Index Support for Mining Data Streams in a Relational DBMS

Elena Baralis, Tania Cerquitelli, Silvia Chiusano, Diego Mostile

Politecnico di Torino - Corso Duca degli Abruzzi, 24 - 10129 Torino, ITALY
{elena.baralis, tania.cerquitelli, silvia.chiusano, diego.mostile}@polito.it

Abstract. This paper presents a novel index, called I-Forest, to support data mining activities on data streams, i.e., sequences of incoming data blocks. This approach is appropriate for itemset extraction on evolving datasets such as analysis of transactional data streams from retail chains. The index is a covering structure that represents transaction blocks in a succinct form and allows different kinds of analysis (e.g., analyze quarterly data). During the creation phase no support constraint is enforced, thus the index provides a complete representation of the data stream. The I-Forest index has been implemented into the PostgreSQL open source DBMS and exploits its physical level access methods. Preliminary experiments have been run to validate the proposed approach.

1 Introduction

Since many real life data are streams of information, data mining faces new challenges. Examples of real life data include streams of transactional data from large retail chains, web server logs, financial stock tickers, call-detail records and network traffic monitoring. Data can be described as a sequence of incoming data blocks, where new blocks arrive periodically. Different kinds of analysis could be performed over such data like (i) mining all available data (ii) mining only the most recent data (e.g., last month data), (iii) mining periodical data (e.g., quarterly data) and (iv) mining selected data blocks (e.g., data related to the first month of last year and the first month of this year). Consider for example streams of transactional data from large retail chains, where every day, after shop closing, a set of transactions is added to the database [8]. In this scenario, market analysts are often interested in analyzing different subsets of transactions to discover customer behaviors. For example, they may be interested in analyzing purchases before Christmas or during summer holidays.

In this paper we address itemset extraction on sequences of incoming data blocks. We propose an index structure, called Itemset-Forest (I-Forest), for mining data streams. It allows the extraction of the complete set of itemsets which satisfy (i) time constraints (which temporal data we are interested in), (ii) item constraints (itemsets containing a selected subset of items), and (iii) support constraints (minimum itemset frequency). The index has been implemented into the open source PostgreSQL DBMS [13].
Since the I-Forest index includes all attributes potentially needed during the extraction task, it is a covering index. Hence, the extraction can be performed by means of the index alone, without accessing the database. The data representation is complete, i.e., no support threshold is enforced during the index creation phase, in order to allow reusing the index for mining itemsets with any support threshold.

The I-Forest index is characterized by a set of compact structures, one for each incoming block. Each structure provides a locally compact representation of the data block. An interesting feature of the index is its ability to represent distinct data blocks by means of different data structures. Hence, the data structure of a block can be adapted to the data distribution. Furthermore, the physical organization of the I-Forest index allows selective access to the needed information, thus reducing the overhead in accessing the disk blocks during the extraction task. A notable feature of the block partitioning approach is its ability to deal with datasets of significant size by partitioning them in smaller, manageable chunks without incurring a significant overhead due to partitioning.

The paper is organized in the following way. Section 2 discusses related work on mining data streams. Section 3 defines the problem addressed in this paper. Section 4 describes the structure of the proposed index, while Section 5 presents algorithms to extract itemsets with time, item and support constraints. Section 6 discusses preliminary experiments to evaluate the effectiveness of the proposed index. Finally, Section 7 draws conclusions and presents future developments of the proposed approach.

2 Related Work

The wide diffusion of streaming data caused an increased interest in performing data mining activities over continuous data flows. Incoming data may arrive as (i) single transactions, or (ii) transaction blocks. A general framework for mining data streams is proposed in [8]. Since a set of transactions (block) is added to the database at the same time, the data model is called block evolution. Different data mining techniques may fit in the proposed framework. Since the I-Forest index structure addresses itemset extraction under block evolution, it may be straightforwardly integrated into the framework proposed in [8].

Many algorithms have been proposed to perform the computationally intensive knowledge extraction task over static databases [1, 10–12]. Since the required memory increases significantly, memory based approaches are not suitable for mining data streams. Disk based approaches [3, 6, 14], which exploit clever data structures to summarize the (static) database on disk, are more suitable for itemset extraction from data streams. In our previous work [3] we proposed an index structure, called I-Tree, to tightly integrate frequent itemset mining in PostgreSQL open source DBMS. The I-Tree is a covering index and its performance is comparable to the state of the art Prefix-tree algorithm [10] on flat file. However, the I-Tree index cannot be incrementally maintained. The I-Forest index improves the I-Tree structure in several directions. It allows (i) item con-
strained extraction, (ii) incremental mining of data streams and (iii) smooth data evolution.

The problem of incremental frequent pattern mining has been addressed in [2, 4, 5]. Works in [2, 4] address incremental frequent itemset extraction under block evolution, while [5] addresses arbitrary insertions and deletions of single transactions. The I-Forest index allows selective access to the information required by the mining process. Hence, differently from the above approaches, it provides a flexible framework in which different types of analysis (e.g., mining only a subset of interesting blocks or items) can be performed.

3 Problem statement

Let \( \mathcal{I} = \{i_1, i_2, \ldots, i_n\} \) be a set of items. A transactional database block \( b \) is a collection of transactions, where each transaction \( T \) is a set of items in \( \mathcal{I} \). Let \( b_k \) be the instance of \( b \) arrived at time \( k \), where \( k \) is used as time identifier. A transactional database \( \mathcal{D} \) at time \( t \) is a finite sequence of data blocks \( \{b_1, b_2, \ldots, b_t\} \) denoted as \( \mathcal{D}[1..t] \). The distribution of items in blocks may be skewed.

\( \mathcal{D}[1..t] \) can be represented as a relation \( R \), where each tuple is a triplet \((\text{Time}, \text{TransactionId}, \text{ItemId})\). When a new block arrives, its transactions are added to \( R \) and \( t \) is updated. Analysis can be performed on sets of blocks not necessarily in sequence. We denote as \( \Omega \) the set of time identifiers of the analyzed blocks (e.g., \( \Omega = \{1, 2, 5\} \) means that we are interested in blocks \( \{b_1, b_2, b_5\} \)). \( R_\Omega \subseteq R \) is the set of tuples associated to the blocks in \( \Omega \).

Given a set of constraints \( \psi \), the itemset extraction task is the extraction of the complete set of itemsets in \( R \) which satisfy \( \psi \). Constraints in \( \psi \) are among the following: (i) Time constraint, which allows the selection of a subset of blocks in \( R \) by means of \( \Omega \). (ii) Support constraint, which defines the minimum support threshold to perform itemset extraction. (iii) Item constraint, which selects from \( \mathcal{I} \) a subset of interesting items for extraction.

4 I-Forest Index Structure

The I-Forest index supports the extraction of itemsets from a data stream, i.e., from the sequence of blocks incoming in relation \( R \). Since the index should be reusable for extraction sessions with different constraints, the data representation is complete, i.e., no support threshold is enforced during the index creation phase.

The I-Forest is a covering index, i.e., it allows itemset extraction without accessing relation \( R \). It includes both time and item identifier attributes, which may be needed during the extraction task. Each block is stored by means of a compact structure, which provides a locally compact representation of \( R \). Each compact structure can adopt a different data representation. Hence, I-Forest can easily adapt to heterogeneous data distributions, where each block is characterized by a different data distribution.

The I-Forest index includes two elements: the Itemset Forest-tree (IF-tree), and the Itemset Forest-Btree (IF-Btree). The first encodes the transaction data
stream. It is a forest of compact structures, called Itemset Forest-blocks, that evolves under time evolution. Each IF-block provides a lossless compact representation of a transactional block $b_k$. Many different compact structures could be adopted (e.g., FP-tree [11], Inverted-Matrix [6], Patricia-Tries [12], I-Tree [3]). Currently each IF-block is based on the I-Tree [3]. The second, IF-Btree, allows selective access to the index (disk) blocks and links information belonging to different IF-block to support efficient retrieval of information during the mining process. In the following we describe in more detail each structure.

**IF-tree** associated to $R$. It is a forest of I-Tree, one for each IF-block. For each relational block $R_k$, an I-Tree ($I_{tk}$) and a Physical Block Selector ($Pbs_k$) are built [3]. The $I_{tk}$ associated to $R_k$ is a prefix tree, where each node corresponds to an item and each path encodes one or more transactions in $R_k$. Each node is associated with a *node support value*. This value is the number of transactions in $R_k$ which contain all the items included in the subpath reaching the node. Each item is associated to one or more nodes in the same $I_{tk}$.

$Pbs_k$ has one entry for each item in $R_k$, for which it reports the $I_{tk}$-*item-support value*, i.e., the item frequency in $R_k$. The $I_{tk}$-item-support value is obtained by adding the supports of all nodes in $I_{tk}$ associated to the item. The *global item support* value is the frequency of the item in $R_k$. This value is obtained by adding the $I_{tk}$-item-supports of the item for each block $b_k$.

---

<table>
<thead>
<tr>
<th>Time</th>
<th>TID</th>
<th>Items</th>
<th>Time</th>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>g,b,h,e,p,v,d</td>
<td>2</td>
<td>1</td>
<td>l</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>e,m,h,n,d,b</td>
<td>2</td>
<td>2</td>
<td>l,q</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>p,c,i,f,o,h</td>
<td>2</td>
<td>3</td>
<td>e,l,q,r</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>j,h,k,a,w,e</td>
<td>2</td>
<td>4</td>
<td>b,d,e,l,q,r</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>n,b,d,e,h</td>
<td>2</td>
<td>5</td>
<td>e,f,l,q,r,z</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>s,a,n,r,b,u,i</td>
<td>2</td>
<td>6</td>
<td>d,x</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>b,g,h,d,e,p</td>
<td>2</td>
<td>7</td>
<td>a,c,d,e,x,z</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>a,i</td>
<td>2</td>
<td>8</td>
<td>a,c,e,f,g,x</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>f,i,e,p,c,h</td>
<td>2</td>
<td>9</td>
<td>a,b,c,x</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>t,h,a,e,b,r</td>
<td>2</td>
<td>10</td>
<td>q,r,x</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>a,r,e,b,h</td>
<td>2</td>
<td>11</td>
<td>b,d,g,h,q,r,x</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>z,b,i,a,n,r</td>
<td>2</td>
<td>12</td>
<td>a,b,c,d,q,r,x,z</td>
</tr>
</tbody>
</table>

Fig. 1. Transaction Block at time at $t=1$  
Fig. 2. Transaction Block at time $t=2$

Figures 1 and 2 report (in a more succinct form than the actual relational representation) two data blocks, arrived at times $t = 1, 2$, used as a running example. Figure 3 shows the structure of the corresponding IF-tree at time $t=2$. It only shows the two I-Tree $I_{t1}$ and $I_{t2}$ associated to the two blocks. It omits the corresponding $Pbs_{t1}$ and $Pbs_{t2}$ to easy readability. Consider item $e$ in $I_{t1}$. Its $I_{t1}$-item-support is 9 (there are two nodes associated to item $e$, the first has node support 6 and the second 3). Furthermore, global item support of $e$ is 13 (its $I_{t1}$-item-support is 9 and its $I_{t2}$-item-support is 4).
As shown in Figure 3, in each $I_t_k$ nodes are clustered in three layers: top, middle and bottom. Correlation analysis is exploited to store in the same (disk) block nodes accessed together during the mining process to reduce the number of reads. Clustering strategy details are described in [3].

Nodes in the I-Forest are linked by means of pointers which allow the retrieval from disk of the index portion required by the mining task. Each node $N_i$ in the I-Forest is provided with pointers to three index blocks: (i) the block including its father, (ii) the block including its first son, and (iii) the block including its left brother. The pointers allow both bottom-up and top-down tree traversal, thus enabling itemset extraction both with item and support constraints. To ease readability only pointers of the first type are shown in Figure 3.

**IF-Btree** is an additional structure of the IF-tree, which has one entry for each item in relation $R$. It allows selective access to the IF-tree blocks during the mining process. Each IF-Btree leaf contains a pair (node physical address, block identifier) for each node in which the item appears.

**I-Forest index storage.** The I-Forest is stored in two relational tables. Table $TPBS$ stores all $Pbs_k$, while table $TIFOREST$ stores every $I_t_k$. The IF-Btree is stored in a B-Tree structure. To access records in $TIFOREST$, in $TPBS$ and in the IF-Btree, we use functions available in the access methods of PostgreSQL [13]. Table $TPBS$ contains one record for each item that appears in each $I_t_k$. Each record contains data block identifier, item identifier, and $I_t_k$-item-support. Table $TIFOREST$ contains one record for each node in $I_t_k$. Each record contains node identifier, item identifier, local node support, physical location (number of the block and offset within the block) of its father, physical location (number of the block and offset within the block) of its first child, and physical location (number of the block and offset within the block) of its left brother.

---

**Fig. 3. IF-tree at time $t = 2$**
5 Mining Itemsets

This section describes how itemset extraction with different constraints exploits the I-Forest index.

The time constraint is expressed by means of a binary mask, with one bit for each data block \( b_k \) in \( R \). If the bit is set, the corresponding data block is analyzed. When no time constraint is enforced, all bits in the mask are set.

The algorithm for item and support constrained analysis is organized in two phases. (i) Retrieval of necessary data. The IF-Btree selectively loads in memory only the index blocks used for the local search in the current extraction phase. (ii) Itemset extraction. It is performed by considering only one item at a time and extracting all itemsets which include the considered item. Each \( I_{t_k} \) considered in the analysis is traversed upward and downward to build in memory the projected tree for the considered item. From the projected tree itemsets are then extracted by means of an fp-growth based algorithm. Our current implementation of the extraction algorithm is based on a modified version of the algorithm in [3], which is extended to perform both upward and downward traversal of all considered \( I_{t_k} \) in I-Forest. To perform itemset extraction, a temporal structure, called \( \text{TempT}_{PBS} \), is built. \( \text{TempT}_{PBS} \) lists all the items that appear in the selected blocks. For each item, the joined support value, i.e., the sum of the \( I_{t_k} \)-item-supports for the selected blocks, is computed. This value is stored in \( \text{TempT}_{PBS} \).

When the support constraint is specified, we select from \( \text{TempT}_{PBS} \) only frequent items, which are analyzed one by one. For each item, the corresponding entries are selected from the IF-Btree. Next, all itemsets including the current item are extracted from all selected \( I_{t_k} \) by traversing the trees both upward and downward.

The item constraint forces the selection of the entries corresponding to the selected item from the IF-Btree. The extraction is then performed as described above. In the general case, the item constraint is first considered and the support constraint is then enforced during the extraction process.

6 Preliminary Experimental Results

We ran some preliminary experiments to evaluate the effect of block partitioning on the performance of frequent itemset extraction. We considered as a representative dataset the click-stream dataset Kosarak [7] (1,017,029 transactions, 41,244 items), which we denote as Kosarak-100. To simulate block evolution, we considered two different configurations. For the first, denoted as Kosarak-50+50, we split Kosarak-100 in two blocks. The first block is characterized by 508,515 transactions and 35,586 items, and the second by 508,514 transactions and 35,139 items. The second configuration, denoted as Kosarak-100+100, is composed by two identical blocks obtained by cloning Kosarak-100.

Both the index creation procedure and the itemset extraction algorithm were coded into the PostgreSQL open source DBMS [13]. The algorithm has been
developed in ANSI C. Experiments have been performed on a 2800Mhz Pentium IV PC with 2Gbyte main memory. The buffer cache of PostgreSQL DBMS has been set to the default size of 64 blocks (block size is 8Kbyte).

To evaluate the performance of our approach we considered Kosarak-50+50 dataset. We compared with the state of the art algorithms [9, 10] for frequent itemsets mining on the Kosarak-100 dataset on flat file. Unfortunately, it has not been possible to perform the extraction task with FP-growth [9] in the considered support range. Figure 4(a) reports the run times\(^1\) for the extraction of frequent itemsets for varying support thresholds. Our approach, albeit implemented into a relational DBMS, yields performance comparable to the Prefix-tree [10] on flat file.

To evaluate the effect of block partitioning, we ran our extraction algorithm on three different dataset configurations: (i) Kosarak-100, which represents our lower bound on performance, since it has no overhead due to block partitioning, (ii) Kosarak-50+50, which is a split of Kosarak-100, with each block characterized by a similar statistical pattern, and (iii) Kosarak-100+100, which represents the worst case data distribution, where the two blocks are identical. Figure 4(b) reports the run times for the extraction of frequent itemsets for varying support thresholds. Block partitioning (Kosarak-50+50 with respect to Kosarak-100) does not yield significant performance difference. The difference decreases with decreasing support thresholds. The worst case (Kosarak-100+100) is never

\(^1\) We neglect time for writing generated itemsets.
more than 20% worse than the best case (Kosarak-100). Hence, even with very
dense data distributions, block partitioning overhead should never be to high.

7 Conclusions and Future Work

We have proposed a new index structure suitable for itemset extraction from data
streams, which are modeled as periodically incoming data blocks. An algorithm
to efficiently extract itemsets by exploiting the proposed index structure has
been presented. Time, item or support contraints may be enforced to drive the
extraction process. Preliminary experiments show that the overhead due to block
partitioning is never very high and decreases for decreasing support thresholds.

As future work, we plan to address adaptive selection of different compact
structures. Since compact structures are linked by means of an IF-Btree, each
data block may be represented by means of a different compact structure suit-
able for its data distribution. In our current implementation we have exploited
the I-Tree structure to represent all blocks. We plan to include data structures
appropriate for sparser data distributions, such as [12].

Currently only single item constraints are implemented. We also plan to
extend our approach to deal with boolean expressions of item constraints.

References

3. E. Baralis, T. Cerquitelli, and S. Chiusano. Index Support for Frequent Itemset
Mining in a Relational DBMS. In ICDE, 2005.
4. D. W.-L. Cheung, J. Han, V. Ng, and C. Y. Wong. Maintenance of discovered
association rules in large databases: An incremental updating technique. In ICDE,
5. W. Cheung and O. R. Zaiane. Incremental mining of frequent patterns without
8. V. Ganti, J. E. Gehrke, and R. Ramakrishnan. Mining data streams under block
evolution. SIGKDD Explorations, 3(Issue 2), January 2002.
In FIMI, November 2003.
11. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation.
In ACM SIGMOD, 2000.
In FIMI, 2003.