Enhancing the Performance of Large-scale Profitable Itemset Mining using Efficient Data Structures

A Muralidhar, Aditya Ashwini Kumar Sathe, Pattabiraman V

Abstract: ***The process of extracting the most frequently bought items from a transactional database is termed as frequent itemset mining. Although it provides us with an idea of the best-selling itemsets, the method fails to identify the most profitable items from the database. It is not uncommon to have minimal intersection between frequent itemsets and profitable itemsets, and the process of extracting the most profitable itemsets is termed as Greater Profitable Itemset (GPI) mining. There have been various approaches to mine GPI in which [7] proposed a two-phased algorithm to optimize regeneration of GPI when the profit value of any item changes. This constituted of keeping track of the pruned items in the first phase and using it to efficiently regenerate GPI in the second phase. This paper proposes an enhancement to the way these changes are tracked by storing the pruned itemsets according to their constituent items, unlike the earlier algorithm that stored records iteration wise. By storing the itemsets according to their constituent items, we make sure that only the required items are being retrieved. In contrast, the earlier algorithm would fetch all the items pruned in any iteration, regardless of its relevance. By fetching only relevant itemset, the proposed method would significantly bring down the computational requirements.***

I. INTRODUCTION

It is becoming increasingly difficult to gain information from a pool of unorganized data due to the sheer increase in the amount of data being generated every day. Hundreds of thousands of individuals purchase goods in retail stores, supermarkets or online businesses. Finding trends in their transaction history would definitely help businesses and retailers better adapt to satisfy their customers' needs. There is a high chance that a person will buy a hair conditioner if he buys a shampoo. Such patterns give the enterprise an idea of the current trends in the consumer market, which aids them formulate a better marketing strategy to maximize profits. Association Rule Mining [4] has described several methods to find such itemsets. However, Association Rule Mining [5] cannot find out the highly profitable items in the transactions. With emerging markets and new customer bases, it is imperative to find out the highly profitable items along with the frequently purchased ones. Greater Profitable Itemset Mining [2] is an approach to find out the highly profitable Itemsets from a transactional dataset. GPI is based on the principle of Downward Closure Property [3]. Whenever the price of any item changes significantly, there is a chance that it might have an effect on the High Utility Itemsets generated; the whole method will need to be performed again. Regenerating the entire GPI from scratch is computationally expensive and time consuming. [7] proposed a two-phase algorithm that keeps track of all the pruned Itemsets in an iteration, and retrieves a particular record while regenerating Itemsets to reduce computational requirements. Although it was found out that the algorithm worked similar to the existing algorithms, the major drawback was that all the pruned Itemsets during an iteration were retrieved, regardless of the relevancy of each individual itemset.

Table 1: Example list of Transactions

Transaction -ID	Item A	Item B	Item C	Item D	Item E	Total Utility
1	2	3	1	0	0	56
2	0	0	0	5	1	85
3	3	1	0	6	0	122
4	0	4	0	3	0	89
5	0	1	2	0	3	59
6	0	0	1	1	5	74
7	2	0	0	0	3	44
8	0	4	0	1	2	79
Total	7	13	4	16	14	**608**

GPI requires transaction table as input which is shown in Table 1. A profit table is also displayed in Table 2 required to find profitable itemsets. Apart from that, a minimum support value (MSV) is required. Any item(set) with a transactional utility value above MSV is considered to be a GPI. This paper provides a better way of keeping track of pruned items for an efficient and effective retrieval. Instead of storing the pruned data iteration-wise, the data will be stored in a key-value table, where using any item as a key would return us an array of all the pruned Itemsets containg that particular item.

Table 2: Example profit table

Item	Profit
A	**7**
B	**11**
C	**9**
D	**15**
E	**10**

Revised Manuscript Received on July 06, 2019.
Muralidhar A, SCSE, Vellore Institute of Technology, Chennai,
Aditya Ashwinikumar Sathe2, SCSE, Vellore Institute of Technology, Chennai,
Pattabiraman V, SCSE, Vellore Institute of Technology, Chennai,

Retrieval Number: I8151078919/19©BEIESP
DOI:10.35940/ijitee.I8151.078919

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

II. LITERATURE SURVEY

A Two-Phase Mining Algorithm was proposed by Ying Liu et al to efficiently prune the number of candidates and obtain the complete set of greater profitable itemsets. A method to discover interesting patterns from a given transactional database was provided by Srikumar Krishnamoorthy [5], to support in excavating of greater profitable itemsets. It also engaged a new trimming approach which is appropriate in trimming unwanted candidates. Chun-Wei Lin et al [6] proposed an algorithm for updating greater profitable itemsets in energetic databases when a part of transactions is deleted. The transaction-weighted utilization (TWU) itemsets were subdivided based on the transaction weighted-utilization of an itemset in the original dataset and in the deleted list. None of the previous contributions clearly define a procedure to update the GPI in case the profit value of any item changes. The profit value of an item can change due to an alteration in trading policies, fluctuations in the availability of raw material or a change in the organization's marketing strategy. Muralidhar et. al [7] proposed a novel algorithm that takes expedites regeneration of GPI whenever there is a significant change in the profit value of one or more items.

III. PROPOSED WORK

The algorithm keeps track of the Itemsets pruned in every iteration. The algorithm has two phases. It creates a GPI from the given dataset in the first phase. During this phase, it creates a record to store all the pruned Itemsets. This paper proposes a new method to store the records, using a key-value pairing.

The following guidelines are followed whenever storing the records in the key-value pair:

i) If an item is pruned in the i^{th} iteration, the values for the corresponding key will be an array of all the i-combinations of the items in which that particular item is a constituent.

ii) For the case of items excluded in the first iteration, a list of all individual items is to be stored.

In the second phase, the algorithm uses the record table along with the new profit value table to re-generate the GPI. It creates a preliminary GPI and then compares all the items in the dataset with the items in the new initial GPI and the items in the initial GPI generated in the first phase. Then the action based on one of the four cases is accomplished:

1) If the item is in the original GPI, but not in the earlier GPI, start the GPI mining procedure from begins to produce entirely the GPI for this item.

2) If the item is in the original GPI and in the earlier GPI, use this item as the 'key' and get the 'value', or, the required record from the data structure and implement it.

3) If the item is not in original GPI but is in the previous GPI, then eliminate entirely the itemsets where the particular item seemed in the previous GPI and make a set join process of the earlier GPI with the original GPI.

4) If the item is not in the original GPI and is not in the earlier GPI as well, avoid the particular item, as it is not required.

Table 5 provides an in-depth insight about the way items were stored during the first phase. It can be easily found out that in any iteration, there are a lot of different Itemsets. According to the algorithm, the first instance of a record in which a particular itemset exists is retrieved for further processing. The major downfall of this kind of structure is that even if we need to process a single itemset, all the other irrelevant Itemsets are retrieved, which creates computational overhead. Table 6 displays the proposed method of storing itemsets, where each row caters to a specific item, instead of a specific iteration. Such kind of key-value pairing can help ensure that only the needed Itemsets are retrieved during any iteration, reducing computational requirements.

IV. IMPLEMENTATION

Table 3: Transaction Table of Online Retail Dataset

TID	Transactions
1	'76T32' (6), '65IKW' (8), '71437' (6), '80132' (6), 'A43VC' (2), '54109O' (6), '14582' (6)
2	'22633' (6), '22632' (6)
3	'22748' (6), '22745A' (6), '21755' (3), '22749P' (8), '84969' (6), '22310' (6), '22622J' (2)
4	'22914' (3), '22912' (3), '22913' (3), '22960B' (6)
5	'21756' (3)
6	'22727' (24), '21724A' (12), '21883' (24), '21731B' (24), '22629' (24), '22631L' (24)
7	'22086' (80)
8	'22633' (6), '22632B' (6)
9	'82482' (6), 'A43VC' (2), '71437' (6), '14582' (6), '82486' (4), '20679' (6), '21068' (6), '76T32' (6)
10	'21258' (32)

Table 4: Profit Table of Online Retail Dataset

Item ID	Profit Value per Unit
22727	5
22086	16
21258	3
22914	14
A43VC	20
22540	11
14582	15
21068	25

Table 5: Sample transactions which are mentioned in [7]

Iteration	Itemsets
1	{76T32, 71437, 22727, 14582}
2	{(21068, 22520),(21068, 22086),(22914, 21068),(22914, 22520)}
…	…

Table 6: Keep rack of records of the proposed method

Key	Value
81253A	{81523A, 22914, 21068, …}
71437	{71437, 21086, 22914, …}
21068	{(21068, 22520), (21068, 22086), (21086, 81523A), …}
22914	{(22914, 21068), (22914, 22520), (22914, 76T32), …}

Two datasets are taken, which are detailed in Table 7. These datasets are further split into ten mini-datasets, with each mini-dataset having 10% more transactions than the previous one. Once the first phase of the algorithm is over, the profit value of some items is changed randomly. This enables the succeeding stage of the proposed algorithm to execute. The algorithm runs for 10 times for one dataset, and it is divided into ten mini-datasets, having 10%, 20%, and so on of the transactions in the original dataset.

Table 7: Details of the datasets

Dataset	No. of Transaction	No. of Items
Chess Dataset	3,200	75
Online Retail Store	25,900	1240

TABLE 8: EXAMPLE OF A PROFIT TABLE AFTER UPDATION

Item	Profit
A	1
B	11
C	9
D	5
E	10

Table 9: Methods and the variable return

Method	Returned Variables
Transaction_Total	TT, Tsum
Transaction_Wt_Util	TWUx
GPI_Create	GPI

Table 10: Variables and their meanings

Variable Name	Meaning
Tot_Tran	k-sized array, ‘k’ being total number of transactions. TT[i] gives cost of transaction [i]
Trans_sum	Sum of all transactions
TWUx	m-sized array, ‘m’ being total number of items. TWUx[i] gives summation of transaction profits in which item ‘i’ was involved.
GPI, GPI_New	Preliminary list which is further built upon in first and second phases.
Lam Rem, rem_New	Lambda value used to compute min_threshold, min_Threshold_New, the cutoff values. List of all items excluded from GPI and GPI_New
iter_record	Key-value table of Itemsets eliminated in first phase

Phase one:

- Tot_Tran, Tran_sum ← Tr_tot( )
- TWU ← Transaction_Weighted_Utility(TT)
- Th ← Trans_sum * lam
- GPI, rem ← GPI_Create(threshold, TWU)
- for n in range 1 to len(GPI) + 1:
    a. consider all probable mixture of the items in the GPI of dimension ‘n’
    b. for a group, estimate the entire Transaction Cost
    c. If this Transaction Cost is above the threshold value, comprise it in GPI_First.
    d. if this Transaction Cost is under the threshold value, contain the items in a distinct set ‘erased’
    e. for every item in the set ‘removed’, push the appropriate value to as per the above guidelines to the corresponding keys in a structure ‘iter_record’
- end for

Phase Two:

- TT_New, Tsum_New ← Transaction_Total()
- Threshold_new ← Tsum_New * lam
- TWU_New ← Transaction_Wt_Util(TT_New)
- GPI_New, rem_New← GPI_Create(threshold_new, TWU_New)
- for all item in total items:
    - if item is in GPI_old and GPI_New:
        - retrieve the value for the key ‘item’ from iter_record and put

it in an collection 't'
- for n in series 1 to len(t) + 1:
  1. consider all potential group of the items in the GPI of dimension 'n'.
  2. for a group, compute the entire Transaction Cost
  3. if this Transaction Cost is above the threshold value, comprise it in the Final_GPI
  4. if this Transaction Cost is under the threshold value, avoid.
- end for

- end if
- else if item is in GPI_old and item is not in GPI_New:
  - for all entries in GPI_First:
    1. if the item is not in the entry, comprise it in Final_GPI
  - end for
- end if
- else if item is not in GPI but is in GPI_New:
  - create groupings with all additional items and associate it transaction cost with the threshold value
  - if the cost is greater than the threshold, comprise it in Final_GPI
- end if
- else if item is not in GPI and not in GPI_New:
  - continue
- end if

- return Final_GPI

V. EXPERIMENTAL SETUP

The datasets described in Table 7 were taken, pre-processed and fed to the algorithm. The algorithm executed in a python3 environment on a MacBook Pro (2015 Model) with 16 GB of memory and an Intel Core i7 4770HQ CPU. Parallelization was not done and the algorithm was limited to a single core.

VI. RESULT AND DISCUSSION

The execution times for the algorithm with the existing data structure and the proposed data structure were observed. Further, a graph was plotted against the number of transactions and execution time. As expected, the proposed method outperformed the older method by a considerable margin. Such results are obtained due to the fact now only the Itemsets relevant to an item are retrieved and processed.

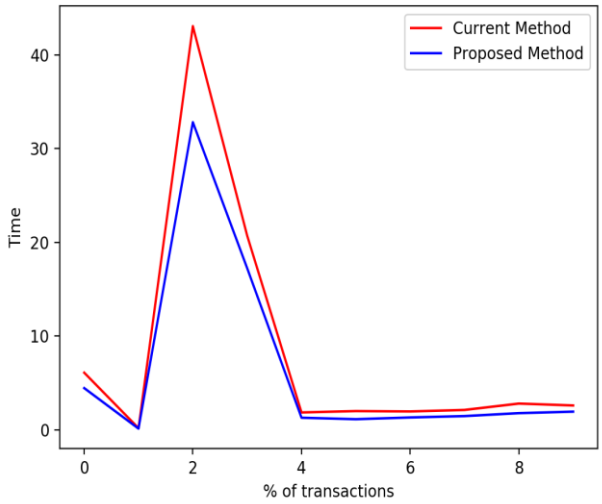

Figure 1: Comparison of two methods with Chess Dataset

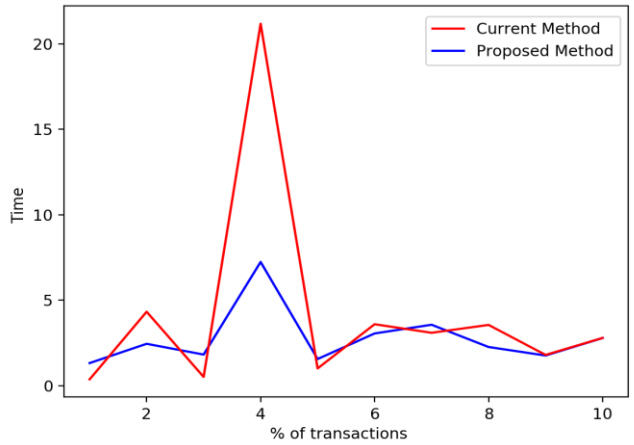

Figure 2: Comparison of two methods with Online-Retail Dataset

VII. CONCLUSION AND FUTURE WORK

The proposed system make use of a specialized data structure which can enhance the efficiency of the algorithm. However, the data structure can be further modified to create a three or more dimensional matrix, to account for a specific item pruned during the iteration. It will be interesting to observe the results of such a data structure. Further, we observe that the same itemset might exist across various keys, depending on the constituent items. A method to prevent processing of redundant itemsets needs to be devised and implemented.

REFERENCES

1. R. Agrawal , T. Imielinski, A. Swami, 1993, mining association rules between sets of items in large databases, in: proceedings of the ACM SIGMOD International Conference on Management of data, pp. 207-216 (Conference Proceedings)
2. D Dubey, S Bhattacharya, High Utility Itemset Mining, International Journal of Emerging Technology and Advanced Engineering, Vol. 2, Issue 8, pp. 476 – 481 (2012) (Journal)
3. Liu, Ying, Wei-keng Liao, and Alok Choudhary. "A two-phase algorithm for fast discovery of high utility itemsets." *Pacific-Asia Conference on Knowledge Discovery and Data Mining*.

Springer Berlin Heidelberg, 2005. (Conference Proceedings)
4. Vaidya, Jaideep, and Chris Clifton. "Privacy preserving association rule mining in vertically partitioned data." *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002. (Conference Proceedings)
5. Srikumar Krishnamoorthy, "Pruning Strategies for mining high utility itemsets." Expert Systems with Applications, Issue 42, pp. 2371 – 2381, (2015) (Journal)
6. Chun-Wei Lin, Tzung-Pei Hong, Guo-Cheng Lan, Jia-Wei Wong, Wen-Yang Lin, "Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases." Advanced Engineering Informatics, Issue 29, pp. 16 – 27, (2015) (Journal)
7. Muralidhar A, Pattabiraman V, Aditya Ashiwnikumar, "A Novel Method to Extract Greater Profitable Itemsets", International Journal of Pure and Applied Mathematics, Vol. 109, Issue 8, pp. 83 – 90, (2017) (Journal)