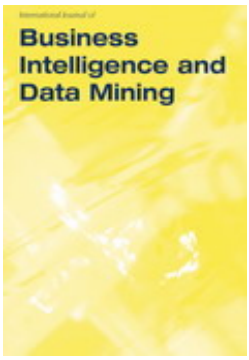


International Journal of Business Intelligence and Data Mining

ISSN online: 1743-8195 - ISSN print: 1743-8187
https://www.inderscience.com/ijbidm

Approaches to parallelise Eclat algorithm and analysing its performance for *K* length prefix-based equivalence classes

C.G. Anupama, C. Lakshmi

DOI: 10.1504/IJBIDM.2022.10043400

Article History:
Received: 26 August 2021
Accepted: 26 October 2021
Published online: 30 November 2022

Approaches to parallelise Eclat algorithm and analysing its performance for *K* length prefix-based equivalence classes

C.G. Anupama* and C. Lakshmi

Department of Computational Intelligence,
School of Computing,
SRM Institute of Science and Technology,
SRM Nagar, Kattankulathur 603203, Kanchipuram, Chennai, TN, India
Email: anupamag@srmist.edu.in
Email: lakshmic@srmist.edu.in
*Corresponding author

Abstract: Frequent item set mining (FIM), is one of the prevalent, well-known methods of data mining and is a topic of interest for researchers in the field of decision making. With the establishment of the period of big data where the data is continuously generated from multidimensional sources with enormous volume, and variety in an almost unrevealed way, transforming this data into valuable knowledge discovery which can add value to the organisations to make an efficient decision making poses a challenge in the present research. This leads to the problem of discovery of the maximum frequent patterns in vast datasets and to create a more generalised and interpretable representation of veracity. Targeting the problems stated above, this paper suggests a parallelisation method suitable for any type of parallel environment. The implemented algorithm can be run on a single computer with multi-core processor as well as on a cluster of such machines.

Keywords: item set mining; frequent items; frequent patterns; Eclat; parallel Eclat; frequent item set mining; FIM.

Reference to this paper should be made as follows: Anupama, C.G. and Lakshmi, C. (2023) 'Approaches to parallelise Eclat algorithm and analysing its performance for *K* length prefix-based equivalence classes', *Int. J. Business Intelligence and Data Mining*, Vol. 22, Nos. 1/2, pp.34–48.

Biographical notes: C.G. Anupama holds a Postgraduate in Software Engineering and currently pursuing her PhD in Computer Science and Engineering. She is currently working as an Assistant Professor in the Department of Computational Intelligence, School of Computing, SRMIST, Tamil Nadu, India with 10+ years of experience and 1+ year industry experience on software automation testing. Her research interest includes data mining, big data, software engineering, software testing, artificial intelligence and machine learning.

C. Lakshmi holds a PhD in Computer Science and Engineering from the SRM University and a Professor in Department of Computational Intelligence, School of Computing, SRM Institute of Science and Technology. Her main area of research interest includes pattern recognition, image processing, machine learning and web services. She is a life time member of the Indian

Society for Technical Education (ISTE), International Association of Computer Science and Information Technology, Singapore and International Association of Engineers – IAENG, Hong Kong. She has published several papers in well-known peer-reviewed journals.

1 Introduction

In the transformation process converting the raw data into considerable and substantial information to make the raw data useful, the vital element is the pattern. Though the method was initially developed to discover the interesting patterns for market basket analysis, later was extended to the tasks of discovering the regularities, interesting correlations, frequent pattern, hidden patterns, useful and understandable patterns between the variables of various other applications. Transforming raw data into knowledge is a process that becomes particularly significant for the organisations that make every effort to understand the behaviour of their customers. The transformation process assist the organisation to build a resilient economic benefit over their peers by proposing an improved and an exclusive personalised experience to the customers, which can both lead to retaining the customer-base and increased profit. The pattern denotes any type of consistency and uniformity, thus considered to be an earnest descriptor of crucial and vital properties of the data (Han et al., 2016; Robu et al., 2019). Many frequent item set algorithms have been recommended since the first approach was coined at the beginning of the 1990s (Agrawal et al., 1993). In the first approach, a level wise breadth first search (BFS) methodology was used to produce candidate item sets whose frequent count was calculated by scanning the dataset numerous times which is one for each size of the candidate item sets. The algorithms such as FP-growth (Han et al., 2000) and Eclat (Zaki, 2020), were later put forth which was based on the depth-first search (DFS) method where input dataset is represented based on the prefix-tree-based memory similarly known as vertical rearrangement of the dataset to work on a given transaction dataset. To date, many effective algorithms are available in the literature for resolving issues pertained to increased time complexity, memory requirements and search space pruning, related to frequent item sets. In many instances, the algorithms are appropriate enough for facing the complications (Aggarwal and Han, 2014) Nevertheless, as a concern related to large volumes of information that has to be analysed in the era of big data, traditional tactics have laid the practicalities aimed at innovative methodologies built on parallel programming environments bearing in mind both shared and distributed memory (Moens et al., 2013). To improve the performance and to handle parallel programming environment, several techniques for improvising the performance of parallel processing for partitioning the database is described in this paper. Measurements on performance on the different parallelised algorithms furthermore allowed us to find some bottlenecks and limitations, which can be circumvented in incorporating the new methods to resolve the same problem.

2 Related work

In general, the algorithms for FIM can be categorised into three main classifications that is pattern growth, join and tree-based (Aggarwal et al., 2014). While the join-based algorithms employ the bottom-up approach to find frequent items in a given dataset and raise them into larger item sets satisfying the minimum threshold value set by the user, set-enumeration concepts are incorporated by tree-based algorithms to solve the problems of frequent item set generation through building a lexicographic tree that mine the item sets either using the breadth-first or depth-first. The divide-and-conquer method to partition the databases and then developing them into extended ones in the databases is instrumented in the pattern growth algorithms (Robu et al., 2019).

While apriori, frequent pattern growth and Eclat have been the popular categories used in frequent item set mining (FIM) algorithms they also suffer from certain shortcomings. The apriori algorithm a join-based method uses horizontal data structure for storage and generates a large number of candidate item sets, thus resulting in multiple scans to the transactional database leading to high cost, memory usage and increased runtime. FP tree being a pattern growth algorithm though reduces the running time by constructing the FP tree and scanning the database only twice it becomes more cumbersome, difficult to build, expensive and difficult to fit in the shared memory when the database is large (Liao, 2015; Vijayarani and Sharmila, 2017). On the contrary Eclat, the tree-based algorithm key features uses a vertical database format associating each item set with a list a transactions where it appears, enumerates simple tied list intersections thus reducing the scans to the database to only once. Further using a lattice approach to decompose the search space into sub lattices and use prefix-based partition method enables processing to be performed independently in the main memory (Zaki, 2020). As quantity of the data remains to surge, Eclat also has problems such as time consuming intersection calculation, wider search space, frequent joining operations, larger memory utilisation, and with lowered mining efficiency in the process of mining frequent item sets (Zhang and Pei, 2016). With the data increasing in bulks in the fields of marketing as well in the fields of marketing, demand to maintain and improve the execution time, processing power, higher availability greater flexibility, and hence applications where task can be performed concurrently, share resources and able to synchronise the parallel data mining algorithms is suitable, however the issues related to structuring the tasks and preserving their sequence has always be an challenge (Moattar and Taharozi, 2018). This has attracted the researches to target parallel approaches to solve the problem of the era of big data. Among the different algorithms available apriori and Eclat has always proved to have a larger potential to apply the parallelisation schemes and hence become a major research interest among the practitioners.

The very first simple solution to parallelise the FIM algorithm was to incorporate multithread architectures. Though the support count was easy to be computed by distributing the data among the nodes (processors), memory requirements placed a concern that had to be solved (Moens et al., 2013). The apriori algorithm was first and foremost implemented on the multithread architecture (Zaki et al., 1997) where new level candidate sets were calculated by dividing the original dataset into various segments before analysing, later a different parallel version which could dynamically incorporate new item sets was coined, since nodes can calculate the support of potential frequent item sets, as they get further results from the other nodes, and make adjustments (Cheung et al., 1998). Later multiple local frequent pattern tree algorithms (parallel FP-growth)

was proposed which distributed the work among the processors unbiasedly (Zaïane et al., 2001). While GPUs were gaining interest, several novel GPU-based approaches were put forth, which had an advantage of speed and optical cost comparing with traditional parallel systems. In the year 2010, apriori-based GPU-FIM was proposed (Zhou et al., 2010) later which was extended to FP growth (Teodoro et al., 2010) as it outperformed the apriori like approaches (Han et al., 2016). Lastly, GPUs were employed to speed up the calculation of support count in FIM which always gave higher performance (Cano et al., 2013) irrespective of the amount of data records which overcame the complications of most time consuming phases (Ventura and Luna, 2016).

Nevertheless, multithread solutions suffer from memory requirements due to shared memory and hence are not scalable enough to apply to big FIM. Thus, distributed computing solutions have also been looked into where parallel programming architectures of both distributed memory and shared computing approaches the FIM applications for distributed systems are surveyed on. The MapReduce framework (Dean and Ghemawat, 2008) and Spark-based framework (Feng and Pan, 2019) was extensively incorporated by many researchers, eases the programming method for distributed data processing. In the later advancements, the MapReduce framework was presented to process data in parallel as small portions in distributed clusters (Dean and Ghemawat, 2008). The first FIM proposed was a version of the FP-growth algorithm, called PFP (Li et al., 2008), targeted on the distributed machines. Later many initial adaptations on single pass counting, fixed passes combined counting, dynamic passes counting of apriori on framework of MapReduce (Lin et al., 2012), but in 2008 truly efficient MapReduce algorithms for DistECLAT and BigFIM based on ECLAT algorithm and the in combination with apriori was coined (Moens et al., 2013). Progressively novel MapReduce methods for FIM were to cope up with the problems related to communication overhead, workload skewness and the intermediate data (Luna et al., 2018; Chon and Kim, 2018). However all these MapReduce, Spark frameworks suggestions appears to be new, it is significant to note that the important enhancements carried out are resultant from the architecture novelty than being breakthroughs in terms related to algorithm. The author suggested a Spark Parallel Eclat (SPEclat), a parallel Eclat algorithm based on Spark framework (Feng and Pan, 2019) which enhanced the data storage techniques to reduce the size of candidate sets. The data was grouped bestowing to the prefix that is used to split into different computing nodes to search space for compressed data. Though the method successfully partitions the data into multiple clusters in view of improving the processing, it can be further optimised by considering load on each individual cluster while partitioning computing nodes thereby reducing the data skew. RDD Eclat the different approaches to parallelise Eclat algorithm for Spark RDD framework was designed with various variants and different datasets was presented but was limited to 1 length-based prefixed method and improvements based on load balancing and better heuristics for equivalent class clustering was looked into (Singh et al., 2020). Looking at the diverse platforms, the interest of the researchers has focused on developing parallel algorithms targeting the data mining programming models. The current focus aims at parallelising the Eclat algorithms by applying various optimisation methods at different stages. Numerous parallel implementations of the Eclat algorithm have been developed for both shared memory and message-passing programming models (Chee et al., 2018; Zhang and Pei, 2016; Zaki et al., 1997; Cheung et al., 1998; Zaïane et al., 2001; Zhou et al., 2010; Dean and Ghemawat, 2008; Feng and Pan, 2019; Li et al.,

2008; Lin et al., 2012; Luna et al., 2018; Chon and Kim, 2018; Singh et al., 2020; Subbulakshmi and Deisy, 2018; Bhuvaneswari and Muneeswaran, 2021; Sjarif et al., 2021). The article is organised as follows: firstly, the preliminaries of FIM and the Eclat algorithm is presented. Next, our approach to parallelise bottom up Eclat algorithm using the python library is discussed. Then, some enhancements and conclusions on using the various heuristics are considered. The last part contains the result of our experiments and discussions.

3 Problem statement and preliminaries

Discovering frequent item sets has always been a crucial problem in data mining research and database applications. This problem is articulated as follows: specified a transaction database which is a set of transactions, discover all the frequent item sets where a frequent item set is one that occurs in at least a user-specified number or percentage of transactions, which means, its support is no less than the threshold otherwise known as minimum support. Eclat employs a upright database format, equivalence class clustering and bottom up lattice traversal which is explained in Figure 1 and the algorithm as proposed by Zaki (2020) is given in Figure 1 and the pictorial representation of horizontal transactional database converted to vertical representation in Tables 1 and 2, and calculating the candidate frequent items-based bottom up lattice-based tree traversal in Figure 2 and the resultant frequent item sets in Figure 3.

Figure 1 Bottom up Eclat algorithm (see online version for colours)

```
Algorithm: Eclat Bottom Up Approach
 Input: The transaction database (T) consisting of transactions (t1, t2, t3 .... tn) and
 corresponding  itemsets I where I = (i1, i2, i3 ,... in)
Output: Frequent Itemsets Fi
Convert Horizontal database to Vertical database
Call BottomUP(ECk)
Start Loop
            ECk = {Null}
             Start Loop
                    Iij = Ii U Ij
                T(Iij) = T(Ii) n T(Ij)
                 If Condition: | Iij | >= min_sup{
                            Eck + 1 = Eck + 1 U Iij
                            LECk = LECk U Aij;
                        }
              End Loop
            If Condition: Eck +1! = null
                Call BottomUP(ECk+1)
End Loop
Return LECk
```

Source: Zaki (2020)

FIM is a prevalent technique for deciding enthralling relations between sets of items in large databases. It forms the foundation of many applications, which perform people behaviour analysis, market prediction (Sjarif et al., 2021) or biological structures analysis. Diverse implementation of sequential algorithms has been proposed in view of

finding the frequent item sets which are both efficient and optimised. Most of them show some capabilities of parallel processing and should be evaluated while running them on many computational nodes. Due increase in the volume of data which has to processed by mining algorithm has impose a greater challenge in parallelisation of those implementation towards optimising the performance across multiple computation units. Though variety of frameworks like Hadoop, Spark, MapReduces ease the parallel process there is always a need for a parallelisation methods suitable for any type of parallel environment.

Table 1 Transactional dataset

Transaction ID (TID)	*Items*
A	1, 4, 5
B	1, 2, 3
C	2, 3, 5
D	1, 2, 5
E	1, 2, 3, 4, 5
F	3, 4, 5

Table 2 Vertical transition representation

Item sets	*TID*
1	A, B, D, E
2	B, C, D, E
3	B, D, E, F
4	A, E, F
5	A, C, D, E, F

Figure 2 Bottom up tree traversal (see online version for colours)

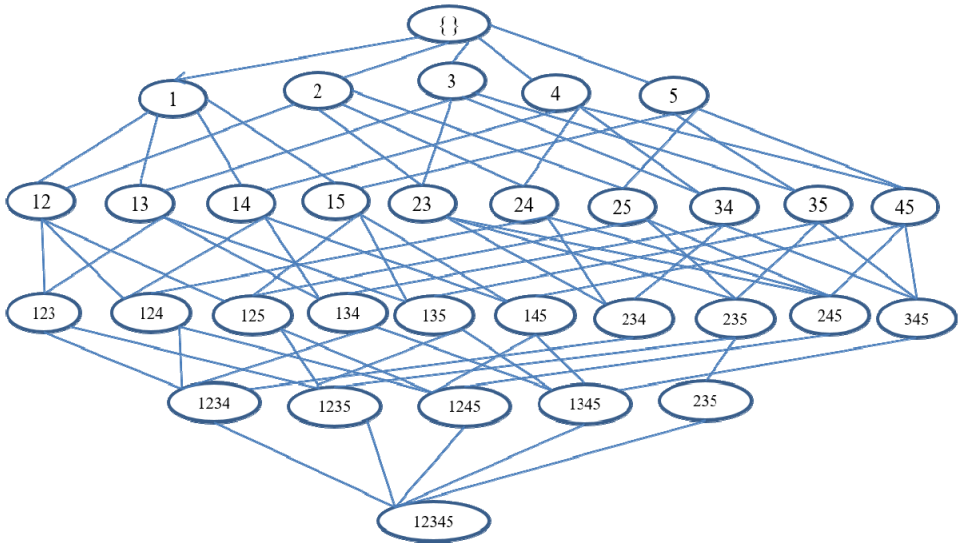


Figure 3 Frequent item sets

Minimum support = 3 (50%)	
Frequent 1 Item set	1, 2, 3, 4, 5
Frequent 2 Item set	(1, 2), (1, 5), (2, 3), (2, 5), (3, 5), (4, 5)

The approach discussed in the work showcases improvisation in performance towards parallelising the work across the different computing nodes which can adopted to all modern environment which supports parallelism by using techniques of default parallelism and hash partitioning. The implemented algorithm can be run on a single computer with multi-core processor as well as on a cluster of such machines. After examining the performance of the proposed methodology in efforts towards parallelised even though show great improvement, but still possess some challenges and weakness, which can be compromised by incorporating modern techniques to optimisation and load balancing approach.

4 Proposed work

RDD Eclat, the different approaches to parallelising Eclat algorithm in the form of variants were proposed for Spark RDD framework (Singh et al., 2020). In the Eclat variant the frequent item sets were calculated for a prefix length of 1, by partitioning the database using default parallelism that is equivalent to the number of computing units and later the dataset is partitioned using hash partitioner where the number of partitions was supplied by the user. The key observations were that the results were explored only for one length prefix-based equivalence classes and hence the proposed work has been based on discovering the results for $k \geq 2$ prefix-based equivalent classes as while applying the transaction filtering technique. The detailed algorithm is given below where the first step is to convert the horizontal database to vertical dataset and stored using hashmap data structure. The transaction filtering is applied before partitioning the database to avoid duplicate item sets and thus eliminating the duplicates before calculating the frequent item sets. In default parallelism the partitions are made based on $k - 1$, where k is the number of item sets, while this has the disadvantage of non-availability of required number of partitions and also the load balancing issues, while the hash partition reduces this overhead as the number of partitions can be set by the user and a hash function is applied on the values corresponding to the prefix of the equivalence classes which returns the remainder as the partition ID which significantly reduces the execution time.

---

*//**Converting horizontal database to vertical data***

Input: the transaction database

Read the input file

Create results set and initialise the counter

For each row in transactions

    Strip and split the items based on the delimiter

```
    For each item in row
        Check if item is in the result set
            Add to the transactions
        Else
            Create a new item in result set
            Update the transaction
    End loop
//Frequent item set extraction: default parallelism
Extract the key from the result set
For each item in keys
    Create a process and map item, suffix list, vertical dataset, result
    Update result in file
    Process creation:
    For each suffix in list // suffix of the current item
    Compute the intersection of transaction id, suffix item
    Compare the resultant is greater than or equal to minimum support
    If resultant is not empty && suffix is of length one
        Update the result with intersection set
    Else
        Compare the prefix of two keys (key[:–1] == suff[:–1])
            Compute the new key
            Update the new key & intersection set in result
    End loop
    Repeat for K + 1, K + 2, K + 3 ... till null frequent items
End loop
//Frequent item set extraction: hash partitioning
Extract the key from the result set
#create number of partition
hash_bin = [], partition = n
#parse the dataset for each key from the itemset
for each key, value from itemset;
    #using each value of the key compute the hash and assign to the corresponding bin
    for each v in value:
        compute the hash value(h)
    if h not in hash_bin.keys():
        hash_bin[h] = [key]
    else:
        hash_bin[h].append(key)
#for each bin create separate process and compute the frequent itemset and update
for each h in hash_bin:
```

```
    Create a process for each bin
    Update result in file
#for each suffix of the current item computes the common transaction and updates the result.
Process creation:
    For each suffix in list // suffix of the current Item
        Compute the intersection of transaction id, suffix item
        Compare the resultant is greater than or equal to minimum support
        If resultant is not empty && suffix is of length one
            Update the result with intersection set
        Else
            Compare the prefix of two keys (key[:–1] == suff[:–1])
                Compute the new key
                Update the new key & intersection set in result
        End loop
        Repeat for K + 1, K + 2, K + 3… till null frequent items
End loop
```

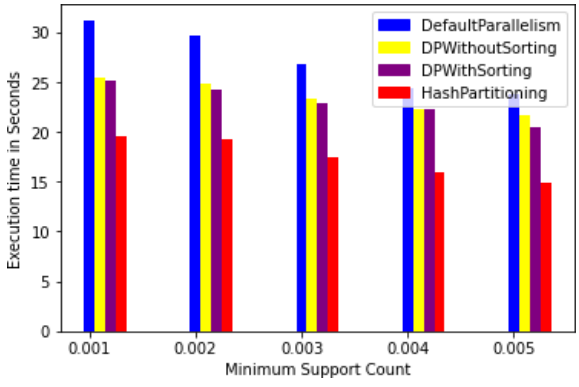
5 Experimental results

The source code of all the algorithms is written in Python. The different datasets used to conduct experiments are as follows: BMS_WebView-1 (Fournier-Viger et al., 2016) is a dataset of KDD CUP 2000 and contains a clickstream data from an e-commerce application. It has long sequences with more than 20 items which consist of real life clickstream data of an e-commerce. The total transactions sum up to 59,602, with 497 items and an average transaction width of 2.5. BMS_WebView_2 (Fournier-Viger et al., 2016) consists of real life clickstream data of an e-commerce with a total transaction count of 77,512 with 3,340 items and an average transaction width of 5. The results were also explored for the synthetic dataset (Frequent Itemet Mining Dataset Repository, http://fimi.ua.ac.be/data) T10I4D100K and T40I10D100K each of 1 lakh transactions and items counting to 870 of average transaction width 10, 1,000 of average transaction width 40.

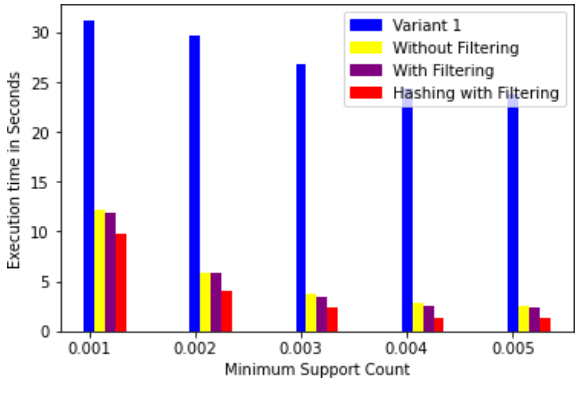
5.1 Processing time with respect to varied support count

Figures 4(a)–4(d) compares the execution time of the proposed algorithms while applying the partition heuristics of default parallelism and hash partition, in addition to applying transaction filtering and sorting the frequent sets. On all the datasets mentioned above the execution time difference between them becomes lesser with increased value of minimum support, and there is a notable reduction in time with respect to applying the hash partitioning heuristics for partitioning the equivalent classes.

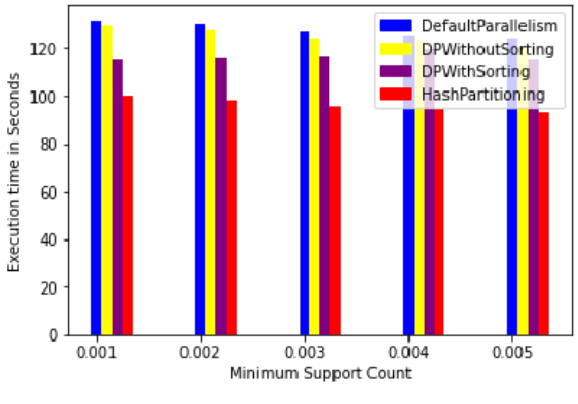
Figure 4 (a) Execution time of variants on dataset BMS_WebView_1 (b) Execution time of variants on dataset BMS_WebView_2 (c) Execution time of variants on dataset T10I4D100K (d) Execution time of variants on dataset T40I10D100K (see online version for colours)



(a)

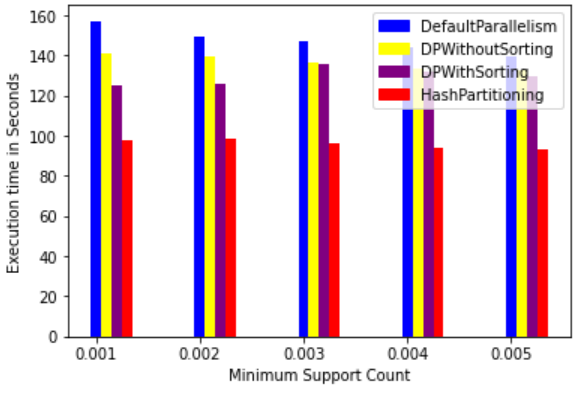


(b)



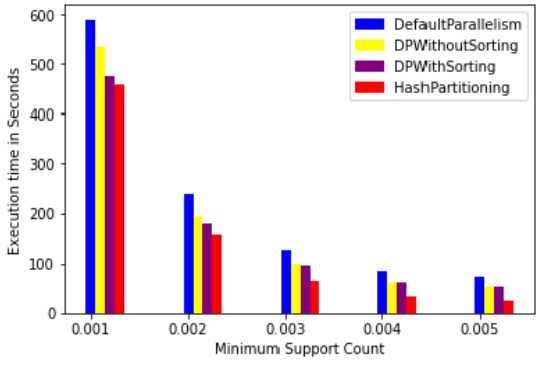
(c)

Figure 4 (a) Execution time of variants on dataset BMS_WebView_1 (b) Execution time of variants on dataset BMS_WebView_2 (c) Execution time of variants on dataset T10I4D100K (d) Execution time of variants on dataset T40I10D100K (see online version for colours)

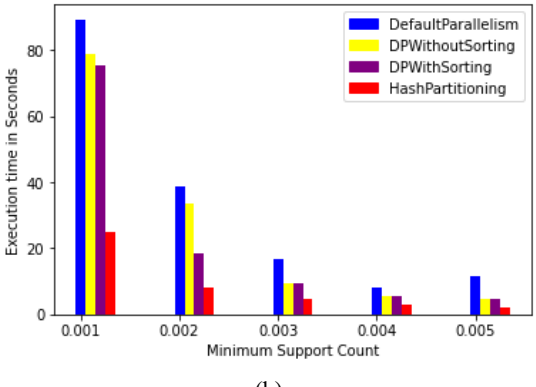


(d)

Figure 5 (a) Execution time for k length prefixes where $k \geq 2$ on dataset BMS_WebView_1 (b) Execution time for k length prefixes where $k \geq 2$ on dataset BMS_WebView_1 (see online version for colours)



(a)



(b)

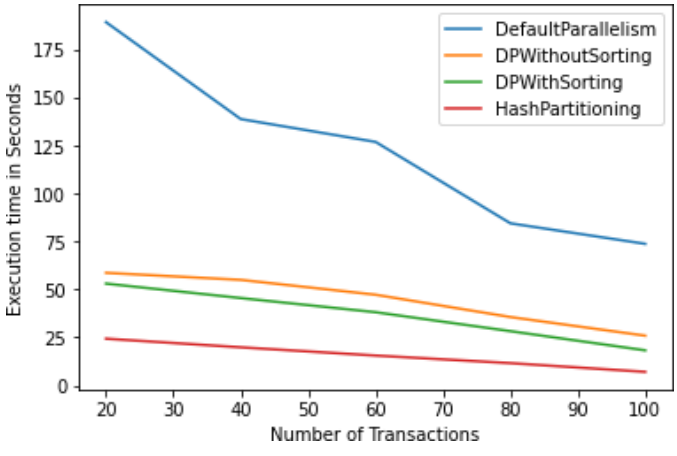
5.2 Execution time on k length prefixes where k ≥ 2

The proposed éclat variants are also applied for $k \geq 2$ prefix-based equivalence classes and the results are compared based on the execution time of the algorithm of all variants which is plotted in Figures 5(a)–5(b) where we can clearly see that the hash partitioned heuristics continue to perform better compared to other variants of default partition.

5.3 Execution time variations with increase in the number of transactions

The results were also compared for *k*-length prefixes where $k \geq 2$ for different datasets (Fournier-Viger et al., 2016) and shown in Figure 6 which indicates that by applying the hash partitioning the execution time can be significantly reduced when implement for $k \geq 2$.

Figure 6 Execution time variations for $k \geq 2$ prefix-based equivalence classes (see online version for colours)



5.4 Discussion

The Eclat algorithm for Spark framework (Singh et al., 2020) is redesigned to suit any parallel environment which is developed using python by following the heuristics of default parallelism and hash partitioning. In default parallelism the partitions are made based on $k - 1$, where k is the number of item sets, while this has the disadvantage of non-availability of required number of partitions and also the load balancing issues. By considering the limitations, the proposed approach partition the dataset using hashing approaches where the number partitions can be specified by the user which is either equivalent to computing units or random specified one. Hash partitioning is carried out by applying the hashing function to all the values corresponding to the prefix of equivalence which is supplied as the partition ID. The transaction filtering has been applied for both the heuristics which constantly reduces the execution time as the number of duplicates that could be generated otherwise is eliminated at every level and shows good results as it is applied for $k \geq 2$ prefix. Further by sorting the infrequent item sets at

every level will make sure that the non-beneficial item sets are eliminated at early stage and thus contributes to reduced combinations.

6 Conclusions and future work

The Eclat algorithm was redesigned for the distributed computing environment using Python which can be run without any complicated or specialised frameworks. The key contribution here is developing the hash partitioned algorithm using python for the distributed environments that can be used to apply for k-length prefixes where $k \geq 2$. We can also note that both the techniques of default parallelism where the equivalent classes were clustered based on $K - 1$, where k is the number of item sets and hash partitioning was applied and also verified with transaction filtering and with and without sorting infrequent items at every level. The algorithm has performed well and significantly reduces the execution time. The results were explored for 8 cores and the observation after analysing the performance has been presented. The performance can further be improved by either increasing the number of cores as well increasing the number of partitions specified by the user. The results are explored only for the ecommerce application and can be explored for other real time applications. The limitations of hash partitioning to improve the performance and also to generate more balanced distribution of equivalence classes will be the addressed in the future work.

References

Aggarwal, C.C. and Han, J. (2014) *Frequent Pattern Mining*, Springer International Publishing, Basel, Switzerland.

Aggarwal, C.C., Bhuiyan, M.A. and Hasan, M.A. (2014) 'Frequent pattern mining algorithms: a survey', in Aggarwal, C.C. and Han, J. (Eds.): *Frequent Pattern Mining*, pp.19–64, Springer, Basel.

Agrawal, R., Imielinski, T. and Swami, A.N. (1993) 'Mining association rules between sets of items in large databases', in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (ICDM '93)*, Washington, DC, pp.207–216.

Bhuvaneswari, M.S. and Muneeswaran, K. (2021) 'A parallel approach for web session identification to make recommendations efficient', *International Journal of Business Intelligence and Data Mining*, Vol. 19, No. 2, pp.189–213.

Cano, A., Luna, J.M. and Ventura, S. (2013) 'High performance evaluation of evolutionary-mined association rules on GPUs', *The Journal of Supercomputing*, Vol. 66, No. 3, pp.1438–1461.

Chee, C.H., Jaafar, J., Aziz, I.A. et al. (2019) 'Algorithms for frequent itemset mining: a literature review', *Artif. Intell. Rev.*, Vol. 52, pp.2603–2621 [online] https://doi.org/10.1007/s10462-018-9629-z.

Cheung, D.W., Hu, K. and Xia, S. (1998) 'Asynchronous parallel algorithm for mining association rules on a shared-memory multi-processors', in *Proceedings of the 10th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '98)*, Puerto Vallarta, Mexico, pp.279–288.

Chon, K. and Kim, M. (2018) 'Bigminer: a fast and scalable distributed frequent pattern miner for big data', *Cluster Computing*, Vol. 21, No. 3, pp.1507–1520.

Dean, J. and Ghemawat, S. (2008) 'MapReduce: simplified data processing on large clusters', *Communications of the ACM*, Vol. 51, No. 1, pp.107–113.

Feng, X.J. and Pan, X. (2019) 'Parallel Eclat algorithm based on Spark', *Applied Computer Research*, Vol. 36, pp.18–21.

Fournier-Viger, P., Lin, C.W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z. and Lam, H.T. (2016) 'The SPMF Open-source data mining library version 2', in *19th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2016) Part III, LNCS 9853*, Springer, pp.36–40.

Frequent Itemet Mining Dataset Repository [online] http://fimi.ua.ac.be/data.

Han, J., Kamber, M. and Pei, J. (2016) *Data Mining Concept and Technique*, 3rd ed., Morgan Kaufmann Publishers, imprint of Elsevier, 225 Wyman Street, Waltham, MA 02451, USA.

Han, J., Pei, J. and Yin, Y. (2000) 'Mining frequent patterns without candidate generation', *SIGMOD Record*, Vol. 29, No. 2, pp.1–12.

Li, H., Wang, Y., Zhang, D., Zhang, M. and Chang, E.Y (2008) 'Pfp: parallel FP-growth for query recommendation', in *Proceedings of the 2008 ACM Conference on Recommender Systems (RecSys '08)*, Lausanne, Switzerland, pp.107–114.

Liao, Y. (2015) *Research and Improvement of Association Rules Mining Based on Directed Graphs*, Southeast University.

Lin, M-Y., Lee, P.Y. and Hsueh, S-C. (2012) 'Apriori-based frequent item set mining algorithms on MapReduce', in *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication (ICUIMC '12)*, Kuala Lumpur, Malaysia, pp.1–8.

Luna, J.M., Padillo, F., Pechenizkiy, M. and Ventura, S. (2018) 'Apriori versions based on MapReduce for mining frequent patterns on big data', *IEEE Transactions on Cybernetics*, Vol. 48, No. 10, pp.2851–2865.

Moattar, M.H. and Taharozi, M. (2018) 'Parallel processing for data mining and data analysis applications', *Journal of Engineering and Applied Sciences*, Vol. 13, No. 5, pp.1228–1234.

Moens, S., Aksehirli, E. and Goethals, B. (2013) 'Frequent item set mining for big data', in *Proceedings of the 2013 IEEE International Conference on Big Data*, Santa Clara, CA, pp.111–118.

Robu, V. and Duarte dos Santos, V. (2019) 'Mining frequent patterns in data using apriori and Eclat: a comparison of the algorithm performance and association rule generation', *6th International Conference on Systems and Informatics*, IEEE.

Singh, P., Singh, S., Mishra, P.K. and Garg, R. (2020) 'RDD-Eclat: approaches to parallelize Eclat algorithm on Spark RDD framework', *Lecture Notes on Data Engineering and Communications Technologies book series (LNDECT*), January, Vol. 44.

Sjarif, N.N.A., Azmi, N.F.M., Yuhaniz, S.S. and Wong, D.H-T. (2021) 'A review of market basket analysis on business intelligence and data mining', *International Journal of Business Intelligence and Data Mining*, Vol. 18, No. 3, pp.383–394.

Subbulakshmi, B. and Deisy, C. 2018) 'An improved incremental algorithm for mining weighted class-association rules', *International Journal of Business Intelligence and Data Mining*, Vol. 13, Nos. 1/2/3, pp.291–308.

Teodoro, G., Mariano, N., Meira Jr., W. and Ferreira, R. (2010) 'Tree projection-based frequent item set mining on multicore CPUs and GPUs', in *Proceedings of the 22nd International Symposium on Computer Architecture and High Performance Computing*, pp.47–54.

Ventura, S. and Luna, J.M. (2016) *Pattern Mining with Evolutionary Algorithms*, Springer International Publishing, Basel, Switzerland.

Vijayarani, S. and Sharmila, S. (2017) 'Comparative analysis of association rule mining algorithms', *International Conference on Inventive Computation Technologies*, IEEE, pp.1–6.

Zaïane, O.R., El-Hajj, M. and Lu, P. (2001) 'Fast parallel association rule mining without candidacy generation', in *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM '01)*, San Jose, CA, pp.665–668.

Zaki, M.J. (2020) 'Scalable algorithms for association mining', *IEEE Transactions on Knowledge and Data Engineering*, June, Vol. 12, No. 3, pp.372–390.

Zaki, M.J., Parthasarathy, S., Ogihara, M. and Li, W. (1997) 'Parallel algorithms for discovery of association rules', *Data Mining and Knowledge Discovery*, Vol. 1, No. 4, pp.343–373.

Zhang, C. and Pei, L. (2016) 'Research and application of improved Eclat algorithm based on MapReduce', *Journal of Beijing Jiaotong University: Natural Science Edition*, Vol. 40, No. 3, pp.1–6.

Zhou, J., Yu, K. and Wu, B. (2010) 'Parallel frequent patterns mining algorithm on GPU', in *Proceedings of the 2010 IEEE International Conference on Systems, Man and Cybernetics*, Istanbul, Turkey, pp.435–440.