels and extendable sets from the lattice of FCIs by NUCLEAR (column tN), and (5) enumerating the FIs from the kernels and extendable sets by NUCLEAR (column tNI). In comparing tG against tN, and tGI against tNI, one can see that the runtime of Minimal Generator to mine the generators is about double that of NUCLEAR for generating the kernels and extendable sets (tN), and the time for GEN_ITEMSETS (tGI) is similarly about twice as that of NUCLEAR for extracting FIs. These make NUCLEAR more efficient than GenIT in extracting the intermediate and the final results.
The time for minings FIs from the lattice using NUCLEAR (tNNI) is mostly much smaller than that for constructing the lattice of FCIs (tCS) using the CharmL algorithm. Thus, in the applications where users have to try different *minsup* thresholds to find an optimal set of FIs for a certain downstream process, our approach might be more efficient than repeatedly mining FIs from scratch like dEclat. Because, the lattice can be constructed only once for a small enough *minsup*, the cost for updating/filtering the lattice is expected to be small, and after that NUCLEAR can be used to query FIs many times. For example, in bioinformatics, we can use NUC to conduct a feature selection which shrinks the high dimensional data to a smaller one before applying a machine learning algorithm. The feature selection might have to be conducted many times to obtain an optimal set of features.

From Table 4, we can see that the number of pairs of kernels and extendable sets (#N) is almost similar to the number of generators (#G) and just slightly bigger than the number of FCIs (#C). Furthermore, we do not need to store the kernels and extendable sets but just the FCIs. Because, for each FCI, we only need the largest FCIs those are its subsets to generate the kernels, extendable sets, and FIs, and this can be done quickly. Thus, using NUCLEAR can save a lot of memory as well.

Table 3. Characteristics of the datasets.

Database	**Abbreviation**	**#Items**	**#Transactions**	**#Average**
Chess	CH	75	3,196	37
Connect	CO	129	67,557	43
Mushroom	MU	119	8,124	23
Retail	RE	16,469	88,162	10.3
T40I10100K	T4	1,000	100,000	40
C20d10k	C2	192	10,000	20
C73d10k	C7	1,592	10,000	73

Table 4. Number of patterns extracted from datasets.

Data	***minsup***	**#FS**	**#CS**	**#G**	**#N**	**#FS/#CS**	**#CS/#N**	**#N/#G**
C2	0.8	8165081	99785	122031	122359	81.83	0.82	1.00
C2	0.85	7525408	95533	116126	116416	78.77	0.82	1.00
C2	0.9	7017040	92087	111297	111564	76.2	0.83	1.00
C2	0.95	6525355	88695	106575	106813	73.57	0.83	1.00
C2	1	6092449	85608	102316	102519	71.17	0.84	1.00
C7	50	25696439	482902	765450	765449	53.21	0.63	1.00
C7	55	9698268	222253	346029	346028	43.64	0.64	1.00
C7	60	4188627	108428	166918	166917	38.63	0.65	1.00
C7	65	1472818	47491	71875	71874	31.01	0.66	1.00
C7	70	543081	19501	29008	29007	27.85	0.67	1.00
CH	50	900355	369450	372603	372603	2.44	0.99	1.00
CH	60	156551	98392	98418	98418	1.59	1.00	1.00
CH	70	24997	23991	23991	23991	1.04	1.00	1.00
CO	60	21184454	68349	68349	68349	309.95	1.00	1.00
CO	70	4093971	35875	35875	35875	114.12	1.00	1.00
CO	75	1561212	24346	24346	24346	64.13	1.00	1.00
CO	80	518875	15107	15107	15107	34.35	1.00	1.00
MU	2	23596649	31767	57728	82483	742.8	0.55	1.43
MU	3	9934877	22229	37972	52165	446.93	0.59	1.37
MU	4	4324745	16565	26984	35597	261.08	0.61	1.32
MU	5	3727905	12854	21160	27801	290.02	0.61	1.31
RE	0.006	975063	504142	532342	542565	1.93	0.95	1.02
RE	0.008	480620	286435	293235	294709	1.68	0.98	1.01
RE	0.01	240852	189077	191265	191650	1.27	0.99	1.00
RE	0.02	67186	65301	65329	65330	1.03	1.00	1.00
RE	0.03	40153	39552	39552	39552	1.02	1.00	1.00
RE	0.04	26925	26666	26666	26666	1.01	1.00	1.00
RE	0.05	19836	19698	19698	19698	1.01	1.00	1.00
T4	0.8	480531	480531	480531	480531	1	1.00	1.00
T4	0.85	432211	432211	432211	432211	1	1.00	1.00
T4	0.9	350323	350323	350323	350323	1	1.00	1.00
T4	0.95	210610	210610	210610	210610	1	1.00	1.00
T4	1	66278	66278	66278	66278	1	1.00	1.00
Note: *#FS: the number of FIs; #CS: the number of FCIs; #G: the number of generators; #N: the number of pairs of kernels and extendable sets.*								

Table 5. Runtimes of the frameworks and of the breakdown process.

Data	***minsup***	**tCS**	**tG**	**tN**	**tNI**	**tNNI**	**tGI**	**GenIT**	**dEclat**	**NUC**
C2	0.800	14.8	*3.6*	*2.4*	6.1	8.5	14.1	32.5	24.6	23.3
C2	0.850	13.9	*3.4*	*2.3*	5.3	7.7	12.8	30.1	22.6	21.6
C2	0.900	13.1	*3.3*	*2.2*	5.0	7.2	11.9	28.3	21.2	20.3
C2	0.950	12.5	*3.2*	*2.3*	4.6	6.9	11.2	26.9	19.6	19.5
C2	1.000	12.0	*3.1*	*1.9*	3.9	5.8	10.3	25.4	18.4	17.8
C7	50.000	192.5	*33.3*	*23.6*	16.3	40.0	outM	outM	88.1	232.4
C7	55.000	58.9	*14.0*	*9.2*	5.4	14.6	19.4	92.2	33.2	73.5
C7	60.000	20.0	*6.3*	*3.9*	2.0	5.9	7.8	34.1	14.6	25.9
C7	65.000	6.5	*2.4*	*1.5*	0.7	2.2	2.7	11.6	5.3	8.7
C7	70.000	2.3	*0.9*	*0.4*	0.4	0.8	0.9	4.1	2.1	3.1
Ch	50.000	109.3	*17.4*	*5.5*	0.6	6.1	3.0	129.6	5.0	115.4
Ch	60.000	14.3	*3.9*	*1.3*	0.1	1.5	0.6	18.7	1.1	15.8
Ch	70.000	1.8	*0.8*	*0.3*	0.0	0.3	0.1	2.7	0.3	2.1
Co	60.000	35.2	*2.4*	*1.8*	16.5	18.3	outM	outM	95.0	53.4
Co	70.000	13.3	*1.2*	*0.9*	2.9	3.8	7.3	21.8	20.5	17.1
Co	75.000	7.9	*0.8*	*0.5*	0.9	1.4	2.6	11.3	8.7	9.4
Co	80.000	4.7	*0.6*	*0.3*	0.3	0.6	0.8	6.1	3.5	5.3
M	2.000	3.8	*2.3*	*1.1*	18.3	19.3	outM	outM	71.2	23.1
M	3.000	2.7	*1.3*	*0.4*	8.0	8.4	28.2	32.1	30.1	11.1
M	4.000	1.9	*0.8*	*0.5*	3.2	3.7	13.8	16.5	13.3	5.6
M	5.000	1.6	*0.9*	*0.3*	2.9	3.2	11.1	13.7	11.3	4.8
RT	0.006	101.2	*9.8*	*4.9*	0.8	5.7	7.8	118.8	60.6	106.9
RT	0.008	57.2	*5.2*	*2.4*	0.3	2.7	1.2	63.6	27.2	59.8
RT	0.010	39.0	*3.1*	*1.5*	0.1	1.6	0.5	42.7	16.5	40.6
RT	0.020	17.1	*1.1*	*0.4*	0.0	0.5	0.1	18.3	5.7	17.6
RT	0.030	12.0	*0.6*	*0.2*	0.0	0.3	0.1	12.8	3.8	12.3
RT	0.040	9.0	*0.4*	*0.2*	0.0	0.2	0.1	9.4	2.8	9.2
RT	0.050	7.2	*0.4*	*0.1*	0.0	0.2	0.0	7.6	2.2	7.4
T4	0.800	427.7	*28.9*	*7.4*	0.3	7.7	1.4	458.0	41.2	435.4
T4	0.850	372.0	*24.6*	*5.9*	0.2	6.1	1.3	397.9	35.2	378.2
T4	0.900	289.2	*19.3*	*5.3*	0.2	5.5	1.3	309.8	29.9	294.7
T4	0.950	120.8	*9.7*	*2.8*	0.1	2.8	0.6	131.1	25.2	123.6

Data	***minsup***	**tCS**	**tG**	**tN**	**tNI**	**tNNI**	**tGI**	**GenIT**	**dEclat**	**NUC**
T4	1.000	75.4	*1.9*	*0.6*	0.0	0.6	0.1	77.4	21.1	76.0

Note: *tCS: time to find FCIs using CharmL; tG: time to find the generators using Minimal Generator; tN: time to generate kernels and extendable sets by NUCLEAR; tNI: time to enumerate FIs from kernels and extendable sets by NUCLEAR; tNNI: time to find FIs from lattice of FCIs, which is the sum of tN and tNI; tGI: time to generate FIs based on FCIs and generators using GEN_ITEMSETS; NUC: time to find FIs from data using CharmL and NUCLEAR; GenIT: time to find FIs using CharmL, Minimal Generator, and GEN_ITEMSETS; dEclat: time to find FIs from data using dElat. outM: out of memory.*

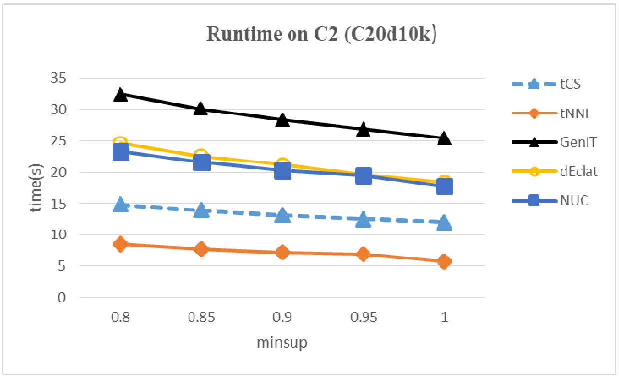

Figure 5. Time execution comparison on C20d10k.

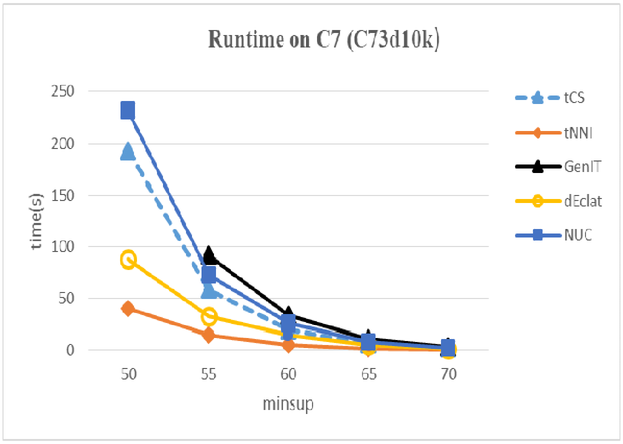

Figure 6. Time execution comparison on C73d10k.

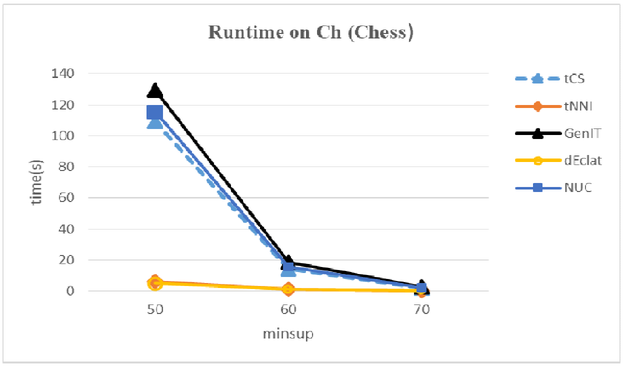

Figure 7. Comparison of time on Chess.

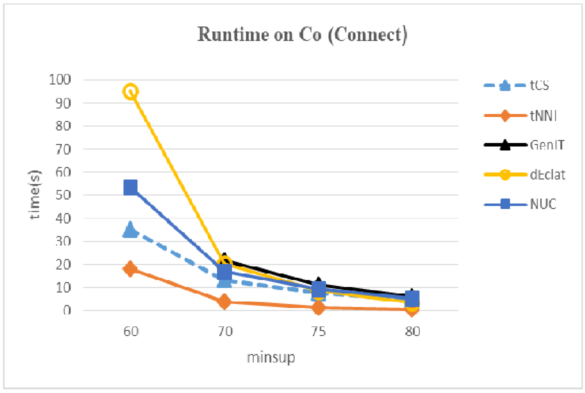

Figure 8. Comparison of time on Connect.

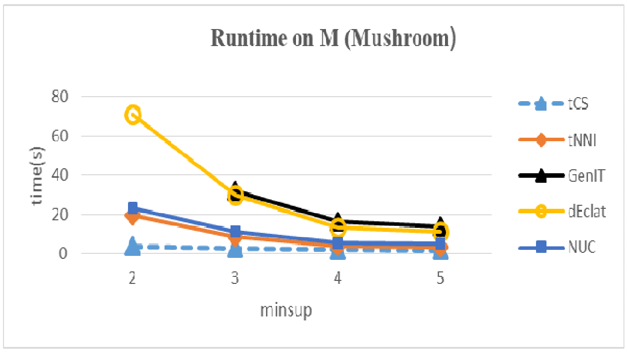

Figure 9. Comparison of time on Mushroom.

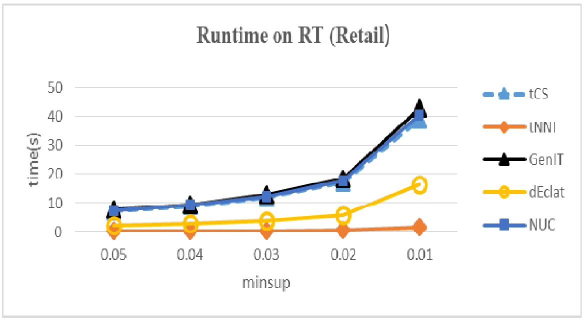

Figure 10. Time execution comparison on Retail.

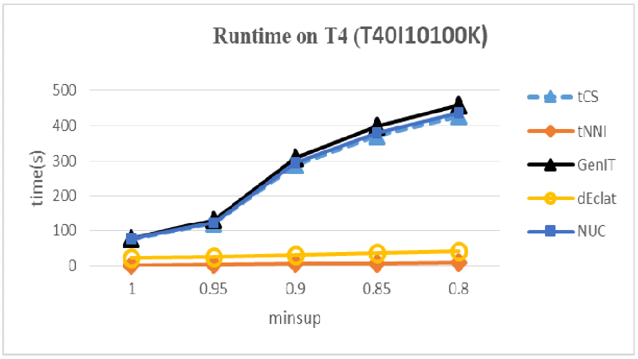

Figure 11. Time execution comparison on T40I10100K.

7. Conclusions and Future Work

Mining FIs from the lattice of FCIs is a reasonable approach since the number of FCIs is often much smaller than the number of FIs. Thus, FCIs can be mined with a limited amount of memory. Especially, because there are parallel and distributed algorithms to mine FCIs for large or high dimensional data, the FCIs are easier to be available than the FIs. Besides, FCIs can be used to partition FIs into equivalence classes that can be used to efficiently process FIs in parallel. This approach is interesting as the lattice of FCIs can be mined once for a minimum support threshold that is small enough and used many times later to derive FIs for different minimum support thresholds.

In this paper, we presented a recurrent formula for generating the kernels and extendable sets from a lattice of FCIs without the need of the generators. They are simple enough so that users can easily and quickly derive the FIs from them, and we even don't need to store them. Thank for that, NUC, the approach using NUCLEAR to mine the FIs from the lattice of FCIs is more efficient than GEN_IT, a similar approach that requires the generators for mining FIs from the lattice of FCIs. NUC is slower than dEclat in the major cases, but it's just mainly because the construction of the lattice by CharmL takes more time than dEclat; whereas, the time for obtaining the FIs from the lattice by NUCLEAR is still considerably small.

In the future, the methods for updating the FIs when *minsup* is changed will be studied for the case that lattice can be constructed only once and reuse many times. We would like to test our approach on the real data, such as bioinformatics data, where FIs cannot be mined directly from data within a reasonable amount of time while our approach with or without parallel implementation can.

Acknowledgements

This work has been partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

The authors would like to thank Tran and the co-authors of [20] for providing us the code of GEN_ITEMSETS algorithm for our experiments.

References

[1] Agrawal R., Imielinski T., Swami N. (1993). Mining association rules between sets of items in large databases. In: ACM SIGMOID, pp. 207-216.

[2] Mai T., Vo B., Nguyen L.T.T. (2017). A lattice-based approach for mining high utility association rules. Information Sciences, 399, 81-97.

[3] Yun U., Lee G., Yoon E. (2017). Efficient high utility pattern mining for establishing manufacturing plans with sliding window control. IEEE Transactions on Industrial Electronics, 64(9), 7239 – 7249.

[4] Mai T., Nguyen L.T.T. (2017). An Efficient Approach for Mining Closed High Utility Itemsets and Generators. Journal of Information and Telecommunication, 1(3), 193-207.

[5] Bundit, M., Nunnapus, B., Arnon, R., Athasit, S., Putchong, U. (2007). Parallel association rule mining based on FI-Growth algorithm. In: ICPDS'07, pp. 1-8.

[6] Lakhal, L., and Stumme, G. (2005). Efficient mining of association rules based on formal concept analysis. In: FCA'05, pp. 180-195.

[7] Grahne G., Zhu J. (2005). Fast algorithms for frequent itemset mining using FP-Trees. IEEE Transactions on Knowledge and Data Engineering, 17(10), 1347-1362.

[8] Zaki, M.J. and Hsiao, C.J. (2005). Efficient algorithms for mining closed itemsets and their lattice structure. IEEE Transactions on Knowledge and Data Engineering, 17(4), 462-478.

[9] Zaki, M.J. (2004). Mining non-redundant association rules. Data Mining and Knowledge Discovery, 9(3), 223-248.

[10] Sahoo, J., Das, A. K., Goswami, A. (2015). An effective association rule mining scheme using a new generic basis. Knowledge and Information Systems, 43(1), 127–156.

[11] Negrevergne, B., Termier, A., Méhaut, J., Uno, T. (2010). Discovering Closed Frequent Itemsets on Multicore: Parallelizing Computations and Optimizing Memory Accesses. 2010 International Conference on High Performance Computing and Simulation (HPCS), pp. 521-528.

[12] Le T., Vo B. (2016). The Lattice-based approaches for mining association rules: a review. WIREs Data Mining and Knowledge Discovery, 6(4), 140-151.

[13] Vo B., Le B. (2009). Mining traditional association rules using frequent itemsets lattice. In: CIE’09, pp. 1401–1406.

[14] Deng Z.H. (2016). DiffNodesets: An efficient structure for fast mining frequent itemsets. Applied Soft Computing, 41, 214-223.

[15] Deng Z.H., Lv S.L. (2015). PrePost+: An efficient N-lists-based algorithm for mining frequent itemsets via Children-Parent Equivalence pruning. Expert Systems with Applications, 42(13), 5424-5432.

[16] Vo B., Le T., Coenen F., Hong T.P. (2016). Mining frequent itemsets using the N-list and subsume concepts. International Journal of Machine Learning and Cybernetics, 7(2), 253-265.

[17] Goethals, B., and Zaki, M. (2003). FIMI '03 Workshop on Frequent Itemset Mining Implementations. http://www.cs.rpi.edu/~zaki/PaperDir/FIMI03.pdf.

[18] Tran N.A., Duong V.H., Tran C.T., Le H.B (2011). Efficient algorithms for mining frequent itemsets with constraint. In: KSE’11, pp. 19-25.

[19] Tran N.A., Tran C.T., Le H.B. (2012). Structures of association rule set. In: ACIIDS’12, pp. 361-370

[20] Truong C.T., Tran N.A. (2010). Structure of set of association rules based on concept lattice. In: ACIIDS’10, pp. 217-227.

[21] Szathmary, L., Valtchev, P., Napoli, A., Godin, R. (2009). Efficient vertical mining of frequent closures and generators. In Advances in Intelligent Data Analysis VIII (pp. 393-404). Springer Berlin Heidelberg.

[22] Anh N. T., Tin C.T., and Bac L.H. (2013). An approach for mining concurrently closed itemsets and generators. In: ICCSAMA’13, pp.355–366.

[23] Vo B., Hong T.P., Le B. (2012). DBV-Miner: A dynamic Bit-Vector approach for fast mining frequent closed itemsets. Expert Systems with ApplIcations, 39(8), 7196-7206.

[24] Vo B., Le B. (2011). Interestingness measures for association rules: Combination between lattice and hash tables. Expert Systems with Applications, 38(9), 11630-11640.

[25] Vo B., Le T., Hong T.P., Le, B. (2014). An effective approach for maintenance of pre-large-based frequent-itemset lattice in incremental mining. Applied Intelligence, 41(3), 759-775.

[26] Agrawal R., Shafer J.C. (1996). Parallel mining of association rules. IEEE Transactions on Knowledge and Data Engineering, 8(6), 962-969.

[27] Han E., Karypis G., and Kumar V. (1997). Scalable parallel data mining for association rules. In: ACM SIGMOD'97, pp. 277-288.

[28] Zaïane, O.R., El-Hajj, M., and Lu, P. (2001). Fast parallel association rule mining without candidacy generation. In: ICDM'01, pp. 665-668.

[29] Pasquier N., Taouil R., Bastide Y., Stumme G., and Lakhal L. (2005). "Generating a condensed representation for association rules," J. of Intelligent Information Systems, vol. 24, no. 1, pp. 29-60.

[30] Ai, D., Pan, H., Li, X., Gao, Y., & He, D. (2018). Association rule mining algorithms on high-dimensional datasets. Artificial Life and Robotics, 23(3), 420-427.

[31] Fournier-Viger P., Lin J.C.W., Vo B., Truong T.C., Zhang J., Le H.B. (2017). A survey of itemset mining. WIREs Data Mining and Knowledge Discovery, 7(4), e1207

[32] http://fimi.ua.ac.be/data.

[33] http://coron.loria.fr/site/downloads_datasets.php.