Boosting Frequent Itemset Mining via Early Stopping Intersections

Huu Hiep Nguyen

{nguyenhuuhiep2@dtu.edu.vn}
Institute of Research and Development, Duy Tan University
P809 7/25 Quang Trung, Danang 550000, Vietnam

Abstract—Mining frequent items from a transaction database has emerged as a fundamental problem in data mining and committed itself as a building block for many pattern mining tasks. In this paper, we present a general technique to reduce support checking time in existing depth-first search generate-and-test schemes such as Eclat/dEclat and PrePost+. Our technique allows infrequent candidate itemsets to be detected early. The technique is based on an early-stopping criterion and is general enough to be applicable in many frequent itemset mining algorithms. We have applied the technique to two TID-list based schemes (Eclat/dEclat) and one N-list based scheme (PrePost+). Our technique has been tested over a variety of datasets and confirmed its effectiveness in runtime reduction.

I. INTRODUCTION

First proposed by Agrawal et al. [2], frequent itemset mining has become a popular data mining technique and has been studied extensively by many researchers. It has played an essential role in many important data mining tasks such as mining association rules [14], sequential patterns [6], [13], correlations [9], episodes [16], classification [10], clustering [11] and so on. Although lots of algorithms have been proposed, how to improve the efficiency of itemset mining algorithms is still one of several key research problems to be solved.

Recently, Deng et al. [3] proposed PrePost and its enhanced version PrePost+ [5] for mining frequent itemsets. Both of them employ a novel data structure named N-list to represent itemsets and adopt single path property of N-list to directly discover frequent itemsets without generating candidate itemsets in some cases. The experiments in [3], [5] show that PrePost/PrePost+ run faster than some state-of-the-art mining algorithms including FP-growth [8] and FP-growth* [7]. By investigating PrePost+, we found that support checking time for candidate itemsets can be reduced largely if we can stop early the N-list intersection for infrequent candidate itemsets. The same idea holds for other schemes that propose and test potential children itemsets by intersecting lists held in parent itemsets. Two such schemes are Eclat [20] which uses transaction ID lists (TID-lists) and dEclat [17] which uses Diffsets.

In this work, we further improve Eclat/dEclat and PrePost+ by proposing a simple yet effective technique to stop early the support checking of infrequent candidate itemsets in depth-first search. Given an infrequent candidate itemset, the Early Stopping technique accumulates the evidence of infrequency and decides early if the candidate’s support is undoubtedly less than the minimum support, so further checking steps are redundant and dropped. The runtime reduction is always guaranteed, especially on datasets with high ratio between the number of candidates and the number of frequent itemsets.

In the next subsection, we review the mainstream of frequent itemset mining.

A. Related Work

Itemset mining is an important problem of data mining with many variations such as frequent itemset mining [2], [8], [20], frequent closed/maximal itemset mining [7], [18], [19], frequent weighted itemset mining [15], erasable itemset mining [4] and so on. However, frequent itemset mining is still the most popular as it plays an important role in association rule mining [2], sequential mining [6], classification [10]. There have been a large number of algorithms which effectively mine frequent itemsets. We may divide them into three main categories:

• Candidate generate-and-test strategy: Methods in this category use a level-wise (breadth-first-search) approach for mining frequent itemsets. First, they enumerate frequent 1-itemsets which are then used to propose candidate 2-itemsets, and so on until no more candidates can be generated. Apriori [2] is a seminal work in this line of research.

• Divide-and-conquer strategy: Methods using this strategy compress the dataset into a summary structure (e.g., FP-Tree, H-struct) and mine frequent itemsets from this structure by using a divide-and-conquer strategy. They do not propose any candidate itemsets. Instead, frequent itemsets are discovered recursively in sub-databases according to the patterns found. FP-Growth [8], FP-Growth* [7] and H-Mine [11] are representative algorithms in this category. All of them run depth-first search.

• Hybrid strategy: Methods in this category use vertical data formats to summarize the database and mine frequent itemsets by using the generate-and-test strategy. However, the generate-and-test strategy is realized in depth-first manner. TID-list based methods Eclat [20], dEclat [19], and N-list-based methods PrePost/PrePost+ [3], [5] are some typical examples.

B. Contributions and Paper Structure

In this study, we have made the following contributions
II. BACKGROUND

In this section, we review basic concepts of frequent pattern mining and describe a transaction database as running example. The Early Stopping technique is clarified in the next two sections.

A. Frequent Itemsets

We assume a dataset \( DB \) consists of \( n \) transactions such that each transaction contains a number of items belonging to \( I \) where \( I = \{ i_1, i_2, ..., i_m \} \) is the set of all items in \( DB \).

The support of an itemset \( X \subseteq I \), denoted by \( \rho(X) \), is the number of transactions in \( DB \) which contain all the items in \( X \). An itemset \( X \) is a frequent itemset if \( \rho(X) \geq \text{minSup} \), where \( \text{minSup} \) is a given threshold. Note that a frequent itemset with \( k \) elements is called a frequent \( k \)-itemset, and \( F_1 \) is the set of frequent 1-itemsets sorted in frequency ascending or descending order.

Table I shows a \( DB \) of 10 transactions with \( I = \{ a, b, c, d, e \} \). The \( \text{minSup} \) is fixed to 3, i.e., itemsets with frequency at least 3 will be output, e.g., \( \{ a, c \} \) with frequency 4 as it appears in the transactions 3,4,6, and 8. In PrePost+ [5], the items are sorted in decreasing frequency as \( \{ a, c, e, d, b \} \) for PPC-tree because their frequencies are 7,7,7,6, and 3 respectively (see the third column of Table I). In the search tree of Eclat/dEclat, the items are sorted in increasing frequency as \( \{ b, c, a, e \} \) and \( \rho(a) = 4 \). In general, \( k \)-itemset \( P_{xy} \) is proposed and tested by intersecting the TID-lists of two (\( k \)-1)-itemsets \( P_x \) and \( P_y \) (which are both frequent, of course).

We then explore frequent 2-itemsets by intersecting the TID-lists of 1-itemsets. For example, \( T(ac) = T(a) \cap T(c) = \{ 3, 4, 6, 8 \} \), so \( \rho(ac) = 4 \). In general, \( k \)-itemset \( P_{xy} \) is proposed and tested by intersecting the TID-lists of two (\( k \)-1)-itemsets \( P_x \) and \( P_y \) (which are both frequent, of course). For example, we have \( T(da) = \{ 1, 4, 6, 8, 10 \} \) and \( T(dc) = \{ 2, 4, 6, 8 \} \), so \( \rho(dac) = 3 \) (frequent).

B. Downward Closure Property and Depth-First-Search

**Downward closure (or anti-monotone) property** [2]:

\[
\forall X : \forall Y \supseteq X, \rho(Y) \leq \rho(X)
\]

That means if an itemset is extended, its support cannot increase. In other words, no superset of an infrequent itemset can be frequent. This fact suggests that we can start the search from small itemsets to larger ones. In the search process, if we know that an itemset \( X \) is infrequent, we will no longer extend its branch [2]. In the search tree (Fig. 1), the path from the root to a node represents an itemset under consideration with its support, e.g., itemset \( dac \) has support 3.

In depth-first-search schemes like Eclat [20], dEclat [7] and PrePost+ [5], the search tree is expanded and visited in depth-first manner. For instance, the order of 15 found frequent itemsets in Fig. 1 is: \( b, bc, d, da, dac, dace, dc, de, a, ac, ace, ae, c, ce, e \).

In the next sections, we present the Early Stopping technique for Eclat/dEclat and PrePost+ respectively.

III. ECLAT/DCLAT WITH EARLY STOPPING

A. Eclat

In [20], Zaki et al. proposed Eclat, a depth-first-search technique for frequent itemset mining. Its basic idea is based on downward closure as in Apriori but the search is depth-first, not level-wise.

Eclat uses vertical format to represent the database in which each itemset has its own list of transaction ids (TID-list). In Table II TID-lists of each item (1-itemset) is a sorted list of transactions containing the item. The TID-list of an itemset \( X \) is denoted \( T(X) \). We need to read the transaction database once to build the TID-lists of all items.

We then explore frequent 2-itemsets by intersecting the TID-lists of 1-itemsets. For example, \( T(ac) = T(a) \cap T(c) = \{ 3, 4, 6, 8 \} \), so \( \rho(ac) = 4 \). In general, \( k \)-itemset \( P_{xy} \) is proposed and tested by intersecting the TID-lists of two (\( k \)-1)-itemsets \( P_x \) and \( P_y \) (which are both frequent, of course).

For example, we have \( T(da) = \{ 1, 4, 6, 8, 10 \} \) and \( T(dc) = \{ 2, 4, 6, 8 \} \), so \( \rho(dac) = 3 \) (frequent).

| Transaction | Items | Reordering in PrePost+ |
|-------------|-------|------------------------|
| 1           | a, d, e | a, e, d                |
| 2           | b, c, d | a, c, d                |
| 3           | a, c, e | a, c, e                |
| 4           | a, c, d, e | a, c, e, d            |
| 5           | a, e    | a, e                  |
| 6           | a, c, d | a, c, d                |
| 7           | b, c    | c, b                  |
| 8           | a, c, e, d | a, c, c, d          |
| 9           | b, c, e | c, e, b               |
| 10          | a, d, e | a, e, d               |

Table II: Vertical format

| b | d | a | c | e |
|---|---|---|---|---|
| 2 | 1 | 1 | 2 | 1 |
| 7 | 2 | 3 | 3 | 3 |
| 9 | 4 | 4 | 4 | 4 |
| 6 | 5 | 6 | 5 | 5 |
| 8 | 6 | 7 | 8 | 8 |
| 10| 8 | 9 | 10| 10 |
a candidate $P_{xy}$ (Line 11) and to check its support against $minSup$ (Line 12).

Looking at the INTERSECT function (Lines 18-29), we found that its runtime is $O(|U| + |V|)$. If $P_{xy}$ is frequent, we stop only when the condition in Line 20 is violated. However, if $P_{xy}$ is infrequent, we can stop the intersection early.

The basic idea is to keep track of skipped TIDs in $U$ (called $s_U$) and $V$ (called $s_V$) (see Lines 37 and 41 in the function INTERSECT_ES). If the number of items that can be matched in $U$ (i.e., $|U| - s_U$) or in $V$ (i.e., $|V| - s_V$) is less than $minSup$, we will surely know that the intersection between $U$ and $V$ is less than $minSup$, resulting in an infrequent candidate itemset. Simply replacing INTERSECT with INTERSECT_ES helps to reduce the number of comparisons, hence incurring less time to run Eclat.

**Example 3.1:** With $T(b) = \{2, 7, 9\}$ and $T(d) = \{1, 2, 4, 6, 8, 10\}$, INTERSECT($T(b), T(d)$) stops at $i = 4$, $j = 6$ and returns $\{2\}$ while INTERSECT_ES($T(b), T(d)$) stops at $i = 3$, $j = 5$ with $s_U = 1, s_V = 3$, telling us that $|U| - s_U = 3 - 1 < minSup$. □

**B. dEclat**

To reduce memory consumption, Zaki et al. [17] proposed a novel vertical data representation called DiffSet which only stores differences in the TID-list of a candidate itemset from its generating frequent parents.

From a pair of nodes $P_x, P_y$ having the same prefix $P$ in the search tree, the authors of [17] show that the diffset $D(P_{xy}) = D(P_y) - D(P_x)$ and $\rho(P_{xy}) = \rho(P_x) - |D(P_{xy})|$, i.e., we can compute the support of an itemset using its parent’s support and its own diffset. The diffsets are usually smaller than TID-lists, so the memory consumption is reduced.

Fig. 2 illustrates such operations on our running example. At the first level, we store $T(x)$ instead of $D(x)$ for all 1-itemsets $x$, especially on sparse databases. At the second level, $D(xy) = T(x) - T(y)$ [17]. For example, $D(bd) = T(b) - T(d) = \{2, 7, 9\} - \{1, 2, 4, 6, 8, 10\} = \{7, 9\}$, hence, $\rho(bd) = \rho(b) - |D(bd)| = 3 - 2 = 1$ (infrequent).

From the third level, the diffsets are computed directly from parents diffsets, $D(P_{xy}) = D(P_y) - D(P_x)$. For example, $D(dac) = D(dc) - D(da) = \{1, 10\} - \{2\} = \{1, 10\}$, so $\rho(dac) = \rho(da) - |D(dac)| = 5 - 2 = 3$ (frequent).

**Algorithm 1 Eclat [20]**

**Input:** $DB$ : database with $n$ transactions. $minSup$.

**Output:** $F$, the set of all frequent itemsets

1: **procedure** Eclat
2: \hspace{1em} Scan DB to get $T(x)$ for each frequent item $x$.
3: \hspace{1em} $F_1 = F_1 \cup \{T(x)\}$
4: \hspace{1em} $F = F \cup \{x | T(x) \in F_1\}$
5: \hspace{1em} TRAVERSE($F_1$)
6: \hspace{1em} return $F$.
7: **end procedure**
8: **function** TRAVERSE($F_k$) \Comment{depth-first-search}
9: \hspace{1em} $F_{k+1} = \emptyset$
10: \hspace{1em} for $T(P_x), T(P_y) \in F_k, x < y$ do
11: \hspace{2em} $T(P_{xy}) = \text{INTERSECT}(T(P_x), T(P_y))$
12: \hspace{2em} if $|T(P_{xy})| \geq minSup$ then
13: \hspace{3em} $F_{k+1} = F_{k+1} \cup \{T(P_{xy})\}$
14: \hspace{2em} $F = F \cup \{P_{xy}\}$
15: \hspace{1em} if $F_{k+1} ! = \emptyset$ then
16: \hspace{2em} TRAVERSE($F_{k+1}$)
17: **end function**
18: **function** INTERSECT($U, V$)
19: \hspace{1em} $Z = \emptyset, i = 1, j = 1$
20: \hspace{1em} while $i \leq |U|$ AND $j \leq |V|$ do
21: \hspace{2em} if $U[i] == V[j]$ then
22: \hspace{3em} $Z = Z \cup \{U[i]\}$
23: \hspace{2em} $i + +; j + +$
24: \hspace{2em} else if $U[i] < V[j]$ then
25: \hspace{3em} $i + +$
26: \hspace{2em} else
27: \hspace{3em} $j + +$
28: \hspace{1em} return $Z$.
29: **end function**
30: **function** INTERSECT_ES($U, V$) \Comment{early-stopping}
31: \hspace{1em} $Z = \emptyset, i = 1, j = 1, s_U = 0, s_V = 0$
32: \hspace{1em} while $i \leq |U|$ AND $j \leq |V|$ do
33: \hspace{2em} if $U[i] == V[j]$ then
34: \hspace{3em} $Z = Z \cup \{U[i]\}$
35: \hspace{2em} $i + +; j + +$
36: \hspace{2em} else if $U[i] < V[j]$ then
37: \hspace{3em} $i + +, s_U + +$
38: \hspace{2em} if $|U| - s_U < minSup$ then
39: \hspace{3em} break
40: \hspace{2em} else
41: \hspace{3em} $j + +, s_V + +$
42: \hspace{2em} if $|V| - s_V < minSup$ then
43: \hspace{3em} break
44: \hspace{1em} return $Z$.
45: **end function**
1) Early Stopping for dEclat: Main steps of dEclat are shown in Algorithm 2. Similar to Eclat, it starts with the creation of TID-list \(T(x)\) for each frequent 1-itemset \(x\) (Line 2). The depth-first search is delegated to the recursive function TRAVERSE (Lines 8-17). The main step in TRAVERSE is to propose a candidate \(P_{xy}\) (Line 11) and to check its support against \(\minSup\) (Line 12) using the formula \(\rho(P_{xy}) = \rho(P_x) - |D(P_{xy})|\).

Looking at the DIFFERENCE function (Lines 18-31), we found that it runs in time \(O(|U| + |V|)\). If \(P_{xy}\) is frequent, we stop only when the condition in Line 20 is violated. However, if \(P_{xy}\) is infrequent, we can stop the difference early.

The basic idea is to check if the support of \(P_{xy}\) is less than \(\minSup\) after a TID is added to \(Z\) (see Lines 40 and 41 in the function DIFFERENCE_ES). If \(\rho_0 - |Z| < \minSup\), we will surely know that the \(P_{xy}\) is an infrequent candidate itemset. Simply replacing DIFFERENCE with DIFFERENCE_ES helps to reduce the number of comparisons, hence incurring less runtime of dEclat. Note that compared to the intersection operation which is symmetric in Eclat, the difference operation is asymmetric.

Example 3.2: Let \(T = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}\) be the set of all TIDs. We have \(D(b) = T - T(b) = \{1, 3, 4, 5, 6, 8, 10\}\) and \(D(d) = T - T(d) = \{3, 5, 7, 9\}\). DIFFERENCE \((D(b), D(d)) = D(d) - D(b) = \{7, 9\}\) stops at \(i = 5, j = 7\) while DIFFERENCE_ES \((D(b), D(d))\) stops at \(i = 3, j = 6, |Z| = 1\), making \(\rho(b) - |Z| = 2 < \minSup\). \(\square\)

IV. PrePost+ With Early Stopping

In this section, we summarize main concepts of PrePost+ such as PPC-Tree, PP-code and N-list. Then we show how to apply Early Stopping to PrePost+.

A. PPC-tree and N-list

Given a reordered \(DB\), PPC-Tree is a tree structure defined as follows

- It consists of one root labeled as \textit{name} (\(\{\}\)), and a set of item prefix subtrees as children of the root.
- Each node in the item prefix subtree contains five fields: \textit{name}, \textit{frequency}, \textit{childnodes}, \textit{pre}, and \textit{post}. The field

\begin{algorithm}[h]
\caption{dEclat \cite{17}}
\begin{algorithmic}[1]
\Require \(DB\) : database with \(n\) transactions, \(\minSup\).
\Ensure \(F\), the set of all frequent itemsets
\Procedure{DECLAT}{ }
\State Scan DB to get \(T(x)\) for each frequent item \(x\).
\State \(F_1 = F_1 \cup \{T(x)\}\)
\State \(F = F \cup \{x|T(x) \in F_1\}\)
\State TRAVERSE\((F_1)\)
\State \textbf{return} \(F\).
\end{algorithmic}
\end{algorithm}

\begin{algorithm}[h]
\caption{PrePost+ \cite{5}}
\begin{algorithmic}[1]
\Require \(DB\) : database with \(n\) transactions, \(\minSup\).
\Ensure \(F\), the set of all frequent itemsets
\Procedure{PREPOST}{ }
\State Scan DB to get \(T(x)\) for each frequent item \(x\).
\State \(F_1 = F_1 \cup \{T(x)\}\)
\State \(F = F \cup \{x|T(x) \in F_1\}\)
\State TRAVERSE\((F_1)\)
\State \textbf{return} \(F\).
\end{algorithmic}
\end{algorithm}

\begin{algorithm}[h]
\caption{Early Stopping for dEclat}
\begin{algorithmic}[1]
\Procedure{DIFFERENCE}{ }
\If{\(\rho(P_x) - |D(P_{xy})| \geq \minSup\)} \State \(F_k+1 = F_k + \{D(P_{xy})\}\)
\EndIf
\EndProcedure
\end{algorithmic}
\end{algorithm}

\begin{algorithm}[h]
\caption{DIFFERENCE_ES(U, V)}
\begin{algorithmic}[1]
\Procedure{DIFFERENCE}{ }
\State \(Z = \emptyset, i = 1, j = 1\)
\While{\(i \leq |U|\) \AND \(j \leq |V|\)}
\If{\(U[i] = V[j]\)} \State \(i++; j++\)
\ElsIf{\(U[i] < V[j]\)} \State \(Z = Z \cup \{U[i]\}\)
\State \(i++\)
\ElsIf{\(U[i] > V[j]\)} \State \(Z = Z \cup \{V[j]\}\)
\State \(j++\)
\EndIf
\EndWhile
\If{\(i \leq |U|\)} \State \(Z = Z \cup \{U[i]|k = i \rightarrow |U|\}\)
\EndIf
\State \textbf{return} \(Z\).
\end{algorithmic}
\end{algorithm}

\begin{algorithm}[h]
\caption{DIFFERENCE_ES(U, V, \(\rho_0\))}
\begin{algorithmic}[1]
\Procedure{DIFFERENCE}{ }
\State \(Z = \emptyset, i = 1, j = 1\)
\While{\(i \leq |U|\) \AND \(j \leq |V|\)}
\If{\(U[i] = V[j]\)} \State \(i++; j++\)
\ElsIf{\(U[i] < V[j]\)} \State \(Z = Z \cup \{U[i]\}\)
\State \(i++\)
\ElsIf{\(U[i] > V[j]\)} \State \(Z = Z \cup \{V[j]\}\)
\State \(j++\)
\EndIf
\EndWhile
\If{\(\rho_0 - |Z| < \minSup\)} \State \textbf{return} \(Z\).
\Else \State \(j++\)
\EndIf
\EndIf
\If{\(i \leq |U|\)} \State \(Z = Z \cup \{U[k]|k = i \rightarrow |U|\}\)
\EndIf
\State \textbf{return} \(Z\).
\end{algorithmic}
\end{algorithm}
Tree by creating nodes named a
Then the first transaction
Similarly, for the second transaction
of the root and two descendent nodes are added. The third and
1-itemsets.

The pre-order and post-order ranks are tagged in a pair of
N-list of k-itemset [3] of each node
PP-code

**Example 4.1:** We have \( e < c \), \( NL(e) = \{ < 1, 1, 3 > , < 4, 3, 3 > , < 11, 11, 1 > \} \) and \( NL(c) = \{ < 3, 5, 4 > , < 7, 12, 3 > \} \), therefore \( NL(ec) = \{ < 3, 5, 3 > , < 7, 12, 1 > \} \) and the support of \( ec \) is \( \rho(ec) = 3 + 1 = 4 \) (see Fig. 6).

**B. PrePost+ Algorithm**

In this section, we briefly recall the PrePost+ algorithm [5] (see Algorithm 3). PrePost+ starts with the construction of PPC-Tree (Line 1) and computation of \( N\)-list of frequent 1-itemsets (Line 2). Again, the idea of depth-first search in Eclat/dEclat repeats here. Recall that PrePost+ combines itemsets sharing the same suffix (not prefix as in Eclat/dEclat).

The main steps of \( NL_{\text{intersect}} \) are depicted in Lines 19-33 (Algorithm 3). Similar to the function \( \text{INTERSECTION} \) in Eclat, we maintain two indexes \( i \) and \( j \) and carry out the intersection from left-to-right. The criteria for the merge (Lines 23,24) are stated in Section 4.3.1 i.e., the \( i \)-th triple in \( U \) is mergeable to the \( j \)-th triple in \( V \) if and only if the former is the ancestor of the latter in the PPC-Tree.

Again with Example 4.1, the step-by-step intersection between \( NL(e) \) and \( NL(c) \) is as follows. \( < 3, 5, 4 > \) is non-mergeable to \( < 1, 1, 3 > \) so it is tested against the next \( j \), i.e., \( < 4, 3, 3 > \) where it is mergeable and returns \( < 3, 5, 3 > \). Then \( < 3, 5, 4 > \), when compared to \( < 11, 11, 1 > \), fails at Line 6, so we consider the next \( i \), i.e., \( < 7, 12, 3 > \). Clearly, \( < 7, 12, 3 > \) is mergeable to \( < 11, 11, 1 > \), returning \( < 7, 12, 1 > \). The intersection stops and we get \( NL(ec) = \{ < 3, 5, 3 > , < 7, 12, 1 > \} \).
Note that NL_intersect is fixed by the item order, i.e., we only intersect NL(xS) with NL(yS) if x < y in frequency ordering.

C. Early Stopping for PrePost+

In PrePost+, NL_intersect runs in O(|U| + |V|). To apply Early Stopping technique, we integrate again the size test into the function NL_intersect in order that if the test fails early, we can stop the computation and return an empty Z.

We present this idea in the function NL_intersect_ES (Lines 34-52). At any triple j of V, if it is non-mergeable to the triple i of U, we increase skip by y_j.freq (Line 44). If the sum of remaining frequencies ρ_V - skip is less than minSup, we stop and return an empty set (Lines 45,46). We demonstrate the effectiveness of NL_intersect_ES in the next example.

Example 4.2: Given NL(b) = \( \{<9, 7, 1>, 10, 9, 1>, <12, 10, 1>, 5, 2, 2>, <6, 4, 1>, <8, 8, 1> \) and NL(d) = \( \{<2, 0, 2>, 5, 2, 2>, <6, 4, 1>, <8, 8, 1> \) if we call NL_intersect(NL(b),NL(d)), we need to run 5 checks for \((i,j) = (1,1), (1,2), (1,3), (1,4), (2,4)\) in which only pair (1,4) matches, so we get NL(bd) = \( \{<8, 8, 1>\} \). With support 1 (less than minSup = 3), db is infrequent.

In calling NL_intersect_ES(NL(b),NL(d)), we know that ρ_V = 6, minSup = 3. After the two (failed) checks \((i,j) = (1,1), (1,2)\), we increase skip to 2 + 2 = 4, making ρ_V - skip < minSup, so we safely conclude that bd is not frequent, omitting the three remaining checks.

D. Remarks on Apriori, FP-Growth, and Bit-Vector Based Algorithms

Apriori [2] is a level-wise (breadth-first search) mining scheme which use horizontal format to count the support for candidate k-itemsets (i.e., itemsets at level k). No list intersection is required in Apriori, so our technique does not apply.

Instead of generating and testing candidate itemsets, FP-Growth [8] and its derivatives FP-Growth* [7], H-Mine [11] recursively project the database into sub-databases using prefix itemsets. Then local frequent patterns are searched to assemble longer global ones. No list intersection is required in FP-Growth/FP-Growth* or H-Mine, so our technique does not apply either.

Algorithm 3 PrePost+

Input: DB : database with n transactions. minSup : minimum support.
Output: F, the set of all frequent itemsets

1: procedure PrePost+
2: Scan DB to obtain F_1 and build the PPC-Tree
3: Scan PPC-tree to generate NL(x)
4: \( F_1 = F_1 \cup \{NL(x)\} \)
5: \( F = F \cup \{x|NL(x) \in F_1\} \)
6: \text{TRAVERSE}(F_1)
7: return F.
8: end procedure

9: function \text{TRAVERSE}(F_k) \triangleright depth-first-search
10: \( F_{k+1} = \emptyset \)
11: for NL(xS), NL(yS) \in F_k, x < y do
12: \( NL(xyS) = NL_intersect(NL(xS), NL(yS)) \)
13: if \( \rho(xyS) \geq minSup \) then
14: \( F_{k+1} = F_{k+1} \cup \{NL(xyS)\} \)
15: \( F = F \cup \{xyS\} \)
16: if \( F_{k+1}! = \emptyset \) then
17: \text{TRAVERSE}(F_{k+1})
18: end function

19: function NL_intersect(U, V)
20: \( i = 1, j = 1 \)
21: \( Z = \emptyset \)
22: while \( x_i \in U, i \leq |U| \) AND \( y_j \in V, j \leq |V| \) do
23: if \( x_i.pre > y_j.pre \) then
24: if \( x_i.post < y_j.post \) then
25: add \( <y_j.pre, y_j.post, x_i.freq > \) to Z
26: \( i += \)
27: else
28: \( j += \)
29: else
30: \( i += \)
31: merge elements in Z
32: return Z.
33: end function

34: function NL_intersect_ES(U, V, \rho_V) \triangleright early-stopping
35: \( i = 1, j = 1 \)
36: \( skip = 0 \)
37: \( Z = \emptyset \)
38: while \( x_i \in U, i \leq |U| \) AND \( y_j \in V, j \leq |V| \) do
39: if \( x_i.pre > y_j.pre \) then
40: if \( x_i.post < y_j.post \) then
41: add \( <y_j.pre, y_j.post, x_i.freq > \) to Z
42: \( i += \)
43: else
44: \( skip = skip + y_j.freq \)
45: if \( \rho_V - skip < minSup \) then
46: return \( \emptyset \)
47: \( j += \)
48: else
49: \( i += \)
50: merge elements in Z
51: return Z.
52: end function
Bit-vector based algorithm such as VIPER [12] applies depth-first search in the same manner as Eclat but uses a compressed bit-vector structure instead. The intersection of decompressed bit-vectors in memory is performed by AND operator. Our technique can be plugged to such algorithms to early determine if the intersection would be less than \( \text{minSup} \) or not.

V. EXPERIMENTS

In this section, we evaluate the performance of the proposed Early-Stopping technique applied to Eclat/dEclat and PrePost+ in terms of runtime and number of comparisons. The datasets are described in Sections [V-A] We show the comparison between standard versions and early-stopping versions in Section [V-B] The algorithms are implemented in C++ and run on a desktop PC with Intel® Core i7-6700@ 3.4Ghz, 16GB memory.

A. Experiment Setup

We use nine datasets as shown in Table III. The datasets were downloaded from FIMI repository (http://fimi.ua.ac.be) and KONECT repository (http://konect.uni-koblenz.de/networks). The columns #Items, #Trans, Avg.Length and minSup show the number of items, number of transactions, average transaction length and the range of \( \text{minSup} \) value respectively.

T40I10D100K is a synthetic market-basket dataset from [2]. It contains 100,000 transactions and 942 items.

MovieLens-1M is a bipartite network containing one million movie ratings from [http://movielens.umn.edu]. Movies play the role of items and ratings of each user stand for a transaction.

Github is a membership network of the hosting site GitHub. The network is bipartite and contains users (transactions) and projects (items).

Retail is anonymous retail market-basket data from an anonymous Belgian retail store.

Kosarak contains sequences of click-stream data from a Hungarian news portal.

Accidents contains anonymous traffic accident data.

Chess is converted from UCI chess dataset. Each transaction is an instance of the chess game and items describe the board and the outcome of the game.

Connect is converted from UCI connect-4 dataset. Each transaction is an instance of the game and items describe the board and the outcome of the game.

Pumsb dataset contains census data for population and housing.

We name the Early-Stopping versions as Eclat-ES, dEclat-ES and PrePost+ES. Recall that our technique is easily plugged to any frequent pattern mining schemes that require the intersection operation for itemset support checking.

B. Effectiveness of Early Stopping Technique

In this section, we evaluate the effectiveness of Early-Stopping schemes Eclat-ES, dEclat-ES and PrePost+ES. Because the schemes run deterministically, all the reported values except runtime do not change. The runtime is the average result of ten runs.

Table III: Dataset properties

| Dataset          | #Items | #Trans | Avg.Length | minSup      |
|------------------|--------|--------|------------|-------------|
| T40I10D100K      | 942    | 100,000| 39.6       | 0.002 .. 0.02|
| MovieLens-1M     | 3,706  | 6,040  | 165.6      | 0.07 .. 0.1  |
| Github           | 56,519 | 120,867| 3.6        | 0.00007 .. 0.00001|
| Retail           | 16,470 | 88,162 | 10.3       | 0.00003 .. 0.00006|
| Kosarak          | 41,270 | 900,002| 8.1        | 0.001 .. 0.004 |
| Accidents        | 468    | 540,183| 33.8       | 0.1 .. 0.4   |
| Chess            | 75     | 3,196  | 37.0       | 0.1 .. 0.4   |
| Connect          | 129    | 67,557 | 43.0       | 0.1 .. 0.4   |
| Pumsb            | 2,088  | 49,046 | 50.5       | 0.1 .. 0.4   |

1) Number of Proposed Candidates and Expanded Nodes: Table IV displays the number of proposed candidates (column #Cands), expanded nodes (column #Nodes) and the ratio between them (column Ratio) on the search tree for each dataset and different values of \( \text{minSup} \). Here \( \text{minSup}_1 \) means the smallest value of \( \text{minSup} \) for the dataset, \( \text{minSup}_2 \) means the next value and so on (see Table III).

Because Eclat/dEclat and PrePost+ traverse the search tree based on items sorted in increasing frequency, the number of proposed candidates and expanded nodes are the same for all the schemes on a given dataset and \( \text{minSup} \). As \( \text{minSup} \) increases, there are less frequent 1-itemsets, so the number of proposed candidates and expanded nodes get smaller too.

We can roughly divide the datasets into two groups by the ratio between the number of candidates and expanded nodes. The first four datasets have the ratio larger than 2 while the remaining have the ratio less than 1.5. As we will see in the following subsections, the ratio suggests different behaviours of mining schemes in both the number of comparisons and runtime.

2) Number of Comparisons: Figures 7 to 15 compare six schemes over nine datasets for different values of \( \text{minSup} \). In each figure, we report the number of comparisons performed in intersection functions on the left and the total runtime (in second) on the right.

First, the Early-Stopping schemes effectively reduce the number of comparisons between pairs of TID-lists (Eclat), Diffsets (dEclat) or N-lists (PrePost+) in all cases. The reduction varies among datasets and mining schemes.

For Eclat-ES, the number of comparisons is cut down considerably in the first three datasets T40I10D100K, MovieLens-1M and Github. The reduction is clear cut for small values of \( \text{minSup} \) and slightly decreases when \( \text{minSup} \) becomes larger.

dEclat-ES and PrePost+ES confirm similar effective reduction on T40I10D100K, MovieLens-1M, Github, Retail and Kosarak.

Finally, we observe that the reduction of comparison operations in Accidents, Chess, Connect and Pumsb is almost negligible. This result can be explained by the ratio column of Table IV. These datasets also exhibit large discrepancies between Eclat and dEclat/PrePost+, confirming that TID-list is much less efficient on these kinds of transaction data.

3) Runtime: The reduction in the number of comparisons naturally translates into the reduction of runtime (see the right plots of Figures 7 to 15). The clear effect is observed in the four datasets and in Eclat-ES for the remaining five datasets.
Figure 7: Number of comparisons and runtime for T40I10D100K

Figure 8: Number of comparisons and runtime for MovieLens-1M

Figure 9: Number of comparisons and runtime for Github

Figure 10: Number of comparisons and runtime for Retail
Figure 11: Number of comparisons and runtime for Kosarak

Figure 12: Number of comparisons and runtime for Accidents

Figure 13: Number of comparisons and runtime for Chess

Figure 14: Number of comparisons and runtime for Connect
Table IV: Number of proposed candidates and expanded nodes

| Dataset          | minSup = minSup1 | minSup = minSup2 | minSup = minSup3 |
|------------------|------------------|------------------|------------------|
|                  | #Cands | #Nodes | Ratio | #Cands | #Nodes | Ratio | #Cands | #Nodes | Ratio |
| T40I10D100K      | 2.16e+07 | 8.99e+06 | 2.41 | 3.22e+06 | 1.25e+06 | 2.51 | 3.31e+05 | 6.52e+04 | 5.07 |
| MovieLens-1M     | 4.24e+07 | 2.29e+07 | 1.85 | 1.00e+07 | 4.86e+06 | 2.06 | 3.12e+06 | 1.36e+06 | 2.30 |
| Github           | 1.11e+08 | 1.94e+07 | 5.71 | 4.99e+07 | 8.30e+06 | 6.02 | 2.65e+07 | 4.07e+06 | 6.51 |
| Retail           | 3.05e+07 | 1.84e+06 | 16.61 | 1.28e+07 | 9.68e+05 | 13.24 | 6.57e+06 | 6.10e+05 | 10.76 |
| Kosarak          | 9.31e+05 | 6.26e+05 | 1.49 | 4.64e+04 | 3.59e+04 | 1.29 | 6.37e+03 | 4.98e+03 | 1.28 |
| Accidents        | 1.01e+07 | 9.96e+06 | 1.02 | 9.08e+05 | 8.87e+05 | 1.02 | 1.55e+05 | 1.50e+05 | 1.03 |
| Chess            | 1.65e+06 | 1.55e+08 | 1.06 | 2.75e+07 | 2.50e+07 | 1.06 | 6.06e+06 | 5.71e+06 | 1.06 |
| Connect          | 1.01e+07 | 8.04e+06 | 1.26 | 1.84e+06 | 1.48e+06 | 1.24 | 5.78e+05 | 4.60e+05 | 1.26 |
| Pumsb            | 7.23e+06 | 5.47e+06 | 1.32 | 3.98e+05 | 2.96e+05 | 1.34 | 4.01e+04 | 2.92e+04 | 1.37 |

Figure 15: Number of comparisons and runtime for Pumsb

Note that the reduction in runtime must take into account the offset caused by the Early-Stopping checks (i.e., Lines 38 and 42 in Algorithm [1] Line 40 in Algorithm [2] and Line 45 in Algorithm [3]). If the candidate itemset is frequent, such checks make Early-Stopping intersection functions incur a small overhead compared to the standard counterparts. For datasets whose number of comparisons is not much saved, the runtime reduction is not guaranteed. This fact is clearly observed in several cases, especially for dEclat-ES and PrePost+ES on Kosarak.

4) Other Remarks: As TID-lists (in Eclat) and Diffsets (in dEclat) are two complementary structures, we observe an interesting tendency: the high number of comparisons or runtime in one scheme implies the low corresponding values in the other.

All the enhanced versions have the same memory consumption as the original schemes. This fact is straightforward because the memory requirement to maintain the support structures like TID-lists, Diffsets and N-lists is unchanged. The number of proposed candidates does not change either.

VI. CONCLUSION

We have presented a simple yet effective Early-Stopping technique to accelerate some existing depth-first search itemset mining algorithms that use the generate-and-test strategy. Our technique is based on an early-stopping criterion for list intersection. We have applied the technique to TID-list in Eclat, diffsets in dEclat and N-list in PrePost+. The number of comparisons in the enhanced versions is always less than that in the original algorithms, leading to runtime cut-down in most of the cases. We have evaluated the Early-Stopping schemes over nine datasets. The results confirm the effectiveness of our improvement and suggest what kind of transaction data will benefit most.

CONFLICTS OF INTEREST

The author(s) declare(s) that there is no conflicts of interest regarding the publication of this paper.

REFERENCES

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic sub-space clustering of high dimensional data for data mining applications. In SIGMOD’98. ACM, 1998.
[2] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In Proc. 20th int. conf. very large data bases, VLDB, volume 1215, pages 487–499, 1994.
[3] Z. Deng, Z. Wang, and J. Jiang. A new algorithm for fast mining frequent itemsets using n-lists. Science China Information Sciences, 55(9):2008–2030, 2012.
[4] Z.-H. Deng, G.-D. Fang, Z.-H. Wang, and X.-R. Xu. Mining erasable patterns. In Machine Learning and Cybernetics, 2009 International Conference on, volume 1, pages 67–73. IEEE, 2009.
[5] Z.-H. Deng and S.-L. Lv. PrePost+: An efficient n-lists-based algorithm for mining frequent itemsets via children–parent equivalence pruning. Expert Systems with Applications, 42(13):5424–5432, 2015.
[6] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas. A survey of sequential pattern mining. Data Science and Pattern Recognition, 1(1):54–77, 2017.
[7] G. Grahe and I. Zhu. Fast algorithms for frequent itemset mining using fp-trees. IEEE transactions on knowledge and data engineering, 17(10):1347–1362, 2005.
[8] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In ACM sigmod record, volume 29, pages 1–12. ACM, 2000.
[9] Y.-K. Lee, W.-Y. Kim, Y. D. Cai, and J. Han. Comine: Efficient mining of correlated patterns. In ICDM, volume 3, pages 581–584, 2003.
[10] L. T. Nguyen, B. Vo, T.-P. Hong, and H. C. Thanh. Classification based on association rules: A lattice-based approach. Expert Systems with Applications, 39(13):11357–11366, 2012.
[11] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang. H-mine: Hyper-structure mining of frequent patterns in large databases. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 441–448. IEEE, 2001.

[12] P. Shenoy, J. R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah. Turbo-charging vertical mining of large databases. In *ACM Sigmod Record*, volume 29, pages 22–33. ACM, 2000.

[13] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *International Conference on Extending Database Technology*, pages 1–17. Springer, 1996.

[14] H. Toivonen et al. Sampling large databases for association rules. In *VLDB*, volume 96, pages 134–145, 1996.

[15] B. Vo, F. Coenen, and B. Le. A new method for mining frequent weighted itemsets based on wittrees. *Expert Systems with Applications*, 40(4):1256–1264, 2013.

[16] C.-W. Wu, Y.-F. Lin, P. S. Yu, and V. S. Tseng. Mining high utility episodes in complex event sequences. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 536–544. ACM, 2013.

[17] M. J. Zaki and K. Gouda. Fast vertical mining using diffsets. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 326–335. ACM, 2003.

[18] M. J. Zaki and C.-J. Hsiao. Charm: An efficient algorithm for closed itemset mining. In *Proceedings of the 2002 SIAM international conference on data mining*, pages 457–473. SIAM, 2002.

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

[20] M. J. Zaki, S. Parthasarathy, M. Ogihara, W. Li, et al. New algorithms for fast discovery of association rules. In *KDD*, volume 97, pages 283–286, 1997.