ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 9, Issue 6, June 2022

Efficiently Mining Frequent Item Sets on Massive Data

1.Devayani.J 2.Dhivya.R 3.Elavarasi.M

4.Janaki.P Ms. R.Geetha (Assistant Professors for CSE)

Department of computer science and Engineering

Bharathiyar Institute of Engineering for Women,

Deviyakurichi-636112

ABSTRACT

Frequent item set mining is an important operation to return all item sets in the transaction table, which occur as a subset of at least a specified fraction of the transactions. The existing algorithms cannot compute frequent item sets on massive data efficiently, since they either require multiple-pass scans on the table or construct complex data structures which normally exceed the available memory on massive data. This paper proposes a novel precomputation-based frequent item set mining (PFIM) algorithm to compute the frequent item sets quickly on massive data. PFIM treats the transaction table as two parts: the large old table storing historical data and the relatively small new table storing newly generated data. PFIM first preconstructs the quasi-frequent item sets on the old table whose supports are above the lower-bound of the practical support level. Given the specified support threshold, PFIM can quickly return the required frequent item sets on the table by utilizing the quasi-frequent item sets. Three pruning rules are presented to reduce the size of the involved candidates. An incremental update strategy is devised to efficiently re-construct the quasi-frequent item sets when the tables are merged. The extensive experimental results, conducted on synthetic and real-life data sets, show that PFIM has a significant advantage over the existing algorithms and runs two orders of magnitude faster than the latest algorithm.

INDEX TERMS

Frequent item set mining, massive data, PFIM algorithm, pruning rule, incremental update.

INTRODUCTION

Frequent item set mining is an important operation that has been widely studied in many practical applications, such as data mining [1]–[3], software bug detection, spatiotemporal data analysis and biological analysis. Given a transaction table, in which each transaction contains a set of items, frequent item set mining returns all sets of items whose frequencies (also referred to as support of the set of items) in the table are above a given threshold. Due to its practical importance, since firstly proposed in [6], frequent item set mining has received extensive attentions and many algorithms are proposed [7]–[9]. The existing frequent item set mining algorithms can be classified into two groups: candidate-generation-based algorithms [10]–[14] and pattern-growth-based algorithms [15]–[17]. The candidate-generation-based algorithms first generate candidate item sets and these candidates are

validated against the transaction table to identify frequent item set. The anti-monotone property is utilized in candidategeneration-based algorithms to prune search space. But the candidate-generation-based algorithms require multiple-pass table scans and this will incur a high I/O cost on massive data. The pattern-growth-based algorithms do not generate candidates explicitly. They construct the special tree-based data structures to keep the essential information about the frequent item sets of the transaction table. By use of the constructed data structures, the frequent item sets can be computed efficiently. However, pattern-growth-based algorithms have the problem that the constructed data structures are complex and usually exceed the available memory on massive data. In frequent item set mining, the number of the frequent item sets normally is sensitive to the value of the support threshold. If the support threshold is small, there will be a large number of frequent item sets and it is difficult for the users to make efficient decisions. On the contrary, if the support threshold is large, it is possible that no frequent item sets can be discovered or the interesting item sets may e missed. The focus of this paper is to find a new efficient algorithm to compute frequent item sets on massive data quickly. One useful trick, which is adopted to speed up the execution in the existing algorithms, is to reuse the work done in the counting operation of the shorter item sets for that of the longer item sets. In this paper, we want to utilize this reuse idea to a much larger degree. Therefore, the overall data set can be divided into two parts: the much larger old data set storing the historical data, and the relative small new data set storing the newly generated data. Based on the description above, this paper devises a new PFIM algorithm (Precomputation-based Frequent Item set Mining algorithm) on massive data, which utilizes the pre-constructed frequent item sets on the old data set to return the frequent item sets quickly. The extensive experiments are conducted on synthetic and real-life data sets. The experimental results show that, PFIM outperforms the existing algorithms significantly, it runs two orders of magnitude faster than the latest algorithm.

The contributions of this paper are listed as follows:

- This paper proposes a novel precomputation-based PFIM algorithm to compute frequent item sets on massive data efficiently.
- Three pruning rules are proposed in this paper to reduce the number of the candidate frequent item sets.
- An incremental update strategy is devised to re-construct the quasi-frequent itemsets quickly.
- The experimental results show that PFIM has a significant advantage over the existing algorithms.

RELATED WORKS

The existing algorithms for frequent itemset mining can be divided into two groups mainly: candidate-generation based algorithms and pattern-growth-based algorithms. This section will review the two kinds of algorithms respectivel.

CANDIDATE-GENERATION-BASED ALGORITHMS

The candidate-generation-based algorithms firstly generate the candidates of the frequent item sets, then the candidates are validated against the transaction table, and the frequent item sets are discovered. It uses the downward closure property, i.e. any superset of an infrequent itemset must also be infrequent, to prune the search space. By a pass of scan on the transaction table, it first counts the item occurrences to find the frequent 1-itemsets F1. Subsequently, the frequent k-item sets in Fk are used to generate the candidates Ck+1 of the frequent (k +

1)-item sets. Another pass of scan is needed to compute the supports of candidates in Ck+1 to find the frequent (k + 1)-itemsets Fk+1. This process iterates similarly until the Fk+1 is empty. Apriori algorithm often needs multiple passes over table, it will incur a high I/O cost on massive data. Savasere et al. [12] propose Partition algorithm to generate frequent itemsets by reading the transaction table at most two times.

The execution of Partition consists of two stages.

In the first stage, Partition algorithm divides the table into a number of non-overlapping partitions in terms of the allocated memory, and the local frequent itemsets for each partition are computed. All the local frequent itemsets are merged at the end of first stage to generate the candidates of frequent itemsets.

In the second phase, another pass over table is performed to acquire the support of the candidates and the global frequent item sets can be discovered. The useful property adopted in Partition is that, every global frequent item sets must be appeared in local frequent item sets of at least one partition. Partition algorithm utilizes vertical table representation of transaction table and the support counting is performed by recursive TID (transaction identifier) list intersection. In the first phase, Partition may generate many false positives, i.e. the item sets are frequent locally but not frequent globally. Therefore, it needs another table scan to remove the false positives. Depending on the allocated memory size, Eclat can recursively partition large classes into smaller ones until each class can be maintained entirely in the memory. Then, each class is processed independently in the breath-first fashion to compute the frequent itemsets. Eclat processes the sublattices sequentially one by one and does not need post-processing overhead as Partition algorithms. Each node in PPC-tree is associated with pre-post code via the pre-order and post-order traversal on the PPC-tree. Each frequent item can be represented by a node-list, i.e. the list of Pre-Post code consisting pre-order code, post-order code and the count of nodes registering the frequent item. PPV fully uses candidate generation to discover frequent itemsets, i.e. the node-lists of the candidate itemsets of length (k + 1) are generated by intersecting node-lists of frequent itemsets of length k, then the frequent itemsets can be reported. PPV can achieve a high execution efficiency since (1) the node-list is more compact than the vertical structure, (2) the support counting is transformed into the intersection of node-lists, (3) the ancestor-descendant relationship of two nodes can be verified efficiently by their pre-post codes.

Three key advantages of negFIN are:

(1) employing bitwise operator to generate new sets of nodes

(2) reducing the time complexity of discovering frequent itemsets to O(n)

(3) using a promotion method to prune the search space in set-enumeration.

PATTERN-GROWTH-BASED ALGORITHMS

Pattern-growth-based algorithms do not generate candidate itemsets explicitly but compress the required information for frequent itemsets in specific data structure. The frequent itemsets can be acquired quickly with the notion of projected databases, a subset of the original transaction database relevant to the enumeration node. The examination process of a node refers to the support counting of the candidate extension of the node. During the search, the projected transaction sets are maintained for some of the nodes on the path from the root to the node P currently being

extended. Normally, the projected transaction sets only contain the relevant part of the transaction database for counting the support at the node P. At the lower levels of the lexicographic tree, a specialized counting technique called bucketing is used to substantially improve the counting time. Han et al. [16] propose a FP-tree-based FP-growth algorithm to mine the complete set of frequent patterns by pattern fragment growth. FP-tree (frequent-pattern tree) is a compact prefix-based trie structure to store the essential information about frequent patterns. In each transaction, only frequent length-1 items, which are sorted with the descending order of support, are used to construct the FP-tree. Then the FP-growth algorithm works on FP-tree rather than on the original database to mine frequent patterns. FP-growth algorithm starts with a frequent length-1 pattern (initial suffix pattern), and the set of frequent items co-occurring with the suffix pattern is extracted as conditional-pattern base, which is then constructed as conditional FP-tree. With the current suffix pattern and the conditional FP-tree, if the conditional FP-tree is not empty, FP-growth performs mining recursively. A special data structure, FP-array, is devised. Given an itemset of m items, FP-array is a (m − 1) × (m − 1) matrix, where each element of the matrix corresponds to the counter of an ordered pair of items. By the special data structure, a new FPgrowth* is proposed, which can reduce the traversal time on FP-tree and speed up the FP-growth method significantly.

PRELIMINARIES

Given a transaction table T of n transactions, each of transactions is a subset of the universe of items U = {i1, i2, . . . , id }. Here, the itemset is a subset of U and a k-itemset is an itemset with k items. A unique transaction identifier TID is associated with every transaction.

Given an itemset IS, itssupport sup(T , IS) is defined as the fraction of transa contain IS as a subset, i.e. sup(T , IS) = |{t|IS ⊆ t, t ∈ T }| n Obviously, the support measures the correlation of the items. For an itemset IS, its greater support value means that the items of IS occur together more frequently in T. Definition 1 (Frequent Itemset Mining): Given a transaction table T and a specified support threshold minsup, frequent itemset mining determines all itemsets whose supports are no less than minsup.

Transaction table T

TID	Items
1	7, 8, 9
2	1, 6, 7
3	9
4	0, 4, 9
5	3, 6, 9
6	0, 1, 4, 9
7	0, 6
8	0, 1, 4, 9
9	1, 2, 3, 6, 7, 8, 9
10	1, 2, 3, 8, 9
11	0, 9
12	2, 8, 9
13	4, 6, 9
14	7
15	2, 3, 4, 7, 8, 9

frequent itemsets given minsup = 0.2

{0}: 0.33, {1}: 0.33, {2}: 0.27, {3}: 0.27, {4}: 0.33, {6}: 0.33, {7}: 0.33, {8}: 0.33, {9}: 0.80, {0, 4}: 0.20, {0, 9}: 0.27, {1, 9}: 0.27, {2, 3}: 0.20, {2, 8}: 0.27, {2, 9}: 0.27, {3, 8}: 0.20, {3, 9}: 0.27, {4, 9}: 0.33, {6, 9}: 0.20, {7, 8}: 0.20, {7, 9}: 0.20, {8, 9}: 0.33, {0, 4, 9}: 0.20, {2, 3, 8}: 0.20, {2, 3, 9}: 0.20, {2, 8, 9}: 0.27, {3, 8, 9}: 0.20, {7, 8, 9}: 0.20, {2, 3, 8, 9}: 0.20

FIGURE 1. The illustration of transaction table in running example.

TABLE 1. Summary of symbols.

Symbol	Meaning
T	the transaction table
d	the number of items
n	the number of transactions in T
T_O	the old table storing historical data
tn_o	the number of transactions in T_O
T_Δ	the new table storing newly generated data
tn_Δ	the number of transactions in T_Δ
$minsup$	the specified support threshold
ω	the lower-bound of practical support threshold
F_{qf}	the file storing the quasi-frequent itemsets
cnt_Δ	the array keeping the count of the items in T_Δ
mas_Δ	the maximum count for items in T_Δ

ions in T which

PFIM ALGORITHM

INTUITIVE IDEA

This part describes intuitive idea of PFIM algorithm. Generally, the number of frequent itemsets is very

sensitive to the value of minsup. If the value of minsup is too small, the number of frequent itemsets will be so large that the users can become overwhelmed with too many results and it is difficult for users to find the really useful information from them. Therefore, in this paper, we assume that there exists a lower-bound for the value of minsup in practical applications. The lower-bound is denoted by ω in this paper. The value of ω can be determined by some domain experts, or the lowest value of the support used in the past frequent itemset mining. On massive data, the existing algorithms often cannot meet the users' requirement, they either need to scan the table multiple times, or need a complex data structure and a high memory consumption. This is the motivation of this paper, i.e. we want to devise a highly efficient algorithm to mine the frequent itemsets on massive data quickly. Some of the existing algorithms, such as FP-tree-based methods

or vertical-representation-based methods, reuse the work that has already been done previously in the current frequent itemset mining, so they can discover frequent itemsets faster. But, when the current frequent itemset mining is done, their works are lost and the next mining still needs to be executed from scratch. On massive data applications, data usually is stored in read/append-only mode [19]. Therefore, the overall transaction table T can be divided into two parts: the large old transaction table TO and the relative small new transaction table T1, i.e. T = TO∪T1. Usually T1 keeps the accumulated new transactions. When the size of T1 reaches to some level, for example, 5% of the size of TO, the data in T1 will be merged into TO. Since the size of TO is much larger than that of T1, we have enough confidence that the time interval of two consecutive merging operations should be long enough. During the time interval between two consecutive merging, TO remains unchanged and only T1 updates frequently. Under such circumstances, given the frequent itemset mining with varying support thresholds, why not we keep the precomputed itemsets whose support values in TO are no less than ω and only compute the required frequent itemsets considering the existence of T1. In this way, the work done for TO can be reused for all the frequent itemset mining in a long enough time. This is the motivation why we develop PFIM algorithm. In the rest of this section, we first show the precomputation operation in Section IV-B, then introduce PFIM algorithm detailedly in Section IV-C and Section IV-D. The update operation of the pre-constructed itemsets are presented in Section IV-E, and some issues are discussed in Section IV-F.

A. PRE-COMPUTATION OPERATION

This part describes the pre-computation operation to generate the required itemsets on the large old transaction table TO whose supports are no less than ω. The required itemsets here are referred to as quasi-frequent itemsets, distinguishing from the frequent itemsets with the support threshold minsup specified by users. Let tno be the number of transactions in TO and tn1 be the number of transactions in T1. Since the size of TO is much large, usually exceeds the size of the allocated memory. Therefore, the process of pre-computing the quasifrequent itemsets consists of two stages: candidate generation and result refinement. In the stage of candidate generation, we retrieve the transactions in TO sequentially and maintain the retrieved transactions in an in-memory buffer BUF, whose size is set according to the size of the allocated memory. If BUF is full, we can compute the local quasi-frequent itemsets in BUF by the current vertical frequent itemset mining algorithms. The quasi-frequent itemsets corresponding to current BUF are kept in a file. Then we empty BUF and continue the sequential scan for the next iteration.

The process is similarly executed until all transactions in TO is retrieved and all local quasi-frequent itemsets are generated. [5] emphasized that Security is an important issue in current and next-generation networks. Blockchain will be an appropriate technology for securely sharing information in next-generation networks. Digital images are the prime medium attacked by cyber attackers. In this paper, a blockchain based security framework is proposed for sharing digital images in a multi user environment.

Old transaction table T_O

TID	*Items*
1	7, 8, 9
2	1, 6, 7
3	9
4	0, 4, 9
5	3, 6, 9
6	0, 1, 4, 9
7	0, 6
8	0, 1, 4, 9
9	1, 2, 3, 6, 7, 8, 9
10	1, 2, 3, 8, 9
11	0, 9
12	2, 8, 9

New transaction table T_Δ

TID	*Items*
1	4, 6, 9
2	7
3	2, 3, 4, 7, 8, 9

F_{qf}

IS	9	1	0	8	6	8,9	1,9	0,9	7	4	3	2	4,9	3,9	2,9	2,8	0,4	2,8,9	0,4,9
SUP	10	5	5	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	3

quasi frequent itemsets with SUP =2

{7,9}, {7,8}, {6,9}, {6,7}, {3,8}, {3,6}, {2,3}, {1,8}, {1,7}, {1,6}, {1,4}, {1,3}, {1,2}, {0,1}, {7,8,9}, {3,8,9}, {3,6,9}, {2,3,9}, {2,3,8}, {1,8,9}, {1,6,7}, {1,4,9}, {1,3,9}, {1,3,8}, {1,2,9}, {1,2,8}, {1,2,3}, {0,1,9}, {0,1,4}, {2,3,8,9}, {1,3,8,9}, {1,2,8,9}, {1,2,3,9}, {1,2,3,8}, {0,1,4,9}, {1,2,3,8,9},

B. BASIC PROCESS

Given the support threshold minsup, this part introduces the basic process that PFIM discovers the frequent itemsets on T = TO ∪ T1.

1)THE SPECIAL CASE

First, we discuss a special case. If T1 is empty, i.e. tn1 = 0, the processing of PFIM is simple. It just needs to read the quasi-frequent itemsets in Fqf sequentially. ∀t ∈ Fqf , let t be the current element retrieved in Fqf . If t.SUP ≥ dtno × minsupe, t.IS is reported as a frequent itemset. Since the elements in Fqf are arranged in descending order of SUP, if t.SUP < dtno × minsupe, it can be guaranteed that all frequent itemsets are discovered and the sequential scan on Fqf terminates

2) THE GENERAL CASE

Of course, usually, T1 is not empty. Due to the existence of new transactions, we may find new frequent itemsets from T1 and TO which are not contained in Fqf . In the rest of this part, we describe the processing of PFIM given that T1 is not empty. An itemset is frequent, if there are at least dn × minsupe transactions in T containing it, where n = tno+tn1 and T = TO ∪T1.

PRUNING OPERATION

In terms of the description in Section IV-C, PFIM can reuse the pre-computation result of TO and reduce the execution cost significantly. In this part, we discuss how to improve PFIM further to speed up its execution by pruning operation. 1) PRUNING IN STEP 2 One main part of the cost in PFIM is to compute the support counts of the itemsets of STCAD in T1, i.e. step 3 in Section IV-C.2. Therefore, if we can reduce the number of itemsets in STCAD in step 2, the counting cost in T1 can be decreased. In Section IV-C.2, ∀t ∈ STCAD, it satisfies: dn × minsupe − mas1 ≤ t.SUP < dn × minsupe That is, we use the maximum count mas1 of the single item in T1 to determine the support count range of the possible frequent itemsets. Obviously, if we can narrow down the support count range, the size of STCAD can be reduced. As described in the process of step 2, PFIM can determine directly whether the quasi-frequent 1-itemsets in Fqf are frequent itemsets. At the end of step 2, PFIM maintains the possible frequent itemsets in STCAD. Before entering the step 3, we wonder whether the size of STCAD can be decreased further. But, it should be noted that, the reason to prune the itemsets in STCAD is that the execution cost in step 3 can be high if STCAD

contains many itemsets, the cost of pruning operation should keep low also. Otherwise, the overall cost in step 2 and step 3 can still be large, which can affect the high efficiency of PFIM. In order to prune itemsets in STCAD as many as possible with a low cost, PFIM first chooses two items of each quasifrequent itemset in STCAD which have the smallest support counts in cnt1. $\forall t \in$ STCAD, we keep an item pair (it,1, it,2) in t.IS in PIP (Pruning Item Pair), (it,1, it,2) are two items with the smallest support counts in cnt1 among all items in t.IS and it,1 $\prec$ it,2. PFIM computes the support counts of the item pairs in PIP in the similar operation as step 3 in Section IV-C.2.

UPDATE OPERATION

As the description above, the new transactions are accumulated in T1. When the size of T1 reaches a certain threshold, for example, 5% of the size of TO, the transactions in T1 and TO are merged. At this point, the quasi-frequent itemsets in Fqf needs to be updated also. Of course, re-construction totally is one choice, i.e. re-compute the quasi-frequent itemsets with the support threshold ω on T = TO $\cup$ T1 from scratch. But the total re-construction can be expensive. Therefore, in this paper, we propose an incremental update strategy, which utilizes the existing information computed already to speed up the update operation. [2] discussed about a method, Optimality results are presented for an end-to-end inference approach to correct(i.e., diagnose and repair) probabilistic network faults at minimum expected cost. One motivating application of using this end-to-end inference approach is an externally managed overlay network, where we cannot directly access and monitor nodes that are independently operated by different administrative domains, but instead we must infer failures via end to-end measurements. [4] discussed about a method, Sensor network consists of low cost battery powered nodes which is limited in power. Hence power efficient methods are needed for data gathering and aggregation in order to achieve prolonged network life. However, there are several energy efficient routing protocols in the literature; quiet of them are centralized approaches, that is low energy conservation.

DISCUSSIONS

This paper assumes that there exists a lower-bound ω for the value of minsup in practical applications. The value of ω is determined by the domain experts or the lowest value of the used support levels. However, some user may submit a frequent itemset mining with the specified minsup which is lower than ω. Although this case should be quite unusual (too many frequent itemsets can be generated), we still hope that PFIM can deal with this case. Total re-computation on T = TO $\cup$ T1 is one choice. But this choice should be expensive and it neglects the pre-computation result of the existing Fqf . A proper alternative is to reuse the precomputation result. Fqf maintains the quasi-frequent itemsets whose support counts in TO are no less than $d\omega \times tnoe$. We first compute the frequent itemsets on TO with support level minsup (here minsup < ω).

PERFORMANCE EVALUATION

A. **EXPERIMENTAL SETTINGS**

To evaluate the performance of PFIM, we implement it in Java with jdk-8u20-windows-x64. The experiments are executed on LENOVO ThinkCentre M8400 (Intel (R) Core(TM) i7-3770 CPU @ 3.40GHz (8 CPUs) + 32G memory + 64 bit windows 7). The used data set is stored in Seagate Expansion STBV3000300 (3TB). In the experiments, the performance of PFIM is evaluated against Apriori [20] and negFIN [25]. The reason to select the two algorithms for performance evaluation is that, (1) Apriori is a classic level-wise algorithm and thus we select it as the baseline algorithm, (2) negFIN is the latest algorithm and it shows a

performance advantage over the other main frequent itemset mining algorithms [25].

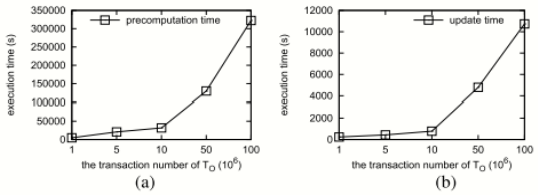

FIGURE 8. The result of pre-computation and update. (a) Pre-computation operation. (b) Incremental update.

PRE-COMPUTATION OPERATION AND UPDATE OPERATION

In this part, we report the execution times of precomputation operation and the update operation. As depicted in Figure 8(a), given that tn1/tno = 0.03, we illustrate the pre-computation cost of PFIM with the varying values of tno. The pre-computation time of PFIM increases quickly with the greater value of tno. At tno = 100×106 , the pre-computation time is 322337.466s. For one hand, this indicates that frequent itemset mining algorithms require a rather long execution time to compute frequent itemsets if the value of minsup is small. For the other hand, the pre-computation operation is expensive on massive data. But it should be noted that the pre-computation only is executed once from scratch and then the incremental update can be performed.

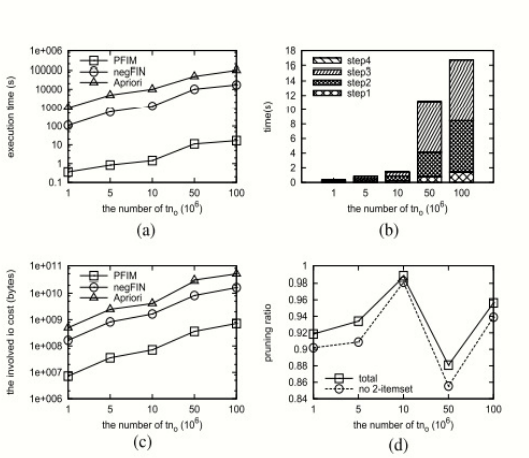

FIGURE 9. The effect of transaction number in old table. (a) Execution time. (b) The time decomposition. (c) The I/O cost. (d) Pruning ratio of PR1.

REFERENCES

[1] A. Ceglar and J. F. Roddick, ‘‘Association mining,’’ ACM Comput. Surv., vol. 38, no. 2, p. 5, 2006.

[2] Christo Ananth, Mona, Kamali, Kausalya, Muthulakshmi, P.Arthy, “Efficient Cost Correction of Faulty Overlay nodes”, International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE), Volume 1,Issue 1, August 2015,pp:26-28.

[3] H. Wang, W. Wang, J. Yang, and P. S. Yu, ‘‘Clustering by pattern similarity in large data sets,’’ in Proc. ACM SIGMOD Int. Conf. Manage. Data, Jun. 2002, pp. 394–405.

[4] Christo Ananth, S.Mathu Muhila, N.Priyadharshini, G.Sudha, P.Venkateswari, H.Vishali, “A New Energy Efficient Routing Scheme for Data Gathering “,International Journal Of Advanced Research Trends In Engineering And Technology (IJARTET), Vol. 2, Issue 10, October 2015), pp: 1-4.

[5] Christo Ananth, Denslin Brabin, Sriramulu Bojjagani, “Blockchain based security framework for sharing digital images using reversible data hiding and encryption”, Multimedia Tools and

Applications, Springer US, Volume 81,Issue 6, March 2022,pp. 1-18.
[6] R. Agrawal, T. Imielinski, and A. Swami, ‘‘Database mining: A performance perspective,’’ IEEE Trans. Knowl. Data Eng., vol. 5, no. 6, pp. 914–925, Dec. 1993.
[7] C. C. Aggarwal, Data Mining: The Textbook. Cham, Switzerland: Springer, 2015.