MFCD: An Optimized Technique for Mining Frequent Closed Item Sets

Sanjay Bohara[1] Abhishek Raghuwanshi[2]
[1]P.G. Student [2]Assistant Professor
[1]Department of Information Technology [2]Department of Computer Science & Engineering
[1,2]MIT Ujjain (M.P.), India

Abstract— The amount of data being collected is increasing rapidly. The main reason is the use of computerized applications. Because of that reason the valuable information is hidden in large amount of data. It is attracting researchers of multiple disciplines to study effective approaches to derive useful knowledge from vast data .Frequent closed item set mining has been a heart favorite theme for data mining researchers for over a decade. A large amount of literature has been dedicated to this research and tremendous progress has been made, ranging from efficient and scalable algorithms for frequent closed item set mining in transaction databases to numerous research frontiers, such as sequential pattern mining, structured pattern mining, correlation mining, associative classification, and frequent pattern-based clustering, as well as their broad applications. In this thesis, we propose a new technique for more efficient frequent closed item set mining. Our proposed algorithm will reduce the complexity of frequent closed item set mining. We present efficient techniques to implement the new approach.

Key words: Frequent Closed Item, Correlation Mining, Pattern Based Clustering, Data Cleaning

I. INTRODUCTION

Recent developments in computing and automation technologies have resulted in computerizing business and scientific applications in diverse areas. Turing the huge amounts of accumulated data into knowledge is attracting researchers in various domains including databases, machine learning, statistics, and so on. From the perspectives of database researchers, the emphasis is on discovering useful patterns hidden within the large data sets. Hence, a central issue for knowledge discovery in databases, also the focus of this thesis, is to develop efficient and scalable mining algorithms as integrated tools for database management systems.

Data mining, which is also referred to as knowledge discovery *in* databases, has been recognized as the process of extracting non-trivial, implicit, previously unknown, and potentially useful information from data in databases. The database used in the mining process generally contains large amounts of data collected by computerized applications. For example, bar-code readers in retail stores, digital sensors in scientific experiments, and other automation tools in engineering often generate tremendous data into databases in a very fast speed. Not to mention the natively computing-centric environments like Web access logs in Internet applications. These databases thus serve as rich and reliable sources for knowledge generation and verification. Meanwhile, the large databases present challenges for effective approaches for knowledge discovery.

The discovered knowledge can be used in many ways in corresponding applications. For example, identifying the frequently appeared sets of items in a retail database can be used to improve the decision making of merchandise placement or sales promotion. Discovering patterns of customer browsing and purchasing (from either customer records or Web traversals) may assist the modeling of user behaviors for customer retention or personalized services. Given the desired databases, whether relational, transactional, spatial, temporal, or multimedia ones, we may obtain useful information after the knowledge discovery process if appropriate mining techniques are used A typical process of knowledge discovery in databases is illustrated in Fig. 1-1.

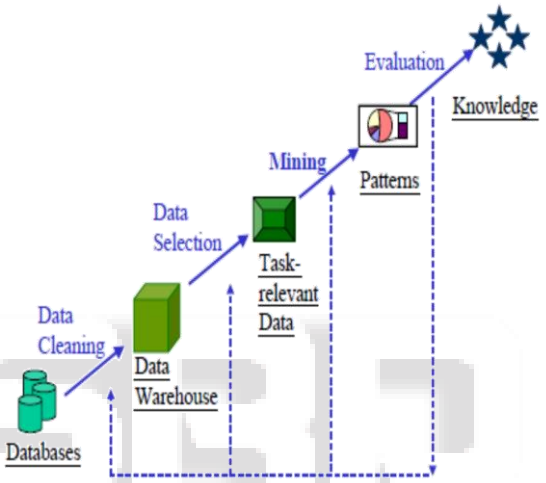

Fig. 1: The process of knowledge discovery in databases

Having the databases, relevant prior knowledge, and the goals of the application domain, the target data set is created by selecting the data required. The data cleaning in Fig. 1-1 may removes those 'dirty' data, e.g. data with incomplete fields, missing or wrong values, in the preprocessing stage. The 'clean' data is then reduced and/or transformed so that the data is represented by the useful features and actionable dimensions. To find the patterns of interest, the users perform the required mining functions, which include summarization of data characteristics, classification/clustering of data for future prediction, association finding for data correlation, trend and evolution analysis, etc. The discovered patterns are *evaluated* and presented as knowledge. The process may iterate and contain certain loops between any two steps.

Of all the mining functions in the knowledge discovering process, frequent pattern mining is to find out the frequently occurred patterns. The measure of frequent patterns is a user-specified threshold that indicates the minimum occurring frequency of the pattern. We may categorize recent studies in frequent pattern mining into the discovery of association rules and the discovery of sequential patterns. Association discovery finds closely correlated sets so that the presence of some elements in a frequent set will imply the presence of the remaining elements (in the same set). Sequential pattern discovery finds temporal associations so that not only closely

Correlated sets but also their relationships in time are uncovered. Data mining represents the integration of several fields, including machine learning, database systems, data visualization, statistics and information theory.

Data mining can be defined as a non-trivial process of identifying

- Valid
- Novel
- potentially useful
- ultimately understandable

Patterns in data. It employs techniques from

- machine learning
- statistics
- databases

Knowledge discovery in databases is a complex process, which covers many interrelated steps. Key steps in the knowledge discovery process are:

- Data Cleaning: remove noise and inconsistent data.
- Data Integration: combine multiple data sources.
- Data Selection: select the parts of the data that are relevant for the problem.
- Data Transformation: transform the data into a suitable format.
- Data Mining: apply data mining algorithms and techniques.
- Pattern Evaluation: evaluate whether they found patterns meet the requirements
- Knowledge Presentation: present the mined knowledge to the user (e.g., Visualization).

Data mining is the process of extracting hidden patterns from data. As more data is gathered, with the amount of data doubling every three years, data mining is becoming an increasingly important tool to transform this data into knowledge. It is commonly used in a wide range of applications, such as marketing, fraud detection and scientific discovery. Data mining can be applied to data sets of any size, and while it can be used to uncover hidden patterns, it cannot uncover patterns which are not already present in the data set.

Data mining extracts novel and useful knowledge from data and has become an effective analysis and decision means in corporation.

Knowledge Discovery in Databases (KDD) is an automated extraction of novel, understandable and potentially useful patterns implicitly stored in large databases, data warehouse and other massive information repositories. KDD is a multi-disciplinary field drawing work from areas including database technology, artificial intelligence, machine learning, neural networks, statistics, pattern recognition, information retrieval, high performance computing and data visualization.

Data mining is an essential step in the process of knowledge discovery in databases, in which intelligent methods are applied in order to extract patterns. Other steps in knowledge discovery process include pre-mining tasks such as data cleaning (removing noise and inconsistent data) and data integration (bringing data from multiple sources to a single location and into a common format), as well as post mining tasks such as pattern evaluation (identifying the truly interesting patterns representing knowledge) and knowledge presentation (presenting the discovered rules using visualization and knowledge representation techniques).

II. Review of Literature

In this section we review some significant closed frequent item sets mining algorithms, and at the same time we deeply analyze the major issues of this pattern extraction problem. These algorithms are very similar to FIM ones, from which they borrow their computational skeleton. Indeed, we describe both level-wise and depth-first algorithms.

A. A-Close [7]:

The first proposed algorithm for mining closed itemsets was A-Close. This was designed on the solid base of Apriori. The idea is to first generate a small set of frequent itemsets, called generators, and then, by calculating their closures, to derive the complete collection of closed itemsets. The resulting algorithm is obviously much faster of Apriori since the number of generators is significantly smaller than the number of frequent itemsets. The authors chose as generators the set of key-patterns since they have an Apriori-like property that helps the mining:

1) Property (Key-Patterns Downward Closure):

An itemset X of cardinality k is a key-pattern if and only if none of its subsets of cardinality $k - 1$ has the same support of X. A slightly modified Apriori extracts all the key-patterns during the first step of the algorithm, and their closures are computed in a second step. The correctness of the algorithm is given by the fact that each closure based equivalence class has at least one key pattern. Unfortunately there is a great drawback of using key-patterns. It is easy to see these are the minimal item sets of closure based equivalence classes. Indeed, each equivalence class has only one closed itemset but it may have several key-patterns. Therefore, A-Close, during its second step, may extract the same closed itemset more than once. This introduces a significant overhead. As for FIM algorithms, also in this case a depth-first approach performs generally better than a level-wise algorithm. This is what the same authors of Eclat and FP-Growth showed with their CFIM algorithms Charm [9] and Closet [8].

These depth-first algorithms use again the idea of finding a set of generators and then calculating their closure. Differently from A-Close, the closure is computed in a sort of in-line fashion. We call this technique closure climbing. Rather than computing closure after the whole collection of generators has been extracted, they compute the closure of a generator as soon as this is found, and then they use the resulting closed itemset to produce new generators. The visit of the search space is indeed very similar to FIM algorithm, but with interleaved closure computations. Given a closed itemsets X, new supersets of length $|X| + 1$ are built and used as generators. Their closure is computed and then used to reiterate the process.

The closure computation involves jumps in the lattice of frequent itemsets, which may or may not violate the underlying lexicographical order. Hence, similarly to the A-Close approach based on key patterns, it is possible to generate the same closed itemset multiple times. As soon as a closed itemset is discovered, it is stored in an incremental data structure. This historical collection is used to detect duplicates. Before calculating the closure of a new generator

X, thus saving some computation, we can check whether there exist a closed itemset Y that includes X and that has the same support of X. In this case, it holds that c(X) = Y and therefore the itemset X would lead to a duplicate. Another important aspect of this kind of algorithms must be taken into account to guarantee their correctness. The great advantage of using dataset projections to speed up the support computation has a negative side-effect when calculating closures. Differently from the level-wise A-Close algorithm, where multiple key-patterns may lead to the same (correct) closure at the price of scanning the whole dataset, by using projections, we risk to produce a set of spurious closed itemsets. This problem is crucial for what regards the order by which itemsets are discovered. Suppose that the closure of a generator cannot be computed in its projected database, if its correct closure was not already discovered and inserted in the historical collection, than the generator would not be detected as a duplicate and an incomplete closure, i.e. a spurious itemset, would be produced. A long as a lexicographical order is used to traverse the search space, the correctness of depth-first algorithms can be guaranteed. If itemsets are not mined in such order, than there is no guarantee of correctness.

B. Charm [9], Closet [8]:

Both Charm and Closet inherit the same data structures and computing framework of their big brothers dEclat and FP-Growth respectively. They implement Algorithm 4, but they differ in the way the closed frequent itemsets are stored in order to exploit the Sub-summation Lemma. Charm adopts a hash table, were the hash function is the sum of the transactions ids supporting an itemset. Closet uses a trie-like structure, indexed by a two-level hash. The first level is based on the last item of the itemset to be checked and the second on its support.

C. FP-Close [12]:

FP-Close is inspired to Closet, thus using the same divide et imperia approach and same FP-tree data structure. What makes FP-Close different from other CFIM algorithms is the application of the projecting approach to the historical collection of closed frequent itemsets. Not only a small dataset is associated to each node of the tree, but also a pruned subset of the closed itemsets mined so far is forged and used for duplicate detection. Indeed, this technique is called progressive focusing and it was introduced by [10] for mining maximal frequent itemsets. Together with other optimizations, this truly provides dramatic speed-up, making FP-Close order of magnitudes faster than Charm and Closet, and also making it worth to be celebrated as the fastest algorithm at the FIMI workshop 2003 [11].

Chi et al. [13] propose an algorithm called Moment for mining frequent closed itemsets over data streams. It uses a CET Tree (Closed Enumerate Tree) to maintain the main information of itemsets. Each node in CET Tree represents an itemset with different node type. Some nodes in CET Tree are not closed so that there are still some redundant nodes in CET Tree. Moment must maintain huge CET nodes for a frequent closed itemset. Chi et al. indicated that the ratio of CET nodes for a closed itemsets is about 20:1. If there are a large number of frequent closed itemsets, it will consume a lot of memory space. When a new transaction arrives, the node is inserted and updated according to its node type. The exploration of frequent itemsets and node type checking are time consuming.

CFI-Stream is another algorithm for this problem [14]. Only the closed itemsets are maintained in a lexicographical ordered tree which is called DIU Tree (Direct Update Tree). Each node consists of a closed itemset and its support count. When a new transaction X arrives, CFI-Stream will generate all the subsets of X, and check if each subset Y is closed or not after the transaction arrives. To check whether an itemset Y is closed or not, CFI-Stream may need to search all supersets of Y from DIU Tree. It takes a lot of time to generate all the subsets of a new transaction and search their supersets from DIU Tree.

III. CLASSIFICATION RULES

In this section, we formally define the closed itemsets [15] and describe some properties.

A. Definition 1 (Closure Operator):

Let T be the subsets of all that transactions in D, $T \subseteq D$, and Y be the subsets of all items appear in D, $Y \subseteq I$. The concept of closed itemset is based on the following two functions f and g:

$$f(T) = \{i \in I \mid \forall t \in T, i \in t\} \quad (1)$$

Function f takes a set of transactions T as an input and returns an itemset included in all transactions belonging to T.

$$g(Y) = \{t \in D \mid \forall i \in Y, i \in t\} \quad (2)$$

Function g takes an itemset Y as an input and returns a set of transactions including Y. A function $C = f \circ g$ is composted by f and g, and is called closure operator [15].

B. Definition 2 (Closed Itemset):

An itemset X is called a closed itemset if and only if

$$C(X) = f \circ g(X) = f(g(X)) = X \quad (3)$$

C(X) is called the closure of X. Definition 2 shows that an itemset X is called closed if and only if X equals to its closure C(X). Otherwise, X is non-closed.

For above example, g({B}) = {t2, t3, t4} since these transactions are including {B}.

Let T = {t2, t3, t4}, f(T) = {AB} since {AB} belongs to each transaction in T. {B} is non-closed since C({B}) = {AB}. From above discussions, we have the following definitions and properties.

C. Definition 3 (Frequent Closed Itemset):

An itemset X is called a frequent closed itemset if and only if X = C(X) and its support is no less than min_sup.

1) Property 1:

If $Y = C(X)$, then $SC(X) = SC(Y)$ [15].

2) Property 2:

If $Y = SC(X)$, then Y is the smallest closed itemset containing X, $X \subseteq Y$.

Moreover, $SC(Y) > SC(Z)$, Z is any superset of X, $X \subset Z$ and $Z \neq Y$.

3) Reason:

Since $X \subset Z$, $SC(X) \geq SC(Z)$. If $SC(X) = SC(Z)$, then each transaction containing X is containing Z, the closure of X is Z instead of Y. It is a contradiction with $Y = C(X)$, $Z \neq Y$. Otherwise, if $SC(X) > SC(Z)$, then $SC(Y) > SC(Z)$

since SC(X) =SC(Y) (Property 1).

4) Property 3:

If SC(X) > SC(Y), Y is any superset of X, $Y \supset X$, then X = C(X) [15].

IV. CONCLUSION

We propose an efficient algorithm, called *MFCD*, for maintaining frequent closed itemsets in a data stream. All closed itemsets and their support counts are records in a link list data structure . Each closed itemset has a unique closed itemset identifier, call *cid*. The cids of super closed itemsets for each item are maintained in the Cid List. As a transaction arrives to the database, it incrementally updates closed itemsets and their supports based on previous mining results. Unlike CFI-Stream, MFCD doesn't need to generate subsets of the transaction, and doesn't need to search supersets for each subset. Via the simple intersection of the transaction andcertain closed itemsets once. The updated closed itemsets can obtained withoutmultiple scans of whole search spaces. Hence, our approach has better performance than previous approaches.

REFERENCES

[1] Y. Bastide, R. Taouil, N. Pasquier, G.Stumme and L. Lakhal. Mining frequent patterns with counting inference. SIGKDD Explorations Newsletter, 2(2):66–75, December 2000.

[2] Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, and L. Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. In CL '00: Proceedings of the First International Conference on Computational Logic, pages972–986, July 2000.

[3] Y. Chi, Y. Yang, Y. Xia, and R. R. Muntz. CMTreeMiner: Mining both closed and maximal frequent subtrees. In PAKDD '04: Proceeding of the eighth Pacific Asia Conference on Knowledge Discovery and Data Mining, pages 63–73, May2004.

[4] RWille. Restructuring lattice theory: an approach based on hierarchies of concepts. In I. Rival, editor, Ordered sets, pages 445–470, Dordrecht–Boston, 1982. Reidel.

[5] X. Yan and J. Han. Closegraph: mining closed frequent graph patterns. In KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 286–295, August 2003.

[6] X. Yan, J. Han, and R. Afshar. Clospan: Mining closed sequential patterns in large datasets. In SDM '03: Proceedings of the third SIAM International Conference on Data Mining, pages 166–177, May 2003.

[7] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In ICDT '99: Proceeding of the 7th International Conference on Database Theory, pages 398–416, January 1999. 8J. Pei, J. Han, and R. Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In DMKD '00: ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pages 21–30, May 2000.

[8] M. J. Zaki and C.-J. Hsiao. Charm: An efficient algorithm for closed itemset mining. In SDM '02: Proceedings of the second SIAM International Conference on Data Mining, April 2002.

[9] K. Gouda and M. J. Zaki. Genmax: An efficient algorithm for mining maximal frequent itemsets. Data Mining and Knowledge Discovovery, 11(3):223–242, 2005.

[10] G. Grahne and J. Zhu. Efficiently using prefix-trees in mining frequent itemsets. In FIMI '03: Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, November 2003.

[11] G. Grahne and J. Zhu. Fast algorithms for frequent itemset mining using fptrees. IEEE Transactions on Knowledge and Data Engineering, 17(10):1347–1362, 2005.

[12] Chi, Y., Wang, H., Yu, P.S., Muntz, R.R.: Moment: Maintaining Closed Frequent Itemsetsover a Stream Sliding Window. In: Proceedings of 2004 IEEE International Conference onData Mining, Brighton, pp. 59–66 (2004).

[13] Jiang, N., Gruenwald, L.: CFI-Stream: Mining Closed Frequent Itemsets in Data Streams. In: Proceedings of 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, pp. 592–597 (2006).

[14] Pasquier, N., Bastide, T., Taouil, R., Lakhal, L.: Discovering Frequent Closed Itemsets for Association Rules. In: Proceedings of the 7th International Conference on Database Theory, Jerusalem, Israel, pp. 398–416 (1999).

