A Framework for Mining Top-K Frequent Closed Itemsets using Order Preserving Generators

R V Nataraj
Department of Information Technology
PSG College of Technology
Coimbatore, India
rvn@ieee.org

S Selvan
Department of Computer Science
St. Peter's Engineering College
Chennai, India.
drselvan@ieee.org

ABSTRACT
In this paper, we propose OP-TKC (Order Preserving Top K Closed itemsets) algorithm for mining top-k frequent closed itemsets. Our methodology visits the closed itemsets lattice in breadth first manner and generates all the top-k closed itemsets without generating all the closed itemsets of a given dataset i.e. in the search space, only closed itemsets that belongs to top-k are expanded and all other closed itemsets are pruned off. Our algorithm computes all the top-k closed itemsets with *O(D+ k)* space complexity, where *D* is the dataset. Experiments involving publicly available datasets show that our algorithm takes less memory and running time than TFP algorithm.

Categories and Subject Descriptors
H.2.8 [**Database Management**]: Database Applications – Data Mining

General Terms
Algorithms, Design, Experimentation.

Keywords
Data Mining – Closed Itemsets – Algorithms – Mining Methods.

1. INTRODUCTION
The Frequent itemset mining is fundamental to several data mining tasks including Association Rule Mining [14][17], life science data analysis [3] and social network analysis [7]. The problem is stated as follows. Given a collection of transactions with each transaction consisting of set of items, a frequent itemset is a subset of set of items that occurs in at least user specified percentage (support) of the transactions. Frequent itemset mining is a computationally demanding task and requires more memory, I/O traffic, high computational power and building efficient algorithms for frequent itemset mining has been an active area of research in the field of data mining. Several algorithms have been proposed to mine frequent itemsets including Apriori, F-Apriori [21], FP-growth [16], FP-growth* [5] and Transaction Mapping Algorithm [4]. But frequent itemset mining suffers from several drawbacks. The main drawback is "too many frequent itemsets". For example, if the frequent itemset length is *x,* then 2^x frequent itemsets would be generated. In many applications with long frequent itemsets, enumerating all possible subsets is computationally infeasible. To overcome the "too many itemsets" disadvantage, closed itemset concept was proposed [8]. The set of closed itemsets of the given transactional database is the condensed representation of set of all frequent itemsets without any loss of information i.e. from the given set of closed itemsets it is possible for us to generate all the frequent itemsets. A frequent itemset is said to be closed if it has no superset with the same support. Several algorithms have been proposed in the literature including A-Close, FP-Close [5], AFOPT-Close [25], B-Miner & C-Miner [3] , and DCI-Close [1]. However, for datasets with large number of items and long itemsets, the number of closed itemsets generated is still large in number. For some real datasets, generating all closed itemsets takes huge running time and for a particular dataset, it took 45 days of running time to generate all the closed itemsets [1]. Moreover, it is difficult to come up with appropriate minimum-support threshold since it requires comprehensive domain knowledge, the mining query and task specific data. It is to be noted that closed itemsets are extensively used in bioinformatics [3][27][28], web usage mining and association rules. Since setting a good threshold is a tedious task, the concept of top-*k* frequent closed itemsets was proposed in [2]. Top-*k* frequent closed itemsets are first *k* closed itemsets obtained by sorting all the closed itemsets of the given dataset in support descending order. If the support is same then the cardinality of the dataset is given precedence. The brute force approach for mining top *k* closed itemsets is to generate all closed itemsets, sort all the closed itemsets in support descending order and output first k itemsets in the sorted list. However, this is a computationally expensive task and also requires huge memory to store all the closed itemsets. For many datasets, storing all closed itemsets in memory require more than the available main memory. Hence, an efficient algorithm is required to mine all top-*k* closed itemsets. To our knowledge, TFP [2] is the only algorithm reported in the literature for mining top-*k* closed itemsets. TFP algorithm uses FP-tree for mining top-*k* closed itemsets. It should be noted that fp-tree based algorithms are optimal only when the dataset is

Compute 2009, Jan 9,10, Bangalore, Karnataka, India.

dense with short itemsets. For sparse datasets, fp-tree occupies large amount of main memory since prefix sharing is very less in sparse datasets. Moreover, for long itemsets, fp-tree based algorithm quickly runs out of main memory [1] which we have validated experimentally by conducting experiments on synthetic datasets generated using IBM dataset generator. The reason is, fp-tree based algorithms store all the frequent itemsets of single prefix path fp-tree in main memory for closure checking. When we run FP-Close algorithm for a support value of 1 for a Boolean dataset of size 25 x 25 with diagonal elements set to zero, the execution terminated outputting "all the buckets are used up". It is because the entire fp-tree of the dataset is a single prefix path tree and there are 2^{25}-1 closed itemsets present in the tree.

In this paper, we propose a novel, fast and memory efficient algorithm based on order preserving closed itemset generation [1] to mine all the top-*k* closed itemsets without generating all of the closed itemsets of given dataset. Our approach processes only top-*k* closed itemsets and prunes away all other closed itemsets. Moreover, the space complexity of our approach is *O(D+ k)* where *D* is the dataset. We have conducted huge number of experiments and our algorithm takes less running time and less memory than TFP algorithm. Moreover, our approach uses same amount of main memory and the main memory usage does not depend on itemset length and nature (sparse or dense) of the dataset. Apart from running time of executions, we have also computed the peak memory usage of executions. To our knowledge, memory usage of execution is not reported in many of the previously reported work on closed itemset mining.

The rest of the paper is organized as follows. Section 2 presents the preliminaries associated with closed itemsets, order preserving closed itemset generation algorithm and problem definition. In Section 3 we present our algorithm, its limitations and theoretical correctness. In section 4, we give our experimental results, while section 5 concludes the paper.

2. PRELIMINARIES

We give the basic definition of closed itemset in this section. In the context of association rule mining, a database *D* is a triple, *D=(T,I,R)*, where *T* is the finite set of transactions and I is the finite set of items. $R \subseteq T \times I$ is a binary relation between transactions and items. Each pair $(t, i) \in R$ denotes the fact that the transaction $t \in T$ is related to the item $i \in I$ i.e. the item *i* is present in the transaction *t*. An itemset $P \subseteq I$ is frequent if support count of *P* exceeds user defined minimum support. An itemset $P \subseteq I$ is closed if $h(P)=P$, where $h(P)=f(g(P))$, $g(P)=\{t \in T \mid \forall\, i \in P\ (t,i) \in R\}$ and $f(T)=\{\, i \in I \mid \forall\, t \in T,\ (t,i) \in R\,\}$. In other words, the function *g* takes a set of items as input and outputs the set of transactions that are related to all items $i \in P$. The function *f* takes a set of transactions as input and outputs the set of items that is common to all the transactions. Closed itemset is a condensed representation of frequent itemsets without any loss of information. For more details on closed itemsets and its properties, readers may refer [7].

Order preserving closed itemset generation algorithm [1] is based on the following principle: "every closed itemset is a superset of another closed itemset". The algorithm visits the search space (item space) in the depth first manner and it outputs the closed itemsets. The procedure attempts to build valid generators, which are subsets of another closed itemset and all the valid generators lead to a closed itemset. The order preserving algorithm takes three parameters as input: *closed_set*, which is initially empty, *pre_set*, which is also initially empty and *post_set*, which contains all the items. *post_set* contains the set of items to be processed whereas *pre_set* contains the processed items from *post_set* that lead to valid generators. *pre_set* is updated when the recursive call returns and it does not change when the recursive call deepens. The algorithm builds all the possible generators by adding items from the *post_set* to *closed_set*. If the supporting transactions of the generator is subset to any one of the supporting transactions of the element $i \in pre\_set$, then the generator is invalid i.e. the closed itemset of the current generator has already been generated while processing item i. The algorithm finds all the valid generators and then computes the closed itemsets.

3 MINING TOP-K CLOSED ITEMSETS

Order preserving closed itemset generation algorithm reported in [1] explores the closed itemset lattice in depth first manner to save memory. There are several problems associated with depth first exploration to mine top-k closed itemsets. Firstly, we cannot output top-*k* unless we explore the complete search space. Secondly, all the closed itemsets visited during the exploration of search space should be stored in main memory. Hence, generation of top-*k* closed itemsets using depth first exploration is both CPU and memory intensive. To efficiently mine the top-*k* closed itemsets, we explore the search space in breadth first manner. However, breadth first exploration requires huge memory and to optimize the memory usage, we use heuristics obtained from problem definition knowledge and the properties of order preserving algorithm. The problem definition knowledge is, we want only first *k* closed itemsets of high support. It should be noted that the order preserving algorithm explores the closed itemset lattice in depth first manner starting from closed itemset with high support to closed itemsets with low support i.e. all of the closed itemsets of level 1 will have more support than the closed itemsets of other levels. As the depth increases, support of the closed itemset decreases and length of the closed itemset increases. Since closed itemsets are explored in support descending order, we only allocate *k* memory locations to store *k* closed itemsets explored in breadth wise manner. When we visit closed itemset, *C*, with support less than k^{th} – closed itemset, *C* is discarded and the entire subtree of *C* is pruned as not belonging to top-*k* closed itemsets. However, if we have only *p* closed itemsets, where $p < k$, then the closed itemset *C* is added at the end if support is less than p^{th} closed itemset or inserted in appropriate position if the support is greater than p^{th} closed itemset.

Proposition 1: At any time, all the closed itemsets in top-k list is maintained in support descending order.

Lemma 1: Order preserving algorithm generates all and only closed itemsets.

Proof: Refer [1].

Lemma 2: Let C be the closed itemset in the search space. Then $C' \supseteq C$ where $C' \in P(C)$ and P(C) is the set of closed itemsets obtained by processing C.

Proof: The proof is straightforward.

Lemma 3: Let C be the closed itemset in the search space and P(C) be the set of closed itemsets obtained by processing C. Then support(C') < support(C) where C' ∈ P(C).

Proof: The proof is straightforward.

Lemma 4: Let C be the m^{th} closed itemset in top-k list where $m \leq k$. Then all the closed item sets C' where $C' \supseteq C$ will occupy only n^{th} position in the top-k list where $n \geq m$ and $n \leq k$.

Proof: According to lemma 3, *C'* will have less support than *C*. According to proposition 1, the position of *C'* in top-*k* list is less than the position of *C*.

Lemma 5: Closed Itemset Pruning: let C be the newly generated closed itemset. If $\neg\exists\, C' \in$ top-k list such that support(C) > support(C') then C does not belong to top-k list and can be pruned.

Proof: The proof is straightforward.

3.1 OP-TKC Algorithm

Let $\mathcal{D}$ be the dataset with $\mathcal{I}$ items and $\mathcal{T}$ transactions. OP-TKC algorithm maintains a top-*k* list with each element of top-*k* list consisting of *pre_set, post_set closed_set* and *support_set.* The first element of list is initialized with { Φ }, { $\mathcal{I}$ }, { Φ }, { $\mathcal{T}$ } for *pre_set, post_set, closed_set* and *support_set* respectively. The algorithm processes the first element of top-*k* list in breadth first manner and appends the resulting closed itemsets in support descending order along with its respective *pre_set, post_set, closed_set* and *support_set.* Processing of a node is done by removing an item, *p*, from *post_set* and adding it to the *closed_set. pre_set* has the already processed items and is used for avoiding duplicate generation. Then the closure of *closed_set* ∪ *p* is computed and the resulting closure along with its *pre_set*, *post_set* and *support_set* are inserted in the appropriate position in top-*k* list. Once the first element is fully processed i.e. when the *post_set* becomes null, the algorithm processes the second element of list and inserts the resulting closed itemsets in appropriate position in top-*k* list. Till the top-*k* is not full, all the generated closed itemsets are inserted along with its supporting information. Once the top-*k* list is full, all the closed itemsets along with its *pre_set, post_set* and *support_set* are discarded as not belonging to top-*k* list if the support is less than k^{th} closed itemset of top-*k* list. Otherwise, the closed itemset and its supporting information are inserted in appropriate position. After inserting, the k^{th} closed itemset now occupies $k+1^{th}$ position and all the elements starting from $k+1^{th}$ position are deleted from the list if $|support\_set(k)| > support(k+1)$. If the support is same, then the length is checked i.e. we check whether $|closed\_set(k)|$ is equal to $|closed\_set(k+1)|$. If both are same, then $k+1^{th}$ element is also retained in the list. The same procedure is continued until all the elements in the linked list have been explored and the top-*k* closed itemsets are generated.

3.2 OP-TKC Pseudo code

INPUT: transactional dataset, k value
OUTPUT: All top-k closed itemsets satisfying size constraints.

```
1.  sort all the items with respect to its support, map the sorted
    items to continuous integers and store it in set F1
2.  construct vertical bit-vector in the mapped space
3.  initialize top-k list
4.  call OP-TKC ( k )

5.  OP-TKC ( k )
6.  {
7.      for ( i=0;  i<k; i++ )
8.          closed_set = list[i].closed_set
9.          post_set = list[i].post_set
10.         pre_set = list[i].pre_set
11.         support_set=list[i].support_set
12.         while (post_set != null )
13.             i'=min(post_set)
14.             post_set = post_set \ i'
15.             post_set' = post_set
16.             closed_set' = closed_set ∪ i'
17.             support_set' = support_set ∩ g(i')
18.             if (( ∀ j ∈ pre_set, support_set' ⊄ g(j)) &&
                    |support_set'| ≥ minsupp(k^th top-k element )
19.                 ∀ k ∈ post_set
20.                     if support_set' ⊆ g(k)
21.                         closed_set' = closed_set' ∪ k
22.                         post_set' = post_set' \k
23.                     endif
24.                 insert closed_set', pre_set , pre_set'
                    and support_set' in list in correct position
25.                 pre_set = pre_set ∪ i'
26.             endif
27.         endwhile
28.     endfor
29. }
```

3.3 Pseudo code Description

The pseudo code uses the set variables *pre_set, post_set, closed_set* and *support_set* to store the current element of top-*k* list that is being processed (line no 8 – 11). The set variables *pre_set', post_set', closed_set',* and *support_set'* are used to store the current closed itemset and its supporting information to be added to the list. While processing the current element of list, *pre_set* monotonically increases and *post_set* monotonically decreases. The *closed_set* and *supporting set* remains same. The processing of the current element of top-*k* list gets completed when its associated *post_set* becomes null i.e. no other closed itemsets could be generated by extending the current *closed_set* with elements from the *post_set*. Whenever a generator is created for new closed itemset (line no 16), the generator is checked for duplication (line no 18). If the support set of generator is a subset to any one of the supporting set of an element of *pre_set,* then the current generator is discarded as duplicate. Otherwise, the closure of the new generator is computed (line no 19-23) and the resulting closed itemset along with its necessary parameters are inserted in the top-*k* list in appropriate position (line no 24).

3.4 Other Optimizations

We have included several optimization techniques to reduce the computation time of top-k closed itemset generation.

Lemma 6: Let s be the support of k^{th} element of top-k list. Then we only need to generate closed item of support ≥ s

Proof: Let *C* be the newly generated closed itemset. *C* is inserted into top-*k* list iff *support(C)* is greater than k^{th} element of top-*k* list (lemma 5). Thus all closed itemsets of support less than *s* need not be generated.

Lemma 7: Raising minimum support: once the top-k list is full we can safely raise the minimum support to support(k^{th} element) of top-k list.

Proof: The proof follows from lemma 6.

Based on lemma 6, we dynamically reduce the dataset by removing all items that does not satisfy the minimum support. Reducing the dataset is a costly operation and we perform dataset reduction only when the difference of current support and k^{th} element support of top-*k* list is very large. This technique is highly efficient for sparse datasets with very large number of items. Since the minimum support value is dynamically updated based on the k^{th} element of top-*k* list, we exploit the following lemma to further reduce the computation time associated with closed itemset generation.

Lemma 8: Let C be the closed itemset that is being processed. Let post_set(C) be the post_set associated with C. If $\exists\, i \in C$ and $j \in$ post_set(C) such that IG[i][j] < current support value, then j can be removed from the post_set.

Proof: The proof easy and thus omitted.

To efficiently implement Lemma 8, we construct special array called interaction array(IG). Interaction array is a 2D array, where the 1st and 2nd dimension are sets of items and each cell of row *r* and column *c* specifies the total transaction in which both *r* and *c* are present. At every updation of support value, we just scan this array to remove unpromising elements from *post_set*.

3.5 Correctness

Theorem 1 shows that OP-TKC can correctly generate all and only all top-*k* closed itemsets.

Theorem 1: Let K be the set of top-k closed itemsets of a transactional dataset. Let K' be the set of top-*k* closed itemsets derived from applying OP-TKC on the dataset. Then $K' = K$. In other words, OP-TKC can correctly generate all and only all top-*k* closed itemsets.

Proof: First, we prove that $K \subseteq K'$ by contradiction. Assume $\exists\, C \in K$ but $C \notin K'$ where C is a closed itemset. Let δ be the set of closed itemsets that does not belong to top-*k* list and removed by the pruning strategy of lemma 5. Then $C \in (K' \cup \delta)$. As shown in lemma 5, closed itemset pruning strategy only removes closed itemsets that does not belong to top-*k* list. So, $C \notin \delta$ and $C \in K.'$ Hence, our assumption that $C \notin K'$ is wrong and we conclude $K \subseteq K'$. Next, we prove $K' \subseteq K$ by contradiction. Assume $\exists\, C \in K'$ but $C \notin K$. Then C is either not closed or does not belong to top-*k* closed itemsets. If C is not closed, then it would not have been generated (lemma 1). Also lemma 5 prunes only closed itemsets that do not belong to top-*k* list. Hence our assumption that $C \in K'$ and $C \notin K$ is wrong. Thus OP-TKC correctly generates all and only top-*k* closed itemsets.

3.6 Limitations

There are certain limitations in our approach when we include the length constraint. The limitation is large main memory usage. If we include length constraint, then all the closed itemsets of length < *l*, where *l* is the user specified minimum length, should be stored in the top-*k* list. We can remove a closed itemset of length < *l* only if its associated *post_set* is null. i.e. only after generating all of its immediate supersets which are closed. Hence, we may need to store more than *k* closed itemsets in the top-*k* list and the peak number of closed itemsets stored in the list is determined by the dataset and the given length.

4 EXPERIMENTS AND RESULT ANALYSIS

We have implemented our algorithm using C language and the code was compiled using 32-bit Microsoft Visual C++ compiler. All the experiments are conducted on Pentium 4 machine with 1 GB of main memory loaded with windows XP operating system. The executable file of TFP algorithm was obtained from the respective authors. To obtain the accurate peak memory usage of executions, we have written our own stub code using windows process library API that will fetch the main memory usage statistics whenever a process is terminated. Since we extract the needed information from the windows kernel itself, the load made by this program on the memory and the processor is completely negligible. Moreover, the entire coding is only a few lines of code and just fetches memory usage data from the windows kernel. We have checked the running times while running this piece of code in background and while this software was not running in background. The observed differences are only in microseconds and in most cases we didn't observe any difference. The dataset generator is downloaded from Illimine project website. For TxIyDz, x indicates the average transaction length, y indicates the average pattern length and z indicates the total number of transaction instances. To make the comparison fair, we have set the length parameter of TFP code to 1. We have done huge number of experiment and we present only representative results here. The plots in Fig. 1 and Fig. 2 show the performance comparison in terms of running time and peak memory usage of executions for OP-TKC and TFP. The data given in Table 1 presents the running time of algorithm and peak memory usage for chess dataset. The data given in Table 2 gives the experimental results for T50I45D10K dataset. The peak memory usage data given in our results only include the main memory usage and not the page file usage. We have also collected the peak page file usage data but we have not reported the results here because the page file usage was negligible for most of our experimental executions. In our future work, we are planning to experiment with very large datasets to correctly assess the performance in terms of peak page file usage. It is to be noted that our implementation is still being optimized and we may get better results in our future work.

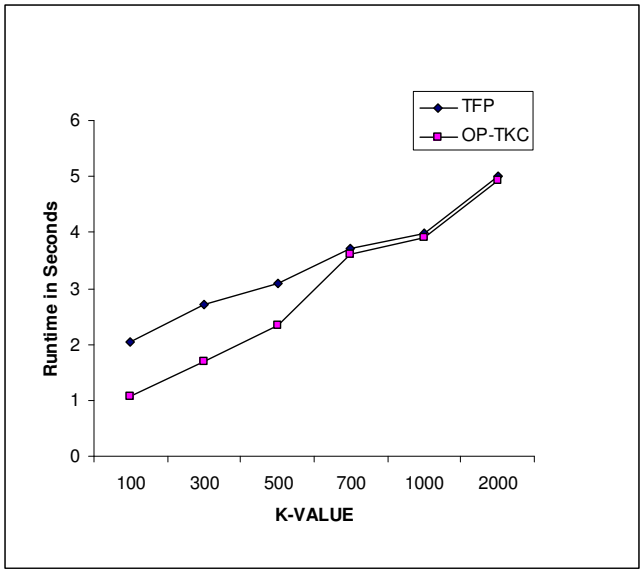

Figure 1: Algorithm runtime Vs K Value for T25I20D50k dataset

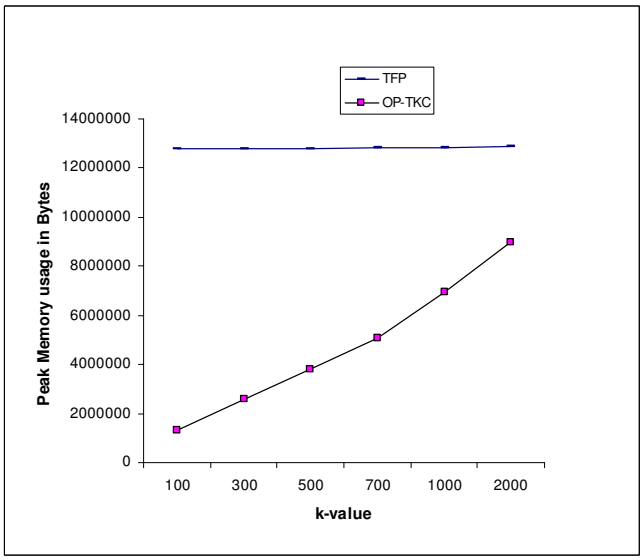

Figure 2: Memory Usage Vs K Value for T500I450D250 dataset

Table 1: Algorithm Running time in Seconds and Peak Memory usage in Bytes for chess dataset.

k-value	TFP		OP-TKC	
	Running time	Peak Memory Usage	Running Time	Peak Memory Usage
100	0.188	9773056	0.093	880640
300	0.203	9805824	0.192	1175552
500	0.256	9838592	0.246	1503232
700	0.289	9871360	0.3	1847296
1000	0.29	9936896	0.392	2338816
2000	0.296	10035200	0.412	3993600

Table 2: Algorithm Running time in Seconds and Peak Memory usage in Bytes for T50I45D10K dataset.

k-value	TFP		OP-TKC	
	Running time	Peak Memory Usage	Running Time	Peak Memory Usage
100	1.062	48803840	0.828	2740224
300	1.822	48804902	1.234	4341760
500	2.625	48840704	2.140	5943296
700	2.766	49045504	2.750	7544832
1000	2.925	49270784	3.025	9977856
2000	3.515	49537024	3.411	18038784

Table 3: Algorithm Running time in Seconds and Peak Memory usage in Bytes for T25I20D10K dataset.

k-value	TFP		OP-TKC	
	Running time	Peak Memory Usage	Running Time	Peak Memory Usage
100	0.794	28065792	0.203	1327104
300	0.988	28143616	0.500	2850480
500	1.066	28241920	0.812	3825664
700	1.162	28307456	1.109	5050368
1000	1.252	28409856	1.352	6942720
2000	1.292	28708864	1.382	13127680

5 CONCLUSION

We have introduced a novel algorithm which explores the closed itemset lattice in breadth first manner to generate all the top-*k* closed itemsets without examining all the closed itemsets. Our algorithm is highly memory efficient since it stores only the input dataset and top-*k* itemsets in memory. Also, our algorithm is time efficient since it expands only nodes in the search space that belongs to top-*k* closed itemsets. We have also discussed the limitations of our approach to mine all top-k closed itemsets with size constraint. We are currently working on to design fast and memory efficient framework for mining top-*k* closed itemsets with size constraint.

6 ACKNOWLEDGEMENTS

We thank Jianyong Wang for providing us the executables of TFP algorithm and responding to our queries. We would like to thank C. Luchesse and S. Orlando for providing us the executables of DCI-Close Algorithm and responding to our numerous queries.

7 REFERENCES

[1] C. Lucchese,S. Orlando and R. Perego, "Fast and Memory Efficient Mining of Frequent Closed Itemsets", IEEE Transactions on Knowledge and Data Engineering, VOL 18, No 1, pages 21-36, January 2006.

[2] J. Wang, J. Han, Y.Lu,P. Tzvetkov, "TFP: An Efficient Algorithm for Mining Top-K Frequent Closed Itemsets, " IEEE Trans. on Knowledge and Data Engineering, Vol 17, No 5, pp. 652-664, May 2005

[3] Liping Ji, Kian-Lee Tan,K H. Tung, "Compressed Hierarchical Mining of Frequent Closed Patterns from Dense

Data Sets," IEEE Trans. on Knowledge and Data Engineering, Vol 19, No.9, Sept 2007.

[4] Mingjun Song, Sanguthevar Rajasekaran, "A Transaction Mapping Algorithm for Frequent Itemsets Mining", IEEE Transactions on Knowledge and Data Engineering,, VOL 18, No 4, pages 472-481, April 2006.

[5] G. Grahne, J. Zhu, "Fast Algorithms for Frequent Itemset Mining Using FP-Trees", IEEE Transactions on Knowledge and Data Engineering, Vol 17, No 10, pages 1347-1362, October 2005.

[6] D.Burdick, M.Calimlim, J.Flannick, J.Gehrke, Y.Yiu, "MAFIA: A Maximal Frequent Itemset Algorithm", IEEE Transactions on Knowledge and Data Engineering, VOL 17, No 11,Pages 1490 - 1504, November 2005.

[7] Jinyan Li, Guimei Liu, Haiquan Li, Limsoon Wong, "Maximal Biclique Subgraphs and Closed Pattern Pairs of the Adjacency Matrix: A One-to-One Correspondence and Mining Algorithms," *IEEE Trans. Knowledge and Data Engineering*, vol. 19, No. 12, pp. 1625-1637, Dec. 2007.

[8] N. Pasquier, Y. Bastide, R. Taouil, and L.Lakhal, "Discovering Frequent Closed Itemsets for Association Rules", Proc. 7th Int. Conf. Database Theory (ICDT'99), pages 398-416, January 1999.

[9] Dao-l Lin and Zvi M. Kedem, "PINCER-SEARCH: An Efficient Algorithm for Discovering the Maximum Frequent Set", IEEE Trans. on Knowledge and data Engineering, VOL 14, No. 3, June 2002.

[10] K. Makino and T. Uno, "New algorithms for enumerating all maximal cliques," in Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT 2004), Springer-Verlag, 2004, pp. 260–272.

[11] Guizhen Yang, "The complexity of Mining Maximal Frequent Itemsets and Maximal Frequent Patterns", KDD'04, Seattle, Washington, August 2004.

[12] R. Agrawal, T. Imielinski, and A. Swami. "Mining association rules between sets of items in large databases", In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pages 207-216, Washington, DC, May 1993.

[13] T. Uno, M. Kiyomi, and H. Arimura, "LCM ver.2: Efficient mining algorithms for Frequent/closed/maximal itemsets," In Proc. IEEE ICDM'04 Workshop FIMI'04, 2004.

[14] K. Gouda, M.J.Zaki, "GenMax: An Efficient Algorithm for Mining Maximal Frequent Itemsets", Journal of Data Mining and Knowledge Discovery, pages 1-20, 2005

[15] C. Lucchese, S. Orlando, P. Palmerini, R. Perego, and F. Silvestri, "KDCI: A Multi-Strategy Algorithm for Mining Frequent Sets," Proc. IEEE ICDM FIMI'03 Workshop, Dec 2003

[16] Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao, "Mining Frequent Pattern without candidate Generation : A Frequent Pattern Approach" Journal of Data Mining and Knowledge Discovery , Springer, pages 53-87, 2004.

[17] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules", In Proceeding of Int. Conf. Very Large Data Bases, pages 487-499, Santiago, Chile, September. 1994.

[18] A. Savasere, E. Omiecinski, and S. Navath, "An efficient algorithm for mining association rules in large databases", In Proc. of Intl. Conf. on Very Large Databases (VLDB), 1995.

[19] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation", In Proceeding of Special Interest Group on Management of Data , pages 1-12, Dallas, TX, May 2000.

[20] R.J. Bayardo, "Efficiently Mining Long Patterns from Databases", Proc. ACM-SIGMOD Int`l Conf. Management of Data, pages 85-93, 1998.

[21] Ferenc Bodon, "A Fast APRIORI Implementation," IEEE ICDM FIMI'03,USA, 2003.

[22] M.J.Zaki, S. Parthasarathy, M. Ogihara and W.Li, "New Algorithms for fast Discovery of Association Rules", Proc. 3rd Intl. Conf. Knowledge Discovery and Data Mining, pages: 283-286, 1997

[23] I.Rigoutsos and A. Floratos, "Combinatorial Pattern Discovery in Biological Sequences: The Teiresias Algorithm, Bioinformatics, Vol 14, pages 55-67,1998.

[24] J. Wang, J.Han and J.pei, "CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets", Proc. 2003 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, pages 236-245, August 2003.

[25] G.Liu, "Supporting Efficient and Scalable Frequent Pattern Mining," PhD dissertation, Dept. of Computer Science., Hong Kong University., May 2005.

[26] C. Lucchese, S. Orlando, P. Palmerini, R. Perego and F. Silvestri, "KDCI: A multistrategy Algorithm for Mining Frequent Sets," *In Proc. IEEE ICDM'03 Workshop FIMI'03*, 2003.

[27] J. Besson, C. Robardet, J.F. Boulicaut and S. Rome,"Constraint Based Concept Mining and its Application to Microarray Data Analysis", Journal of Intelligent Data Analysis, pp. 59-82, 2005.

[28] F. Pan, G. Cong, A.K.H. Tung, J. Yang, M. J. Zaki, "CARPENTER: Finding Closed Patterns in long Biological Datasets", SIGKDD-03, 2003.