Survey on the Techniques of FP-Growth Tree for Efficient Frequent Item-set Mining

Rana Krupali
Parul University
Limda, Baroda
India

Dweepna Garg
Parul University
Limda, Baroda
India

ABSTRACT
Analysis has been carried out in terms of FP-Growth Tree techniques to determine which technique can be used efficiently in order to achieve higher scalability and performance. Construction and development of classifier that works with more accuracy and performs efficiently for large database is one of the key tasks of data mining techniques. Secondly training dataset repeatedly produces massive amount of rules. It's very tough to store, retrieve, prune, and sort a huge number of rules proficiently before applying to a classifier. In such situation FP is the best choice but problem with this approach is that it generates redundant FP Tree. A Frequent pattern tree (FP-tree) is type of prefix tree that allows the detection of recurrent (frequent) item set exclusive of the candidate item set generation. It is anticipated to recuperate the flaw of existing mining methods. FP – Trees pursues the divide and conquers tactic.

General Terms
FP- tree structure, Apriori algorithm, Association Rule

Keywords
Data Mining, KDD, Association Rule, FP-Growth Tree, FP-Growth Tree Techniques.

1. INTRODUCTION
In Data Mining the task of finding frequent pattern in large databases is very essential and has been studied on huge scale in the past few years. Unfortunately, it is computationally expensive, especially when a huge number of patterns exist.
The FP-Growth Tree Algorithm, proposed by Han, is an efficient as well as scalable method for mining the full set of frequent patterns by pattern fragment growth, using an extended prefix-tree structure for storing compressed and crucial information about frequent patterns named frequent-pattern tree (FP-tree). In that study, Han proved that this method outperforms other popular methods to mine frequent patterns, for e.g. the Apriori Algorithm and the Tree Projection. In other works it was proved that FP-Growth computes with better performance than other methods, including Eclat and Relim. The popularity and efficiency of FP-Growth Algorithm contributes with many studies that propose variations to improve its performance.
The FP-Growth Algorithm can be considered to find an alternative frequent item sets without making use of the candidate generations, hence it improves performance .So far it uses a divide-and-conquer strategy. The core of this method is the usage of a special data structure named frequent-pattern tree (FP-tree), helps in retaining the item-set association information.

In other words, it works as follows: first it compresses the input database by creating an FP-tree instance for representing frequent items. After the initial step, divides the compressed database into a set of conditional Growth which reduces the search costs looking for short patterns recursively and afterwards concatenating them all in the long frequent patterns, providing high selectivity.

In large databases, it's not possible to hold the FP-tree in the main memory. A strategy to solve this problem is to first partition the database into a set of tiny databases (called projected databases), and then an FP-tree from each of these tiny databases can be constructed.

Databases in which each itemset associated with one frequent pattern. So, each such kind of the database is mining is carried out separately.

FP- TREE STRUCTURE
Among from the many algorithms suggested like Apriori algorithms. It is based upon the anti-monotone property. Due to their two main problems i.e. repeated database scan and high computational cost, there is need of compact data structure for mining frequent item sets, which moderates the multi scan problem and improve the candidate item set generation. Tree projection is an efficient algorithm based upon the lexicographic tree in which each node represents a frequent pattern [7] [8] [11].

FP-Growth algorithm [11] [21] is an efficient algorithm for producing the frequent itemsets without generation of candidate item sets. It based upon the divide and conquers strategy. It needs a 2 database scan for finding all frequent item sets.[41] This approach compresses the database of frequent itemsets into frequent pattern tree recursively in the same order of magnitude as the numbers of frequent patterns, then in next step divide the compressed database into set of conditional databases.

The negative association rule is the complement of the general association rules.[14] Mining negative association rules will be related to many non-frequent item set, in order to effectively and produce positive and negative association rules, mining model of Xindong Wu, are a kind of PR, discovering both positive and negative association rules; Xiangjun Dong, et al. gives the multilayer minimum support degree model MLMs at the same time, the minimum reliability and correlation coefficient of positive and negative association rules mining.[33] [37]

In order to solve the positive and negative association rules generated a large number of frequent item sets and the performance of the algorithm problem, a mechanism is proposed for mining positive and negative association rules is improved FP Tree, its characteristics: advantages (1) Inheritance of FP Tree, do not need to scan database repeatedly, do not produce candidate item set; (2) the negative items as similar item transaction inserts a tree is constructed ,

not expanding the original database,with compressed data structure to store the transaction database of relevant information, different nodes can be shared prefix path; (3) doesn't generate conditional pattern base, do not need to a large number of conditions pattern tree resources waste memory and time overhead structure.

The frequent-pattern tree is a kind of compressed structure which stores quantitative information about frequent patterns in a database. [15] [17] [27]

Han defines the FP-tree as the tree structure which is defined below:

- One of the roots can be labelled as "null" with a set of item-prefix sub trees as children, and a frequent-item-header table;
- Each node in the item-prefix sub tree consists of three fields:
- Item-name: registers the item to be represented by the node;
- Count: The number of transactions represented by the portion of the path reaching the node;
- Node-link: links to the next node in the FP-tree carrying the same item-name, or null if there exist none.
- Each entry in the frequent-item-header table consists of only two fields:
- Item-name: is same as to the node;
- Head of node-link: a pointer to the first node in the FP-tree carrying the item-name.

2. FP-GROWTH TREE VARIATIONS

2.1 DynFP-Growth Algorithm [12] [13] [31]

The Dyn FP-Growth has mainly focused to improve the FP-Tree algorithm construction based on the issues such as:

- The resultant FP-tree may not be unique for the same "logical" database;
- The process requires minimum two complete scans of the database.

For solving this initial issue Gyorödi C., et al. proposes the usage of a support descending order all together with a lexicographic order, by ensuring way the uniqueness of the resulting FP-tree for different "logically equivalent" databases.

In order to solve the next issue they proposed devising a dynamic FP-tree reordering algorithm, and using this algorithm a "promotion" to a higher order of at least one item is detected in the database.

One of the very important feature of this approach is that it's not at all essential for rebuilding the FP-Tree as the actual database is updated. That's only required to execute the algorithm taking the new transactions and the stored FP-Tree.

Other adaptation proposed, was of the dynamic re-ordering process, which was a modification in the original structures, executed by replacing the single linked list with a doubly linked list for linking the tree nodes to the header and adding a master-table to the same header.

2.2 FP-Bonsai Algorithm [14] [27]

The FP-Bonsai improves the FP-Growth performance by pruning the FP-Tree using the Ex-Ante data-reduction technique. The pruned FP-Tree which was usually reduced FP- Growth Tree was called FP-Bonsai.

2.3 AFOPT Algorithm [13] [31]

Emphasizing the FP-Growth algorithm performance based Liu proposed the AFOPT algorithm. AFOPT algorithm mainly aims to improve the FP-Growth performance in four perspectives:

1) Item Search Order: As the search space is divided, all items in the database are sorted in specific order. The number of the conditional databases constructed may vary too much using various items search orders.

2) Conditional Database Representation: Traversal and construction cost of a conditional database majorly is dependent on its representation.

3) Conditional Database Construction Strategy: constructing every conditional database physically may be quite expensive which may adversely affect the mining cost of each individual conditional database.

4) Tree Traversal Strategy: The traversal cost of a tree can be minimal using top-down traversal strategy.

2.4 NONORDFP Algorithm [27] [31]

The Nonord fp algorithm was motivated by the running time and the memory space required for the FP-Growth algorithm. The theoretical difference is the main data structure (FP-Tree), which was even more compact and which was not at all required to re-build it for each conditional step. A compact, memory efficient representation of an FP-tree by using Trie data structure, along with a memory layout which allows faster traversal, faster allocation, and optionally projection was introduced.

2.5 FP-Growth *Algorithm [13] [27]

It was proposed by Grahne et al [and is solely based in the conclusion about the usage of CPU time to compute frequent item sets using FP-Growth. Observation was that 80% of CPU time was used to traverse FP-Trees. Hence, an array-based data structure was used by combining it with the FP-Tree data structure for reducing the traversal time, and implies several optimization techniques.

2.6 PPV, Pre-Post, and FIN Algorithm [12] [13]

These all three algorithms were proposed by Deng, and these were based on three novel data structures popularly known as Node-list, N-list, and Node-set respectively in order to facilitate the mining process of frequent item-sets. They were based on a FP-tree with each node encoding with pre-order traversal and post-order traversal. If Compared Node-lists, N-lists and Node-sets are more efficient. It may result into the efficiency of Pre-Post and FIN is higher than that of PPV.

3. MODIFIED FP-GROWTH TREE ALGORITHM

Many things are to be considered for FP growth approach that may result as the weakness of frequent item sets.

The current perspectives used to mine the incremental database are not much efficient. The computation cost get

high with re-initialization of algorithm for the updated set in the database of data. [11][21][43]

For huge databases a single individual processor may not be capable enough for memory for storing the entire data as well as to process the data in the same way.

Basically FP growth tree needs memory space to store the tree structure.

FP-tree can also be quite expensive to build the tree-structure for the data.[45]

The recursion has four different implementations that suit differently sized FP trees:

Very large FP trees that contain millions of nodes are treated by simultaneous projection: the tree is traversed once and a projection to each item is calculated simultaneously. This phase is applied only at the rst level of recursion; very large trees are expected to arise from sparse databases, like real market basket data; conditional trees projected to a single item are already small in this case. [48]

Sparse aggregate is an aggregation and projection algorithm that does not traverse those part of the tree that will not exist in the next projection. [21][27][31] To achieve this, a linked list is built dynamically that contains the indices to non-zero counters. This is similar to the header lists of FP-trees. This aggregation algorithm is used typically near the top of the recursion, where the tree is large and many zeroes are expected. The exact choice is tunable with parameters.

Single chain: In this case no aggregation and calculation of new counters is needed, so a specialized very simple recursive procedure starts that outputs all subsets of the paths in the tree as a frequent itemset.

The core data structure is a trie. Each node contains

A counter and a pointer to the parent. As the trie is never searched, only traversed from the bottom to the top, child maps are not required. The nodes are stored in an array, node pointers are indices to this array. Nodes that are labelled with the same item occupy a consecutive part of this array, this way we do not need to store the item identiers in the nodes. Furthermore, that do not need the header lists, as processing all nodes of a specied item requires traversing an interval of this array. This also allows faster execution as only contiguous memory reads are executed. We only need one memory cell per frequent item to store the starting points of these intervals (the itemstarts array).

Dense aggregate is the default aggregation algorithm. Each node of the tree is visited exactly once and its conditional counter is added to the counter of the parent.

This is the default aggregation algorithm and it is very fast due to the memory layout of the data structure, described later. [11][25][41]

Single node optimization is used near the last levels of recursion, when there is at most one node for each item left in the tree. (This is a slight generalization of the tree being a

The parent pointers (indices) and the counters are stored in separate arrays (parents and counters rsp.) to t the core algorithm's exibility: if projection is not benecial, then the recursion proceeds with the same structural information (parent pointers) but a new set of counters.

The item intervals of the trie are allocated in the array ascending, in topological order. This way the bottom-up and top-down traversal of the trie is possible with a descending

Rsp ascending iteration through the array of the trie, still only using contiguous memory reads and writes. This order also allows the truncation of the tree to a particular level/item: if the structure is not rebuilt but only a set of conditional counters is calculated for an item, then the recursion can proceed with a smaller sized new counters array and the original parents and item starts array.

Table 1. Comparison Table Between Apriori Algorithm And Fp-Tree Algorithm

PARAMETERS	APRIORI ALGORITHM	FP-GROWTH ALGORITHM
Technique / Methods	Applies Apriori properties, join, pruning properties.	First it constructs conditional frequent pattern tree & conditional pattern base from database.
Memory utilization	As large no. of candidate generation is required, large memory space is occupied.	Comprises of compact structure & no candidate generation require less memory.
Number of scans	Multiple scans required.	Scanning the database only twice.
Time	Execution time is more as candidate has to be generated each time.	Execution time is far less as compared to the Apriori algorithm.

4. RESULTS

4.1 Table representing minimum support and execution time

Minimum support	Time taken to execute(In seconds)FP-GROWTH Trees
2	130
3	124
4	88
5	77

4.2 Graph representing minimum support and execution time

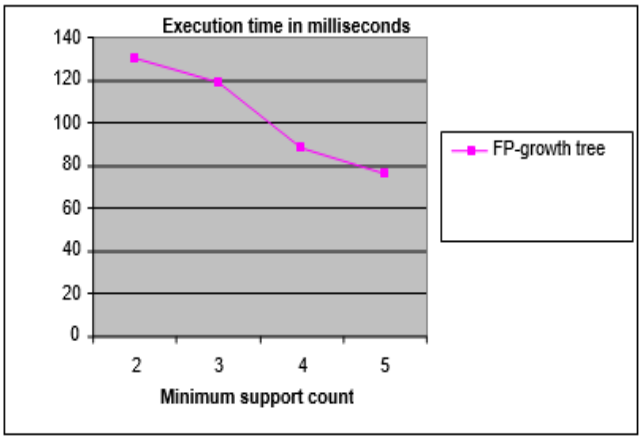

Figure: 1 Represents graph of minimum support and execution time

4.3 Table represents no. of records and execution time

Number of records	Time taken to execute (In millisecond) **FP-GROWTH Tree**
200	87
300	97
400	140
500	201

4.4 Graph representing minimum support and execution time.

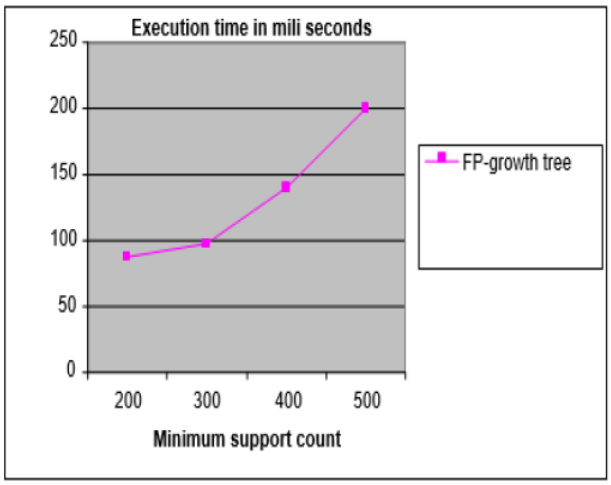

Figure: 2 Represents graph of minimum support and execution time

5. CONCLUSION

FP-Growth is the first successful tree base algorithm for mining the frequent item sets by using its various techniques mentioned its performance can be increased as per the requirements. As in the case of large database its structure fails to fit into main memory hence for this purpose new techniques have been came into existence for reducing data-set and generating tree-structure that may consist of the variations of the classic FP-Tree and result in higher performance.

6. REFERENCES

[1] Abd-Elmegid L A., El-Sharkawi M E., El-Fangary L M., Helmy Y K., Vertical Mining of Frequent Patterns from Uncertain Data, Computer and Information Science. 2010; 3(2); 171–179.

[2] Aggarwal C C., An Introduction to uncertain data algorithm and applications, Advances in Database Systems. 2009; 35; 1–8.

[3] Aggarwal C C., Philip S Yu., A Framework for Clustering Uncertain Data Streams, Data Engineering, IEEE 24th International Conference on ICDE'08. 2008; 150-159.

[4] Aggarwal C C., Yan L., Wang Jianyong, Wang Jing., Frequent pattern mining with uncertain data, In Proc. KDD. 2009; 29-37.

[5] Agarwal, R. C., Aggarwal, C. C., & Prasad, V. V. V. (2000, August). Depth first generation of long patterns. In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 108-118). ACM.

[6] Agrawal R., Srikant R.titFast algorithms for mining association rules In Proc. VLDB 1994, pp.487–499.

[7] Borgelt, C. (2005, August). An Implementation of the FP-growth Algorithm. In Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations (pp. 1-5). ACM.

[8] C.C. Agarwal, "An Introduction to uncertain data algorithm and applications", Advances in Database Systems, 2009, 35; pp 1–8.

[9] Chen H., Ku W S., Wang H., Sun M T., Leveraging Spatio-Temporal Redundancy for RFID Data Cleansing, In SIGMOD. 2010.

[10] Chui C K., Kao B., Hung E., Mining Frequent Itemsets from Uncertain Data, Springer-Verlag Berlin Heidelberg PAKDD'07. 2007; 4426; 47-58.

[11] Deshpande A., Guestrin C., Madden S R., Hellerstein J M., W. Hong., Model-Driven Data Acquisition in Sensor Networks, VLDB; 2004.

[12] Goethals, B. (2003). Survey on frequent pattern mining. Univ. of Helsinki.

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

[14] Grahne, G., & Zhu, J. (2003, November). Efficiently Using Prefix-trees in Mining Frequent Itemsets. In FIMI (Vol. 90).

[15] Han I., Kamber M., Data Mining concepts and Techniques, M. K. Publishers. 2000; 335–389.

[16] Han, J., Pei, J., & Yin, Y. (2000, May). Mining frequent patterns without candidate generation. In ACM SIGMOD Record (Vol. 29, No. 2, pp. 1-12). ACM.

[17] H. Huang, X.W, and R. Relue, "Association Analysis with One Scan of Databases", Proceedings of the IEEE International Conference on Data Mining, 2002.

[18] Huang J., Antova L., Koch C., Olteanu D. MayBMS: A probabilistic database management system, in Proc. ACM SIGMOD'09. 2009; 1071–1074.

[19] International Journal of Intelligent Computing Research (IJICR), Volume 6, Issue 3, September 2015 Copyright © 2015, Infonomics Society 619.

[20] Jagrati Malviya, Anju Singh, "A comparative study of various database techniques for frequent pattern generation", ACSIT Nov 2014.

[21] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen , U. Dayal , and Hsu, M.-C. FreeSpan, "Frequent pattern-projected sequential pattern mining", ACM SIGKDD, 2010.

[22] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation", SIGMOD 2000, pp 1-12.

[23] Jiawei Han, Jian Pei, Runying Mao," Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach", Data Mining and Knowledge Discovery, April 2001, Kluwer Academic Publishers, Manufactured in the Netherlands.

[24] Jiawei Han, M. Kamber, "Data Mining-Concepts and Techniques", Sam Francisco 2009, Morgan Kanufmann Publishers.

[25] Khare N., Adlakha N., Pardasani K R., Karnaugh Map Model for Mining Association Rules in Large Databases, International Journal of Computer and Network Security. 2009; 1(2); 16–21.

[26] Leung C K S., Carmichael C L., Hao B., Efficient mining of frequent patterns from uncertain data, In Proc. IEEE ICDM Workshops'07. 2007; 489-494.

[27] Leung C K S., Hao B., Efficient algorithms for mining constrained frequent patterns from uncertain data, Proceedings of the 1st ACM SIGKDD Workshop on Knowledge Discovery from Uncertain Data. 2009; 9-18.

[28] Li Haoyuan, Yi Wang, Zhang Dong, Zhang Ming, Chang Edward, "PFP: Parallel FP Growth for query Recommendation".

[29] Lijuan Zhou, Xiang Wang, "Research of the FP Growth algorithm based on Cloud Environment", Journal of Software, March 2014, volume 9, N0. 3.

[30] Lin Y C., Hung C M., Huang Y M., Mining Ensemble Association Rules by Karnaugh Map, World Congress on Computer Science and Information Engineering. 2009; 320–324.

[31] Lin, D. I., &Kedem, Z. M. (1998). Pincer-search: A new algorithm for discovering the maximum frequent set. In Advances in Database Technology—EDBT'98 (pp. 103-119). Springer Berlin Heidelberg.

[32] Liu, G., Lu, H., Yu, J. X., Wang, W., & Xiao, X. (2003, November). AFOPT: An Efficient Implementation of Pattern Growth Approach. In FIMI.