IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON RECENT ADVANCES IN CLOUD RADIO ACCESS NETWORKS

Received April 4, 2017, accepted April 17, 2017, date of publication April 27, 2017, date of current version June 7, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2699172

Enabling Far-Edge Analytics: Performance Profiling of Frequent Pattern Mining Algorithms

KHUBAIB AMJAD ALAM[1], RODINA AHMAD[1], AND KWANGMAN KO[2]

[1]Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur 50603, Malaysia
[2]Department of Computer Engineering, School of Computer and Information Engineering, Sangji University, Wonju 220-702, South Korea.

Corresponding authors: Khubaib Amjad Alam (khubaibalam@siswa.um.edu.my), Rodina Ahmad (rodina@um.edu.my), and Kwangman Ko(kkman@sangji@.ac.kr)

This work was supported by Bright Spark Unit, University of Malaya under Grant BSP-1632-13.

ABSTRACT

Far-edge analytics refers to the enablement of data mining algorithms in far-edge mobile devices that are part of mobile edge cloud computing (MECC) systems. Far-edge analytics enables data reduction in mobile environments, hence reducing the data transfer rate and bandwidth utilization cost for mobile-edge communication. In addition, far-edge analytics facilitates local knowledge availability to enable personalized mobile data stream mining applications. Existing literature mainly addresses classification and clustering problems in far-edge mobile devices, but the problem of frequent pattern mining (FPM) remains unexplored. This paper presents the results of an experimental study on the performance profiling of frequent pattern mining algorithms. We developed a real mobile application for performance analysis and profiling of 21 FPM algorithms with various real data sets in terms of execution time, storage complexity, sparsity, density, and data set size. According to the experimental results, large-sized data sets with high sparsity increase computational and storage cost in far-edge mobile devices. To address these issues, we propose a framework and discuss the relevant research challenges for seamless execution of FPM algorithms in MECC systems.

INDEX TERMS Association rules, data mining, far-edge analytics, frequent item sets, mobile cloud computing.

I. INTRODUCTION

Mobile edge cloud computing is an emerging research area in mobile cloud computing [1]. MECC systems facilitate the provision of conventional cloud computing services through edge servers that include cloudlets, micro data centers, and smart routers, amongst many others [2]–[7]. Edge servers provide networking, computing, and storage services at one-hop wireless distances from mobile devices, such as smartphones, wearable devices, mobile Internet of Things (IoTs), and body sensor networks [8], [9]. In the present study, these mobile devices are referred to as far-edge mobile devices. Edge servers facilitate minimizing the latency and prolonging the battery lifetime of far-edge mobile devices. However, continuous data transfers in edge servers increase bandwidth utilization cost and dependency on Internet connections. Alternately, the increasing computational power in far-edge mobile devices and close proximity of on-board data sources and computational resources like memory and CPU reduce latency. In addition, utilizing on-board computational resources in mobile devices allows the reduction of raw data transmission, hence minimizing bandwidth utilization costs. Far-edge analytics refers to the provision of knowledge discovery and data mining functionalities in mobile applications through far-edge mobile devices [10]. A far-edge analytics system, as depicted in Figure 1, is based on a mobile device such as a smartphone to collect and aggregate heterogeneous data from multiple on-board and off-board data sources [11], [12]. On-board data sources include sensing elements, e.g. cameras, accelerometers, compasses, microphones, touch screens, etc. Similarly, edge analytics systems gather data from device-resident files maintained for resource status information, application logs, Wi-Fi and Bluetooth scans, to name a few. Conversely, off-board sensing elements include a variety of wearable sensors that gather data to recognize ambulatory activities, monitor physiological bio-markers, and sense external environments. In this study, a smartphone-based far-edge analytic system is envisioned that provides maximum execution support for knowledge discovery and data mining operations using on-board computational resources.

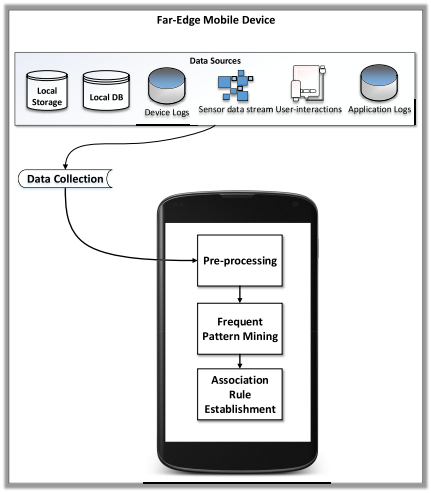


FIGURE 1. FPM in far-edge device.

This article contributes in multiple ways as follows. A detailed literature review is presented of selected FPM algorithms proposed for mining different types of frequent itemsets, including association rules, rare itemsets, closed itemsets, high utility itemsets, erasable itemsets, relevant and useful itemsets, minimal non-redundant association rules, top-k association rules, and top-k non-redundant association rules. Subsequently, an android mobile application is developed for performance profiling of the selected algorithms. Experiments were conducted with real datasets from the UCI repository and three mobile devices, including two smartphones and one tablet PC. In addition, the experimental results are analyzed in terms of dataset memory space consumption, execution time, sparsity, density, and size. We selected the highly compute intensive frequent pattern mining algorithms in order to put maximum computational load on mobile devices. Considering the results of batch data mining algorithms, we proposed a theoretical framework in section 5, which will not only support the stream mining data but also ensure the context-aware and adaptive execution of frequent pattern mining algorithms in mobile environments.

The key concerns relevant to FPM in far-edge mobile devices are articulated and a generic framework is proposed for future study in this important research area. A few highly related works are presented in Section 2. A detailed discussion of selected algorithms comprises Section 3 along with the data collection and experimental setup details. Section 4 entails results and discussion while Section 5 presents a generic framework for handling the revealed issues. The article concludes with Section 6.

II. RELATED WORK

Far-edge analytics systems are required to consider various resource constraints, such as battery power, on-board storage, memory, and the CPU for successful execution of knowledge discovery and data mining algorithms [12], [13]. These constraints hindered the exploitation of conventional data mining algorithms in earlier systems. Numerous studies focusing on far-edge analytic systems are presented in recent literature. For example, researchers have proposed StreamAR, which mines sensor data streams using a cluster-based classification approach [14]. In addition, Mobile WEKA, an android-based general data mining platform provides numerous clustering, classification, and association-rule mining algorithms [15]. Similarly, Open Mobile Miner (OMM) was developed as an adaptive data mining system that provides a library of light-weight data mining algorithms [16]. Moreover, CARDAP is an extension of OMM for deploying the context-aware real-time data analytics platform by integrating far-edge analytics with Fog cloud computing services [17].

To cope with the problem of limited resources, the Pocket Data Mining (PDM) framework was developed as an agent-based data mining framework [18]. PDM utilizes local devices to form a peer-to-peer network and execute data mining tasks collaboratively. Hence, the successful development of these far-edge analytics systems is evidence of the adoption of far-edge mobile devices as data mining platforms. Current far-edge analytics systems function either in adaptive mode, which enforces the compromise over knowledge pattern quality, or by enabling light-weight and domain-specific algorithms. Therefore, a need has emerged for generic far-edge analytics systems that handle conventional heavy-weight data mining algorithms.

Frequent pattern mining is a key research area in knowledge discovery and data mining. Formally, FPM is basically applied over I(set of items):$\{i_1, \ldots, i_n\}$ and T(set of transactions): $\{t_1, .., t_n\}$ where $T \subseteq I$ [19], [20]. Transaction ID (TID) is used to uniquely identify a transaction in database. T contains A(a set of items) iff $A \subseteq T$. The association rule (AR): $A \Rightarrow B$ over two itemsets A and B exists iff $A \subset I$ and $B \subset I$ and $A \cap B = \emptyset$. The rule $A \Rightarrow B$ contains the transactions with minimum support *minsup* for $A \Rightarrow B$ and confidence *minconf* for $A \cup B$. Moreover, for a given set of transactions D, the rule for minimum confidence (*minconf*) and minimum support (*minsup*) are specified by users and all rules that support *minconf* and *minsup* are generated for D resultantly. The itemsets and their ARs vary in simple, closed, maximal, rare, sporadic and utility based itemsets. Maximal frequent itemset as $M = \{I \in L | \nexists I' \in L, I \subset I'\}$, frequent closed itemset as $\text{FC} = \{C \subseteq I | C = h(C) \wedge support\ (C) \geq minsup\}$, and maximal closed frequent itemsets are defined as following: $\text{MC} = \{C \in FC | \nexists C' \exists FC, C \subset C'\}$. Here, h(C) represents closure of 'C' base on Galois (concept) connection [21]. Most of the FPM algorithms generate quality knowledge patterns when maximum amount of data is kept in memory. Keeping in view this constraint of FPM algorithms, we performed the performance profiling presented in this article.

III. METHODS

A. ALGORITHMS

A variety of FPM algorithms are proposed in literature for closed, maximal, rare, sporadic, and utility based items. Moreover, AR mining algorithms are also available to find

TABLE 1. Selected algorithms.

No.	Algorithm	Objective	Year	Authors
1	Apriori	ARs	1994	R. Agrawal and R. Srikant
2	AprioriTid	ARs	1994	R. Agrawal and R. Srikant
3	FP-Growth	ARs	2001	J. Han et al
4	Relim	Itemsets	2005	C. Borgelt
5	Eclat	Itemsets	2000	M. J. Zaki
6	dEclat	Itemsets	2003	M. J. Zaki and K. Gouda
7	Charm	ARs	2002	M. J. Zaki and C.J. Hsiao
8	DefMe	Itemsets	2014	A.Soulet and F. Rioult
9	Pascal	Itemsets	2000	Y. Bastide et al
10	Zart	Minimal non-redundant ARs	2006	L. Szathmary
11	AprioriRare	Rare Itemsets	2007	L. Szathmary
12	Apriori Inverse	Sporadic ARs	2005	YS Koh and N Rountree
13	U-Apriori	Itemsets	2007	CK Chui et al
14	Two-Phase	High Utility Itemsets	2005	Y. Liu et al
15	VME	Erasable Itemsets	2010	Z. Dong and X. Xu
16	MISApriori	ARs	1999	B. Liu et al
17	IGB	Relevant and Useful ARs	2004	Gh. Gasmi et al.
18	MNR	Minimal Non-redundant ARs	1998	M. Kryszkiewicz
19	AClose	Closed Itemsets	1999	N. Pasquier et al
20	TopKRules	Top-K ARs	2012	P. Fournier-Viger et al
21	TNR	Top-K Non-redundant ARs	2012	P. Fournier-Viger and VS. Tseng

the direct and indirect association between underlying itemsets. Keeping in view, the large variety of FPM algorithms, we carefully selected 21 classical algorithms, enlisted in Table- 1, for performance profiling in strictly mobile environments. The basic reason for selecting batch data mining approach is the lack of privision of dedicated memory in mobile deviceswhich is the major requirement of stream mining algorithms. This bottleneck hinders the applicability of mobile devices as data stream mining platforms. Another major reason for selecting batch data mining is to assess the worst case performance of mobile devices with maximum computing load which, if somehow deployed, are not promised by data stream mining algorithms as they work in forgetting mechanism and only operate using current data in hand. We selected Apriori and AprioriTid algorithms that are used for itemset mining at basic level and establish ARs later [19]. The mining process is: [$\forall$ large itemset A, {find non-empty subsets of A}] $\wedge$ [$\forall$ non-empty subset B, {output (rule): B $\rightarrow$ (A-B)} if the ratio [support (A)/support (B)] $\geq$ *minconf*].

Apriori and AprioriTid are multi-pass algorithms where candidate items satisfying *minsup* are found and processed iteratively to generate candidate itemsets. This iterative procedure continues until there is no large itemset found. Considering, the subset of any large itemset is also large, helps in joining large itemsets with $k-1$ items and deleting those subsets with no large itemsets and forming large itemsets with k-items.

Apriori algorithm works by counting items to determine large $1-itemsets$. Subsequently, it works in two phases. For example for pass k, $Apriori-gen$ function is used to generate candidate itemsets C_k, which works on input from $(k-1)_{th}$ pass using large itemset of $k-1$ i.e. L_{k-1}. Next the support of C_k is counted after a database scan; here C_k from a transaction are needed to be determined efficiently for fast counting. A subset function based on hash-tree is used to store C_k where leaf nodes of the tree contains itemsets list and internal nodes stores hash tables. The cost of keeping whole tree in memory is reduced by counting C'_{k+1} at k_{th} pass. This strategy works when cost of scanning database is more than cost of keeping in memory and counting additional C'_{k+1} - C_{k+1} candidates.

Alternately, AprioriTid is different than Apriori in accessing database for one time in counting minimum support after first pass. The candidate itemsets are encoded after first pass to reduce the size of data and reading efforts at later passes. AprioriTid generates $\overline{C}_k$ using Apriori-gen function before the pass begins but the support is counted without accessing database. The set $\overline{C}_k$ contains the items of the form $< Tid, \{I_k\} >$ where each I_k represents a large $k-itemset$ with transaction identifier (Tid). Another optimization of Apriori was proposed by introducing Pascal which is based on pattern counting inference [22]. The proposed scheme counts some of the frequent and infrequent items and determines the support of other items on the basis of obtained frequencies. This strategy reduced repetitive database scans.

The increasing computational cost of candidate generation in large itemsets especially with long patterns has led to the development of FP-growth algorithms [23]. The algorithm is based on frequent-pattern (FP) tree to store important information about frequent patterns in compressed form. FP-Growth algorithm performed efficiently due to three-step strategy: *a)* database is compressed and stored in $FP-tree$ to avoid multiple database scans at later passes, *b)* a pattern-frequent growth scheme is used to reduce the cost of candidate generation in long patterns, and *c)* a $divide-n-conquer$ approach is used to confine the search space and reduce the cost of level-wise search as used in Apriori algorithms.

Recursive Elimination (Relim) algorithm was inspired by FP-growth but works without prefix trees [24]. Here, the items are counted and compared with *minsup* value given by the user at first and then arranged in ascending order for

fast processing. It is to be noted that the core of Relim is a recursive function that has less computational cost than Apriori algorithms. The recursive function works by eliminating infrequent items from the transactions then selecting transactions containing least frequent items. Subsequently the least frequent items are deleted from selected transactions and the procedure starts again until there remains only frequent large itemsets.

The issues of multiple iterations, data-skew and complicated internal data structures led to poor data locality and additional computational and storage requirements. Eclat [25], resolved these issues using three strategies: *a)* used vertical $Tid - list$ format database for efficient enumeration, *b)* used lattice/sub-lattice approach for search space decomposition, and *c)* decoupled the pattern search from decomposition problem. Furthermore, each sub-lattice was enumerated using bottom-up search strategy. Eclat was effective for long itemsets. The scalability of Eclat is affected due to high memory requirements during intermediate results. A variant of Eclat, called dEclat, based on *diffsets*, was proposed to handle these issues [26]. The *diffsets* keep track of differences between class member's tidset and prefix tidset hence optimize intersection operation. The dEclat was not suitable for very long pattern mining due to intensive memory and computation requirement to keep *diffsets*.

We selected AClose for frequent closed itemset mining [21]. AClose uses the approach of limiting search space to closed itemset lattice, as an alternate of subset lattice. This approach helped in limiting AR without information loss and is very useful for densely correlated data. Although FC, found earlier, determines the exact frequency but the issue of traversing all itemsets efficiently was solved by Charm algorithm [27]. Charm used IT-tree (Itemset-Tidset tree) to explore both itemset and transaction space at the same time. The traversals were made using hybrid search method that identified FCs but skipped many levels as compared with earlier search methods which traversed at each level and all possible subsets. The algorithms also used a hash-based strategy to remove non-closed itemsets. Yet, Charm used *diffsets* to compute intermediate results but linear scalability was still an issue. This challenge of optimizing intermediate results was addressed by dCharm, which eliminated branches and established relationships among several *diffsets* for itemset growth [26]. Furthermore, we selected DefMe to mine minimal patterns that are another form of condensed patterns. DefME used critical-objects for fast minimality checking in depth-first search but redundancy is an issue that needs to be addressed.

Minimal non-redundant rules are lossless and informative. These rules are best choice for the rules with same information and same support. Zart, based on Pascal, was introduced to mine non-redundant ARs [28]. The algorithms works in three steps: *a)* finds frequent itemsets and marks frequent generators, *b)* filters FCs, and *c)* associates generators to their closures. The strength of Zart was its ability to track generators, which are the closed itemsets with same support but their subsets have different support.

Most of the literature covered frequent pattern mining techniques highlighting the need to investigate rare pattern mining algorithms that are very useful in some specific application areas. For example, health-related application to mine rare symptoms of a particular disease from Electronic Health Records (EHRs). The AprioriRare algorithm is selected to assess the performance of such algorithms in mobile environments [29]. The algorithm works in two steps: *a)* to traverse in frequent itemsets and *b)* enlist the rare itemsets. The algorithms have potential limitation of storing all rare itemsets which could be more challenging during establishment of ARs. Hence, the need for compact storage scheme arises.

The notion of finding rare items and frequent patterns simultaneously caused rare item problem which occurs due to two facts *a)* usually one *minsup* value is set for all itemsets by assuming that all frequent and rare itemsets comply with settled threshold which is a rare case in real world applications and *b) minsup* has to be set very low which cause large candidate sets and need exceptionally large memory. Hence, we selected MISApriori [30], which handle rare item problem by setting multiple *minsup* values to find frequent and rare itemsets at the same time. Moreover, the literature reports that CFPGrowth is efficient than MISApriori [31]. Yet it could be deployed in mobile environments but the issue is the input requirements that all itemsets should be ordered and minimum supports of each item should be explicitly defined in a separate file. These two bottlenecks restricted us to select this algorithm for automated analysis in our test application.

Another exceptional case is sporadic rules mining where some rules have very low support but have high confidence. The *minsup* needs to be set very low in Apriori algorithm to mine the sporadic rules. We selected AprioriInverse to handle such situation [32]. The algorithm works by setting maximum support (*maxsup*) in the place of *minsup* and ignores all itemsets above *maxsup* threshold. The algorithm is flexible enough to mine two kinds of sporadic rules: *a)* perfect sporadic rules with items support strictly below *maxsup* and *b)* imperfect sporadic rules with items having flexible support value that may be greater than *maxsup*. In some cases, AprioriInverse is not able to find imperfect sporadic rules.

The process of frequent itemset mining is totally dependent upon support values, which is not an issue in complete datasets with prior information. The definition of support values is challenging with uncertain datasets without prior information. Hence, expected support value is given to handle the probabilistic nature of the data. We selected U-Apriori which was proposed to find frequent itemsets from uncertain data [33]. The algorithm works differently by incrementing support count by the product of existential capabilities of all items $a \in A$ instead of incrementing by one as in the case of Apriori. Furthermore, the algorithm handles insignificant candidate support using trimming technique which trims all items with low existential probabilities hence the size of dataset is reduced.

High utility itemsets are another form of frequent patterns discovered to find the association between high-value (weightage) items. For example, in m-commerce settings, the items in a store that generate most revenue are considered to be high utility items. The two-phase algorithm addresses the similar issues [34]. The algorithm works by defining utilization weightage of each transaction in first phase and maintains downward closure property to find candidate sets with only combination of high utilization weightage items. Some low utility itemsets are also estimated at this phase but no itemsets is underestimated. In the second phase, an extra database scan is performed and overestimated itemsets are filtered out. The two-phase algorithm tends to be more costly than other frequent itemset mining algorithms because of multiple database scan and construction of high-utilization weightage model for each transaction. Yet, an efficient algorithm, FHM, is reported in algorithm but it is not in the scope of this study [35].

Conversely, erasable itemsets contain frequent items with low utility in specific scenarios. Mining and managing erasable itemsets can increase the efficiency of overall system. For example, mining of infrequent components and maintaining their stock in manufacturing industry is helpful in managing overall budget. We selected VME, as an alternate of META which is less efficient and scans database repeatedly [36], [37]. In addition, the inability to automatically prune irrelevant data is another weakness of META. The VME algorithm tracks the 'ids' of products in an itemset using PDI_list, which is a new data representation method. VME algorithm uses union operators on product 'ids' and prunes irrelevant data automatically.

In addition with itemset mining algorithms, we have selected six ARs mining algorithms that include FP-Growth, IGB, AprioriInverse, MNR, TopKRules, and TNR.

FP-Growth algorithms, as discussed earlier, uses *minsup* and *minconf* threshold to mine frequent itemsets and generate ARs. The usefulness and relevance of ARs become key attributes when mining frequent patterns from large datasets. We selected a two-step IGB (Informative and Generic Basis of Association Rules) ARs algorithm [38]. In first step, IGB finds closed itemsets and their associated generators using Zart algorithm and in second step the algorithm generates ARs. The IGB set of ARs is the set of associations of the form A $\rightarrow$ B-A, where A is a minimal generator of B, and B is a closed itemset having a support higher or equal to *minsup*, and the confidence of the rule is higher or equal to *minconf*.

Similarly, to find perfectly sporadic ARs, we selected AprioriInverse, discussed earlier [32]. AprioriInverse generates an AR with confidence $\geq$ *minconf* and support of non-empty subset of A$\cup$B is lower than *maxsup* value. AprioriInverse works in two steps *a)* perfectly sporadic itemsets are mined using *minsup* and *maxsup* values and *b)* perfectly sporadic ARs are generated in accordance with *minconf* applied over itemsets found in first step.

The size of underlying dataset plays a critical role in AR discovery. The problem of large ARs also exists in large datasets like large candidate itemsets and long frequent itemsets. The representative association rules (RR) were introduced to address this issue [39]. The RR represents a smallest set of rules that covers all ARs. Here, a cover operator was introduced to generate those ARs which are not RRs. The cover operator works without database scans. The selected algorithm, Minimal Non-redundant Rules (MNR), selects a compact and lossless set of minimal non-redundant ARs. It first finds closed itemsets and their associated generators using Zart algorithm. Then, it generates minimal non-redundant ARs.

Finally, we selected two algorithms, TopKRules and TNR, to mine Top-K ARs [40], [41]. The users set '*k*' value instead of *minsup* value and TopKRules algorithm generates top-k ARs resultantly. The algorithm generates top-k rules by setting *minsup*=0 and user-given *minconf* value. TopKRules increments *minsup* with support value of the lowest AR from top-k rules, found earlier, and prunes the search space. The algorithm traverses the new search space and repeats the process again. It sets the new *minsup* and prunes the search space. The process keeps-on repeating until there is no candidate rule remains for top-k ARs. Yet, the TopKRules algorithm is very effective but it may generate redundant ARs. Hence, we selected TNR that removes redundancy and returns non-redundant top-k ARs. The reported performance analysis show that TopKRules performed efficiently than TNR.

After carefully selecting the above algorithms, we selected some relevant datasets. The details of these datasets are presented in next subsection.

B. DATA COLLECTION

The collection of relevant datasets for performance profiling of studied algorithms is of prime importance. We have selected nine datasets related to Market Basket Analysis. The details of datasets and corresponding attributes (size, number of transactions (Tx), sparsity, density, minimum acceptable *minsup*, candidate counts, and itemset counts) with minimum *minsup* threshold are presented in Table- 2. The datasets are downloaded from sequential pattern mining framework [42] library and UCI machine learning repository [43]. The selected datasets are of two types: *a)* test datasets and *b)* real datasets. The test datasets are small sized datasets which are the reduced variants of some datasets used in the studied algorithms. Conversely, two real datasets, Mushroom and Retail are selected in this study. Mushroom is a real world dataset with 8124 transactions and retail contains 88163 transactions. The notion for selecting these datasets is to compare the performance of studied algorithm on medium and large-sized datasets.

It should be noted that we used ContextPasquier99 to test Apriori, AprioriTid, FP-Growth, Relim, Eclat, dEclat, A-Close, and Charm. Similarly, ContextZart to test DefMe, Pascal, Zart, AprioriRare, Closed ARs, and MNRs. While, UApriori was tested with ContextUncertain, two−phase with

TABLE 2. Selected datasets.

Dataset	Size	Tx	Sparsity	Density	Minsup	Cand. Counts	Itemsets Counts
CPasquier99	small	5	Medium	Medium	0	31	31
CZart	small	5	Medium	Medium	0	31	31
CInverse	small	5	Medium	Medium	0	31	31
CUncertain	small	4	Medium	Medium	0	31	31
DB_Utility	small	5	Medium	Medium	0	127	127
CVME	small	6	Medium	Medium	0	247	247
CIGB	small	6	Medium	Medium	0	31	31
Mushroom	small	8124	high	low	0.35	1382	1121
Retail	small	88163	low	high	0.05	16	16

DB_Utility, and VME with ContextVME. Moreover, ContextInverse was used to test AprioriInverse. Further, ContextIGB was used to test MISApriori, FP-Growth ARs, IGB, TopKRules, and TNR. Finally, all algorithms were tested using real datasets that are Mushroom and Retail as discussed earlier.

C. AIMS AND OBJECTIVES

- To investigate how does the size of datasets affects the performance of FPM algorithms in mobile devices?
- To find out how does the performance of FPM algorithms in mobile devices is affected by density and sparsity of different datasets?
- To find out how do the space requirements of FPM algorithms deviate in mobile devices?
- To determine how does the execution time of FPM algorithms deviate when tested with different datasets in mobile devices?

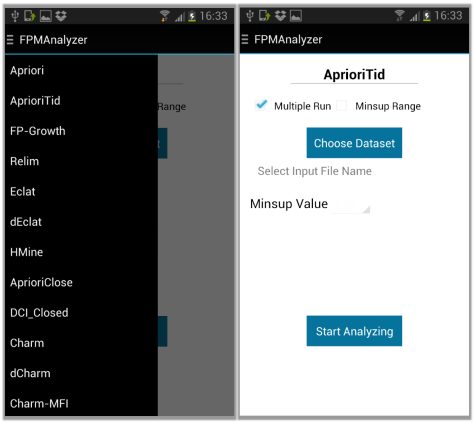


FIGURE 2. Screenshot of performance analyzer.

D. EXPERIMENTAL SETUP

The availability of devices in different configurations with fine-grained computing power is the key requirement to test the application in different environments. Hence, we selected three mobile devices (two smartphones and one tablet) including Samsung Galaxy S4, Lenovo A396, and Asus fonepad. The detailed specification of these devices is presented in Table- 3. We developed mobile application (see Figure 2) and tested it rigorously for the performance profiling of FPM algorithms in different mobile environments.

TABLE 3. Selected devices.

Specification	Samsung	Lenovo	ASUS
Model	GT-18552	A390	Fonepad
Processor	ARM-v7a	Dual Core 1GHz	Intel Atom Z2420
Storage	8GB	4GB	8GB
Available Memory	845MB	512MB	1GB
Operating System	Android 4.1.2	Android 4.0	Android 4.1

We prepared 1323 test cases [21 (algorithms) $\times$ 21 (input *minsup* combinations with interval of 0.05 in range of 0.00 and 1.00) $\times$ 3 (devices)] and each test case is repeated for five times to get the average value because the performance results of algorithms varied in each iteration. In addition, the *minconf* for all relevant AR mining algorithms was set as 0.50.

IV. RESULTS AND DISCUSSION

In this section, a set of experiments is designed to evaluate the impact of dataset size, sparsity and density on the performance of FPM algorithms. Similarly A set of experiments examines the deviation in the space requirements and execution time of FPM algorithms in mobile devices.

Firstly, the Impact of size of datasets on the performance of FPM algorithms is evaluated. Dataset size plays a critical role in far-edge analytic systems due to computational constraints. In this investigation, FPM algorithms when tested with small datasets executed seamlessly and provided maximum results. The effect of dataset size is studied in terms of four attributes: *a)* lowest *minsup* value supported by all FPM algorithms in the study, *b)* size of tree generated for traversals, *c)* number of candidate items counted during intermediate computations, and *d)* maximum itemsets counted on the least *minsup* threshold value. To limit the experimentation, the results were obtained from Apriori algorithm which is reported to be least efficient in terms of candidate generation, item-set counting, and tree size. The results of the experiment are depicted in Figure 3.

The experiments showed that all algorithms worked fine with small datasets when least *minsup* $= 0.0$ which itself is a good sign for adopting mobile devices as data mining platform for heavy-weight FPM algorithms. Another effect of datasets is the size of intermediate tree generated to search candidate items and itemsets. The tree size varied for different datasets. For example, the tree did not expanded with DB_Utility because utility value of each item

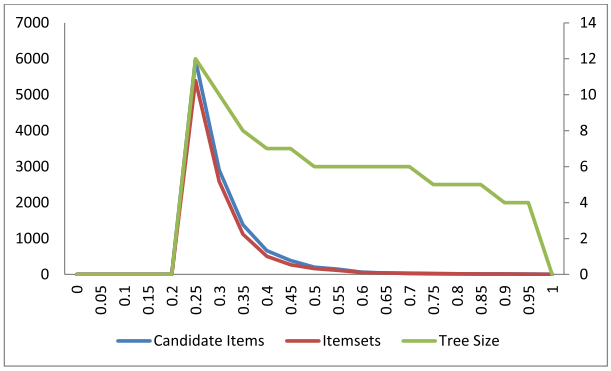


FIGURE 3. Effects of small scale datasets.

is already mentioned within each transaction hence reduced the need for candidate items. Alternately, it increased to 6 levels in case of ContextPasquier99, ContextZart, ContextUncertain, and ContextIGB and 12 levels when processing ContextVME. It could be concluded that nature of data is also important for seamless execution of FPM algorithms. In addition with this, maximum candidate items and itemsets are counted on the basis of *minsup* value. In this case, 31 candidate items were counted in ContextPasquier99, ContextZart, ContextUncertain, and ContextIGB. The results were different with DB_Utility and ContextVME as 127 and 247 items respectively. Finally the results for candidate itemsets remained similar as candidate counts i.e. 31 (ContextPasquier99, ContextZart, ContextUncertain, and ContextIGB), 127 (DB_Utility) and 247 (ContextVME).

Similarly, the experimental results, as presented in Table- 4, with medium-sized dataset like Mushroom exhibited good performance. Yet, maximum data is processing seamlessly but the mobile devices were needed to be compromised in some situations. For example, Samsung S4 supported seamless execution with least *minsup* = 0.35, Lenovno A396 worked best with least *minsup* = 0.30 and Asus fonepad supported FPM algorithms with least *minsup* = 0.25. Alternately, the number of candidate items and itemsets is huge due to underlying nature of dataset. It was noted that for all three devices the number of candidate items and corresponding itemsets are 5971 and 5393 for Samsung S4, 2904 and 2587 for Lenovo, and 1382 and 1121 for Asus fonepad.

Finally, for Retail dataset, most of the algorithms underperformed due to extensive memory and processor requirements to mine large datasets. In this case, eight algorithms managed to process the whole data successfully even with lesser *minsup* value i.e. 0.05 as compared to Mushroom with at least 0.25. These algorithms include Apriori, FPGrowth, Relim, Zart, AprioriRare, MNR, MISApriori, IGB. Similarly, candidate item (24) and itemsets (16) were lesser in this experiment.

Here, the question arises about the nature of dataset that even the algorithms completely processing the dataset with exceptionally large number of transactions is producing such

TABLE 4. Effects of medium and large size datasets.

Parameter	Samsung S4	Lenovo A396	ASUS Fonepad
Mushroom Dataset			
Least Minsup	0.35	0.30	0.25
Tree Size	08	10	12
Candidate Items	1382	2904	5971
Itemsets	1121	2587	5393
Retail Dataset			
Least Minsup	0.05	0.05	0.05
Tree Size	04	04	04
Candidate Items	24	24	24
Itemsets	16	16	16

TABLE 5. Density and sparsity of datasets.

Dataset	N	n	d	s
ContextPasquier99	25	16	64%	36%
Mushroom	1018874	193586	19%	81%
Retail	1146179	908576	79.27%	20.73%

small amount of candidate items and itemsets. Hence, it is perceived that there could be some other underlying characteristics of dataset that can directly affect the overall results.

Second experiment analyzes the Impact of density and sparsity on the performance of FPM algorithms in mobile devices. The investigations were made to find the effect of density and sparsity on overall results of different datasets. Three datasets ContextPasquier99, Mushroom and Retail were selected in this case. ContextPasquier99 has 5 attributes and 5 transactions, Mushroom has 22 attributes and 8124 transactions, and Retail contains 88163 transactions with 6 attributes.

Generally, density represents the percentage of populated cells and sparsity represents the percentage of non-populated cells in the dataset. We measured the density and sparsity using Equation- 1 and Equation- 2 where d and s represents density and sparsity receptively. In addition, the dataset contains total N possible items and n populated items. The results of these calculations are presented in Table- 5.

$$d = (n/N) \times 100 \tag{1}$$

$$s = 100 - d \tag{2}$$

The effects of density and sparsity were measured in terms of *a)* tree size, *b)* number of candidate items and *c)* number of itemsets.

Firstly, the tree size in ContextPasquier99 as depicted in Figure- 4 remained small as the tree growth went to 6 levels with least *minsup* (i.e. 0.0) and 3 levels with most *minsup* (i.e. 0.80). The growth trend reveals that tree-size initially remained higher but start decreasing when *minsup* values were increased. Similarly, the number of candidate items and resultant itemsets shown the same trend. Maximum number of items and itemsets discovered using ContextPasquier99 remained 31 with least *minsup* (0.0). Alternately, minimum number of items and itemsets remained 6 and 4 respectively with most *minsup* (0.80). The overall effect on tree-size, items and itemsets with different *minsup* values can be witnessed in Figure- 4.

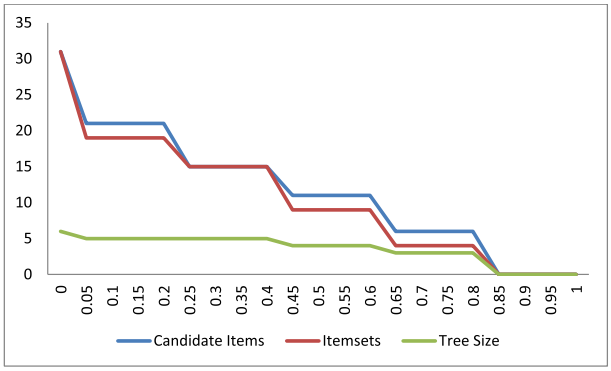


FIGURE 4. Effect of density and sparsity - CP99.

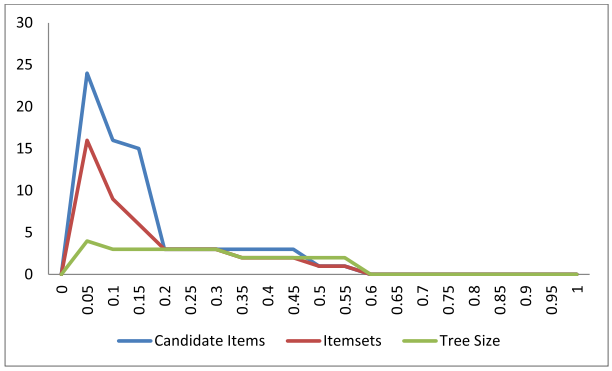


FIGURE 6. Effect of density and sparsity - retail.

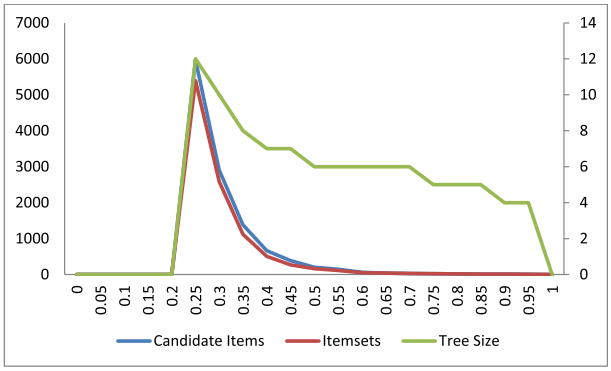


FIGURE 5. Effect of density and sparsity - mushroom.

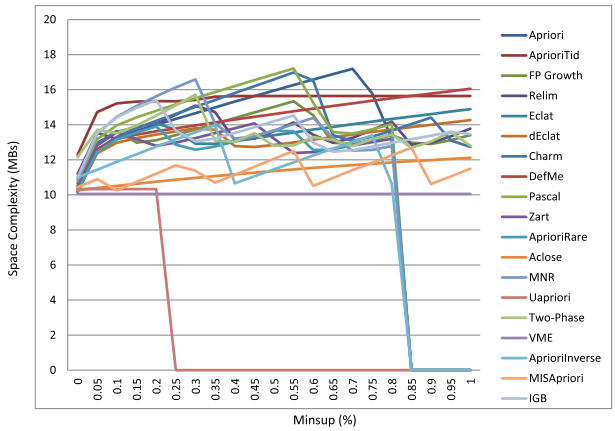


FIGURE 7. Space complexity - small dataset.

Secondly, the experiment with Mushroom revealed that the algorithms started discovering items and itemsets when least *minsup* was 0.25. Initially, the tree size, plotted on secondary-axis in Figure- 5, grown up to 12 levels because of sparse nature (i.e. 81 %) and a large range of attributes (i.e. 22). The tree size gradually started decreasing with average growth up to 6 levels when *minsup* values increased and it came to 4 levels when most *minsup* was 0.95. Similarly, the discovered items and itemsets were 5971 and 5393 when least *minsup* was settled as 0.25 but on average 791 items and 689 itemsets were discovered using Mushroom. Moreover, the discovered items and itemsets with most *minsup* i.e. 0.95 were reported as 7. Further, the growth trend of all three parameters could be witnessed in Figure- 5.

Finally, experiment with retail dataset revealed that tree size grown up to 4 levels with least *minsup* $=$ 0.05 but decreased up to 2 levels when most *minsup* was settled as 0.55. The apparent effect of high density (i.e. 79.27%) and minimum number of attributes (i.e. 6) could be witnessed when average growth remained between 2 to 3 levels. Similarly, candidate items and itemsets remained on average 6.8, and 4.36 respectively. Alternately, the results reported with most *minsup* (i.e. 0.55) showed that tree size grown up to 2 levels while only 2 candidate items and itemsets discovered. The overall parametric trend of experiment with retail dataset is shown in Figure- 6.

After carefully studying the outcomes of all three experiments, we can infer that sparsity and density of the dataset significantly impact the performance of FPM algorithms when deployed in mobile environments.

Third experiment examines the deviation in space requirements of FPM algorithms in mobile devices. The importance of space complexity increased due to limited storage and memory resources in mobile devices. The intermediate result generation and internal data structure of algorithms may lead to erroneous execution of data mining algorithms. The experiments were performed to uncover the space consumption trends of FPM algorithms in mobile environments. Again, the tests were performed over small, medium and large-sized datasets. The acquired space consumption trends are depicted in Figures- 7, - 8 and - 9.

The results of small-sized datasets witnessed that most of the algorithms consumed 13$\pm$3 MB of average disk space with maximum offset of 726KB during complete execution of algorithms. Hence, this small variation in each iteration limited us to consider the average space consumption. Alternately, seven multi-scan algorithms behaved little different and stopped consuming the space when there was no candidate items. These include Apriori, Pascal, Zart, AprioriRare, MNR, UApriori, and AprioriInverse. Rest of the algorithms

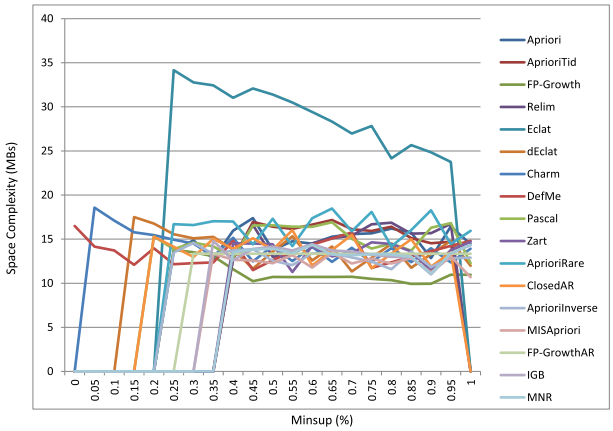


FIGURE 8. Space complexity - medium dataset.

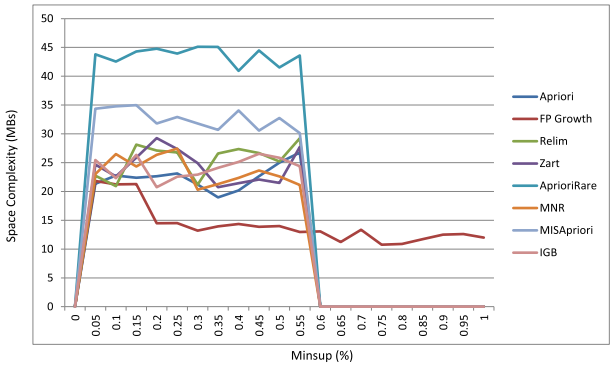


FIGURE 9. Space complexity - large dataset.

consumed space for all *minsup* values because of one-time candidate generation to get rid of repetitive database scans.

Space complexity of medium-sized dataset (i.e. Mushroom) is different than small-sized datasets. The relatively larger amount of data and candidate generation requirements of the algorithms restricted most of the algorithms to find candidate items and itemsets at small *minsup* values. The performance analysis reveals that all algorithms worked best with least threshold representing 40% *minsup*. Yet, the average space consumption of all algorithms with 40% and above *minsup* range remained between 15±3 MB of average disk space but FPGrowth and Eclat varied in performance. Here, FPGrowth consumed less space on average 12±2MB and Eclat consumed 29±4 MB of average disk space. It should be noted that the overall offset in individual experiments remained less than 1MB hence the average values were rounded off to nearest integer for easy representation and interpretation.

Alternately, due to overall effect of sparsity, the results with less than 40% *minsup* values were varied. Here, Charm is the only algorithm that discovered itemsets even when least *minsup* was 0% but in this case it consumed relatively large disk space i.e. 46.21MB. Similarly DefME consumed 14.15MB at 5%, Eclat consumed 15.5MB at 15%, ClosedAR consumed 15.27MB at 20%, and FPGrowthAR consumed 13.35MB of disk space with 30% *minsup* threshold. In addition five algorithms Apriori, FPGrowth, Eclat, AprioriRare, and AprioriInverse consumed 13.6MB, 13.8MB, 34.2MB, 16.7MB, and 13.5MB respectively with 25% least *minsup* threshold. It was observed that four algorithms including AprioriTid, Relim, Zart, and MNR did not run with least *minsup* below 40% threshold.

Further analysis of large-sized datasets as depicted in Figure- 9 gives a different view. The dataset was more dense hence the algorithms consumed resources with least *minsup* threshold of 5%. In addition, the data distribution of underlying dataset enabled to discover itemsets within the range of 5% (least *minsup*) and 55% (most *minsup*). The average space consumption of all algorithms varied a little. The detailed analysis are summarized in Table- 6 which gives the detailed information about the range of space consumption and maximum offset during each experiment set. In addition, working *minsup* range where algorithms consumed memory also highlighted in Table- 6.

TABLE 6. Space consumption of FPM algorithms using retail dataset.

Algorithm	Average apace consumed	Maximum offset	Minsup range (%)
Apriori	22±4	440KB	5-55
FPGrowth	12±2	783KB	0-100
Relim	25±4	98KB	5-55
Zart	24±3	472KB	5-55
AprioriRare	43±2	113KB	5-55
MNR	23±4	834KB	5-55
MISApriori	32±1.5	1.2MB	5-55
IGB	24±1	335KB	5-55

The overall analysis of large-sized dataset showed that space consumption for five algorithms remained 24±3MB with an average maximum offset of 436KB. These five algorithms include Apriori, Relim, Zart, MNR, and IGB. The minimum average space was consumed by FPGrowth which was 12±2MB with an offset of 783KB. Alternately, two algorithms MISApriori and AprioriRare consumed 32±1.5MB and 43±2MB respectively. In addition it was noted that FPGrowth was still consuming resources even there were no-candidate itemsets. Hence, it is perceived that multiple database scans lead towards memory efficiency which is an essential requirement in mobile devices. Yet, the multiple database scans could lead towards the optimal performance but the effect on overall time complexity is a prime issue. Hence further investigations were made to find this effect.

Finally, the last experiment examines the Deviation in execution time of FPM algorithms in mobile devices. Limited memory and storage space are prime considerations for the deployment of data mining algorithms in mobile environments. Considering these issues, formal analysis of average execution time of all algorithms were made. Again the results were reported for small, medium, and large-sized datasets. It should be noted that results shown in Figures- 10, - 11 and - 12 are presented with different time-scale. Here the execution time for small datasets is presented in milliseconds (ms). Alternately the execution time of medium and large-sized datasets is measured in seconds(s).

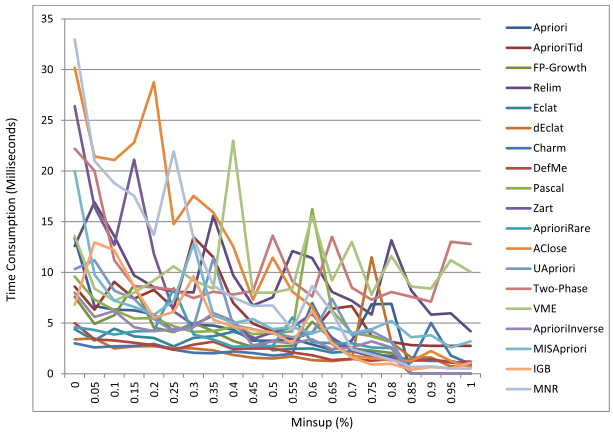


FIGURE 10. Execution Time- small dataset.

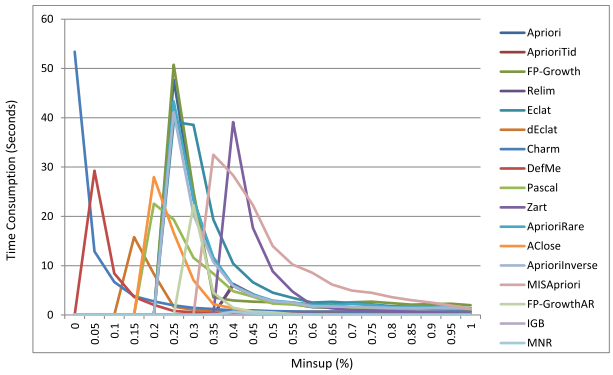


FIGURE 11. Execution Time - medium dataset.

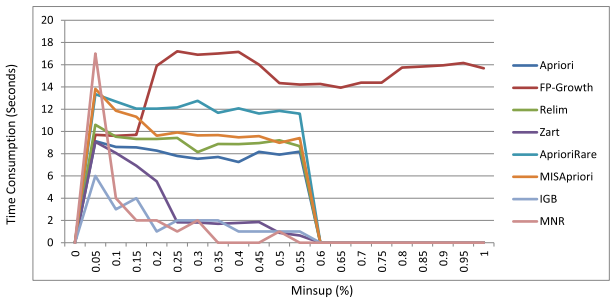


FIGURE 12. Execution Time - large dataset.

The experimental results of small datasets denote that all algorithms executed seamlessly without any extra time requirements. The trend shows that maximum average execution time was 33ms which is also negligible. Overall average execution time of all algorithms remained 5.7ms. Most of the algorithm took average 6±3ms which does not effect the overall performance of algorithms. The details of average execution time of all algorithms are presented in Table- 7 below. In addition, maximum offset from average execution time is also presented to asses the time consumption trade-off.

Alternately, data size and extensive memory requirements for intermediate computations prolongs the overall execution time in medium and large-sized datasets. In case of Mushroom dataset, all executed algorithms except Eclat, Charm, and Two-phase took 5±3 seconds with an average maximum offset of 3 seconds. The average execution time of Eclat, Charm, Two-Phase, and MISApriori algorithms remained 11.25, 6.91, 27.71, and 29.59 seconds respectively. The results showed that Two-Phase performed worse in this case with average execution time of 27.71 and offset 16.32 seconds. It should be noted that VME and UApriori failed to execute with Mushroom dataset because of different nature of dataset.

In addition, when executed with large dataset most of the algorithms failed to be executed. But the average execution time of running algorithms remained 5±3 seconds with an average maximum offset of 3 seconds. The results as shown in Table- 7 depicts that although the average execution time is seamless but the size of dataset affected the overall performance of mobile devices.

A. RECOMMENDATIONS

The significance of mobile devices as data mining platforms is attributed to intermediate data generation, input dataset size, and search strategies in FPM algorithms. These attributes affect the overall time and space complexity of FPM algorithms. In view of the performance profiling results, it can be concluded that far-edge mobile devices act as data mining platforms for small and medium-sized datasets, but they necessitate some basic parameter tuning for large-sized datasets. Hence, three parameter tuning strategies are proposed to meet the challenge of large-sized dataset mining: a) chunked data analysis, b) context-based knowledge discovery, and c) periodic data analysis. The optimal strategy must ensure seamless maximum data processing with minimal latency. Once divided into manageable data chunks, large datasets may be very useful in this regard. The data mining application needs to index all data chunks and integrate the resultant pattern intelligently to aggregate the final results. Formally, a large dataset D could be equally divided into N number of partitions P and could be mathematically presented in Equation- 3 and- 4.

$$D = \sum_{i=1}^{n} P_i \tag{3}$$

where each

$$P_i = \frac{D}{N} \tag{4}$$

In this case, the data management overhead may prolong the overall execution time, but it is perceived that very large datasets can be harnessed easily using this approach. The on-board sensors in mobile devices allow gathering different contextual information relating to locations, activities, device usage, and device (dis)charging patterns. The scheduling of data mining tasks based on contextual information may become very handy to accomplish the seamless execution of FPM algorithms in mobile devices. For example, location-related information can enable recognizing frequently visited places like the home and office. Such location information

TABLE 7. Average execution time of FPM algorithms.

Algorithm	Small-size datasets		Mushroom		Retail	
	Time (ms)	Offset (ms)	Time (s)	Offset (s)	Time (s)	Offset (s)
Apriori	5.22	3.10	6.81	1.76	8.10	3.39
AprioriTid	5.99	2.58	2.20	0.97	failed	
FPGrowth	4.21	0.59	6.91	2.46	14.72	5.98
Relim	9.48	6.75	2.28	0.54	9.17	3.63
Eclat	2.71	0.98	11.25	3.45	failed	
dEclat	2.49	0.67	2.01	1.32	failed	
Charm	2.55	2.12	9.22	1.45	failed	
DefMe	2.28	0.67	2.38	3.38	failed	
Pascal	5.72	4.65	6.91	6.89	failed	
Zart	8.10	2.45	5.99	3.52	3.63	1.54
AprioriRare	3.81	4.30	6.88	3.84	12.16	6.34
AClose	11.02	6.48	8.51	3.49	failed	
UApriori	6.00	6.2	failed		failed	
Two-Phase	10.41	2.12	27.71	16.32	failed	
VME	10.35	7.89	failed		failed	
AprioriInverse	4.23	3.30	6.31	1.58	failed	
MISApriori	6.19	4.29	29.59	4.13	10.29	3.11
IGB	4.69	3.46	0.02	0.575	2.18	1.33
MNR	9.34	4.67	0.10	0.665	2.63	1.3

coupled with device-usage patterns may be convenient for scheduling data mining tasks when the device is not being utilized. Hence, constructing a model that can predict suitable times for scheduling data mining tasks can have a significant role in this matter. For example, the model can predict when a person is busy having a meal, sleeping, working in the office, driving on a highway, or exercising in a local park. In addition, information about when a phone is charging, locked, or not in use can also be very helpful for devising a data mining strategy for mobile devices. Finally, the data mining tasks can be scheduled periodically by taking dataset portions without considering contextual information.

V. A FRAMEWORK FOR FREQUENT PATTERN MINING IN MECC SYSTEMS

Considering the mentioned recommendations, it is perceived that maximum FPM algorithm execution can be easily performed locally without the need for remote data processing systems like servers and cloud services. However, we propose a framework for frequent pattern mining in MECC environments (Figure 13). This framework facilitates six types of operations, namely data stream acquisition, data preprocessing, data fusion, data management, frequent pattern mining and summarization, and context-aware computation offloading.

A. DATA STREAM ACQUISITION

The profiling performance results revealed that dataset size plays a vital role in the seamless execution of FPM algorithms. Therefore, the input dataset volume needs to be handled carefully. In case of live data streams, the velocity of incoming data is another key consideration. To handle the volume and velocity aspects, the datasets and data streams are divided into equal-sized data chunks and traditional sliding window-based methods are used to traverse the data files. The sizes of the sliding windows vary according to the FPM algorithms selected and the required minsup and minconf threshold values. However, sliding window expansion yields better results, as most algorithms prefer keeping maximum data in memory to avoid frequent database scans and input file reads.

B. DATA PREPROCESSING AND DATA FUSION

Dataset density and sparsity are also critical in the performance maximization of FPM algorithms, as discussed in section 4. In addition, live sensor data collection may introduce noisy data streams due to false sensor calibrations (e.g. radio signal interference from other devices or systems in the surrounding) and improper sensor placement (e.g. sensor position, number, and orientation). Therefore, each dataset and/or data stream is preprocessed separately to obtain useful, noise-free, and highly-dense data chunks.

C. DATA FUSION

Data fusion strategies enable integrating preprocessed data chunks from multiple data sources. However, the strategies vary according to the algorithms nature and mobile application functionalities. A thorough analysis revealed that the data structure and data processing behavior of FPM algorithms significantly impact FPM algorithm performance. Multi-scan algorithms like AprioriTid are memory efficient but compute-intensive. Conversely, Apriori is a single-scan algorithm with fewer computational requirements but more memory requirements. Therefore, data fusion strategies are designed keeping in view the processing behavior and internal data structures of FPM algorithms.

The second main consideration for data fusion is the core functionalities provided by mobile applications, i.e. type of data source, like static datasets for batch processing or live data streams for stream processing, and data they produce (structured, unstructured, or semi-structured). Because FPM algorithms may need to be re-configured to accommodate the required data types, the variability in live data streams is another factor that needs attention.

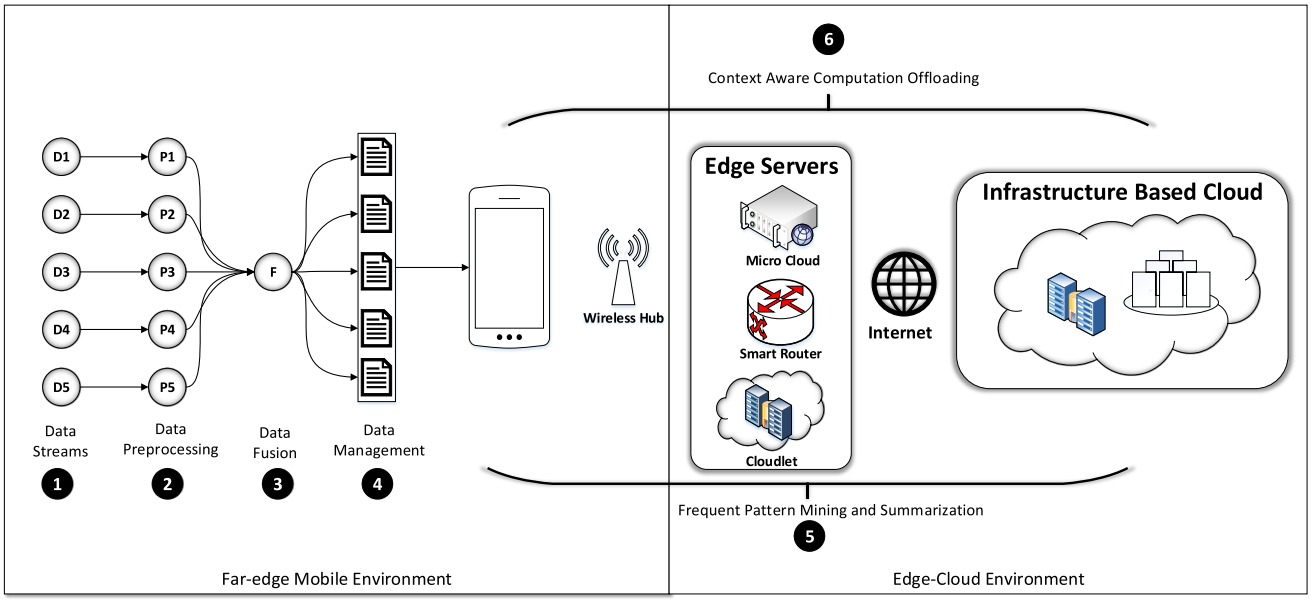


FIGURE 13. Proposed Framework for FPM in MECC System.

The variability factor emerges owing to the diverse sampling rates of different sensors in mobile applications. An example is a mobile application that finds the association among frequent physical activities and locations visited by a mobile user. The accelerometer for activity recognition is usually a sample of over 25 readings per second. However, the users location does not change so frequently, so data fusion from these two data sources is a challenging task. The complexity gradually increases as additional data sources are involved in mobile applications.

D. DATA MANAGEMENT

MECC systems involve massive heterogeneity in terms of mobile device processing power, bandwidth availability for different communication interfaces, and Internet connection availability for communications between far-edge mobile devices and edge servers. Therefore, the optimal execution of real-time mobile applications becomes an NP-hard problem. The data management of datasets and data streams in transient data stores helps cope with this problem. The data streams are stored in multiple data files and processed whenever adequate resources are available either in far-edge mobile devices or in edge-cloud environments.

E. FREQUENT PATTERN MINING AND SUMMARIZATION

The framework supports FPM algorithm execution in three modes. Far-edge mobile devices serve as a primary platform for data mining; however, due to the limited resource availability and high computational requirements of some FPM algorithms, the data files are processed in either edge servers or back-end cloud infrastructures. The multi-stage processing of FPM algorithms requires efficient summarization strategies for pattern aggregation and overall knowledge management. Therefore, the pattern summarization operations are performed in a back-end cloud environment and knowledge synchronization is performed between far-edge mobile devices and edge-cloud infrastructures.

F. CONTEXT AWARE COMPUTATION OFFLOADING

The framework is designed to support context-aware computation offloading for the seamless execution of FPM algorithms in MECC systems. To achieve optimal load-balancing and dynamic offloading, deep-context models must find the fine-grained situations and determine the adequate execution mode for FPM algorithms. The context model assists with determining the correlation between far-edge mobile devices and their resource availability, usage behavior, (dis)charging patterns and movement patterns, the sequential patterns of Internet connection availability, the maximum size of data chunks, and the execution histories of FPM algorithms. Although context management in MECC systems helps in devising optimal execution strategies, the design of such context model is challenging. So far, we carried out the performance profiling of FPM algorithms, articulated the relevant challenges and proposed a framework to address the challenges discovered in this significant research area. However, the proposed framework still needs further evaluation.

VI. CONCLUSION

Research on far-edge analytics is still in its early stages. Deploying data mining algorithms in far-edge devices, such as smartphones, wearable devices, wireless body area networks and mobile IoTs can help reduce massive data streams that are being transmitted to cloud data centers. This data reduction in far-edge mobile devices lowers the data transfer and bandwidth utilization costs in MECC systems. How-

ever, the performance profiling of FPM algorithms revealed that the size and sparsity of large datasets hamper the performance of far-edge mobile devices. To address these problems, we proposed three performance tuning strategies and a framework for the execution of FPM algorithms in MECC systems. In future, we will continue this research by developing real-world case studies for the proposed framework. The main focus of our research will be on performing real-time analytics on mobile streaming data in MECC systems.

REFERENCES

[1] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the Internet of Things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, Aug. 2016.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[3] M. H. Rehman, C. S. Liew, T. Y. Wah, and M. K. Khan, "Towards next-generation heterogeneous mobile data stream mining applications: Opportunities, challenges, and future research directions," *J. Netw. Comput. Appl.*, vol. 79, pp. 1–24, Feb. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804516302995

[4] G. I. Klas. (2015). *Fog Computing and Mobile Edge Cloud Gain Momentum Open Fog Consortium, Etsi Mec and Cloudlets*. [Online]. Available: http://yucianga.info/?p=938

[5] T. H. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, and L. Sun. (2015). "Fog computing: Focusing on mobile users at the edge." [Online]. Available: https://arxiv.org/abs/1502.01815

[6] K. Ha and M. Satyanarayanan, "Openstack++ for cloudlet deployment," School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-15-123, 2015.

[7] K. A. Alam and R. Ahmad, "A hybrid fuzzy multi-criteria decision model for cloud service selection and importance degree of component services in service compositions," in *Proc. 12th Int. FLINS Conf. Uncertainty Modelling Knowl. Eng. Decision Making (FLINS)*, vol. 10. 2016, p. 334.

[8] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya. (2016). "iFogSim: A toolkit for modeling and simulation of resource management techniques in Internet of Things, edge and fog computing environments." [Online]. Available: https://arxiv.org/abs/1606.02007

[9] H. Dubey, J. Yang, N. Constant, A. M. Amiri, Q. Yang, and K. Makodiya, "Fog data: Enhancing telehealth big data through fog computing," in *Proc. ASE BigData SocialInform.*, 2015, p. 14.

[10] M. H. ur Rehman, C. S. Liew, T. Y. Wah, J. Shuja, and B. Daghighi, "Mining personal data using smartphones and wearable devices: A survey," *Sensors*, vol. 15, no. 2, pp. 4430–4469, 2015.

[11] O. D. Lara and M. A. Labrador, "A survey on human activity recognition using wearable sensors," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 3, pp. 1192–1209, 3rd Quart., 2013.

[12] M. H. ur Rehman, C. S. Liew, and T. Y. Wah, "UniMiner: Towards a unified framework for data mining," in *Proc. 4th World Congr. Inf. Commun. Technol. (WICT)*, Dec. 2014, pp. 134–139.

[13] M. M. Gaber, J. Gama, S. Krishnaswamy, J. B. Gomes, and F. Stahl, "Data stream mining in ubiquitous environments: State-of-the-art and current directions," *Wiley Interdiscipl. Rev., Data Mining Knowl. Discovery*, vol. 4, no. 2, pp. 116–138, Mar./Apr. 2014.

[14] Z. S. Abdallah, M. M. Gaber, B. Srinivasan, and S. Krishnaswamy, "StreamAR: Incremental and active learning with evolving sensory data for activity recognition," in *Proc. IEEE 24th Int. Conf. Tools Artif. Intell. (ICTAI)*, vol. 1. Nov. 2012, pp. 1163–1170.

[15] P. Liu, Y. Chen, W. Tang, and Q. Yue, "Mobile WEKA as data mining tool on Android," in *Advances in Electrical Engineering and Automation*. Heidelberg, Germany: Springer, 2012, pp. 75–80.

[16] S. Krishnaswamy *et al.*, "Open mobile miner: A toolkit for mobile data stream mining," in *Proc. ACM KDD*, Jun. 2009. [Online]. Available: https://researchportal.port.ac.uk/portal/en/publications/open-mobile-miner-a-toolkit-for-mobile-data-stream-mining(1790663c-6c6b-4b9a-9e7e-7c224b31173a)/export.html

[17] P. P. Jayaraman, J. B. Gomes, H. L. Nguyen, Z. S. Abdallah, S. Krishnaswamy, and A. Zaslavsky, "CARDAP: A scalable energy-efficient context aware distributed mobile data analytics platform for the fog," in *Advances in Databases and Information Systems*. Cham, Switzerland: Springer, 2014, pp. 192–206.

[18] F. Stahl, M. M. Gaber, M. Bramer, and P. S. Yu, "Pocket data mining: Towards collaborative data mining in mobile computing environments," in *Proc. 22nd IEEE Int. Conf. Tools Artif. Intell. (ICTAI)*, vol. 2. Oct. 2010, pp. 323–330.

[19] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases (VLDB)*, vol. 1215, 1994, pp. 487–499.

[20] M. H. ur Rehman, C. S. Liew, and T. Y. Wah, "Frequent pattern mining in mobile devices: A feasibility study," in *Proc. 6th Int. Conf. Inf. Technol. Multimedia*, Nov. 2014, pp. 351–356.

[21] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," in *Database Theory—ICDT*. Heidelberg, Germany: Springer, 1999, pp. 398–416.

[22] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal, "Mining frequent patterns with counting inference," *ACM SIGKDD Explorations Newslett.*, vol. 2, no. 2, pp. 66–75, 2000.

[23] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Mining Knowl. Discovery*, vol. 8, no. 1, pp. 53–87, 2004.

[24] C. Borgelt, "Keeping things simple: Finding frequent item sets by recursive elimination," in *Proc. 1st Int. Workshop Open Source Data Mining, Freq. Pattern Mining Implement.*, 2005, pp. 66–70.

[25] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 372–390, May 2000.

[26] M. J. Zaki and K. Gouda, "Fast vertical mining using diffsets," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2003, pp. 326–335.

[27] M. J. Zaki and C.-J. Hsiao, "CHARM: An efficient algorithm for closed itemset mining," in *Proc. SDM*, vol. 2. 2002, pp. 457–473.

[28] L. Szathmary, A. Napoli, and S. O. Kuznetsov, "ZART: A multifunctional itemset mining algorithm," Res. Rep., 2006.

[29] L. Szathmary, A. Napoli, and P. Valtchev, "Towards rare itemset mining," in *Proc. 19th IEEE Int. Conf. Tools Artif. Intell. (ICTAI)*, vol. 1. Oct. 2007, pp. 305–312.

[30] B. Liu, W. Hsu, and Y. Ma, "Mining association rules with multiple minimum supports," in *Proc. 5th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 1999, pp. 337–341.

[31] Y.-H. Hu and Y.-L. Chen, "Mining association rules with multiple minimum supports: A new mining algorithm and a support tuning mechanism," *Decision Support Syst.*, vol. 42, no. 1, pp. 1–24, Oct. 2006.

[32] Y. S. Koh and N. Rountree, "Finding sporadic rules using apriori-inverse," in *Advances in Knowledge Discovery and Data Mining*. Heidelberg, Germany: Springer, 2005, pp. 97–106.

[33] C.-K. Chui, B. Kao, and E. Hung, "Mining frequent itemsets from uncertain data," in *Advances in Knowledge Discovery and Data Mining*. Berlin, Germany: Springer, 2007, pp. 47–58.

[34] Y. Liu, W.-K. Liao, and A. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," in *Advances in Knowledge Discovery and Data Mining*. Heidelberg, Germany: Springer, 2005, pp. 689–695.

[35] P. Fournier-Viger, C.-W. Wu, S. Zida, and V. S. Tseng, "FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning," in *Foundations of Intelligent Systems*. Cham, Switzerland: Springer, 2014, pp. 83–92.

[36] Z. Deng and X. A. I. A. E. I. Xu, "An efficient algorithm for mining erasable itemsets," in *Advanced Data Mining and Applications*. Heidelberg, Germany: Springer, 2010, pp. 214–225.

[37] Z.-H. Deng, G.-D. Fang, Z.-H. Wang, and X.-R. Xu, "Mining erasable itemsets," in *Proc. Int. Conf. Mach. Learn. Cybern.*, vol. 1. Jul. 2009, pp. 67–73.

[38] G. Gasmi, S. B. Yahia, E. M. Nguifo, and Y. Slimani, "A new informative generic base of association rules," in *Advances in Knowledge Discovery and Data Mining*. Heidelberg, Germany: Springer, 2005, pp. 81–90.

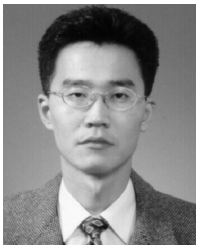
[39] M. Kryszkiewicz, "Representative association rules and minimum condition maximum consequence association rules," in *Principles of Data Mining and Knowledge Discovery*. Heidelberg, Germany: Springer, 1998, pp. 361–369.

[40] P. Fournier-Viger, C.-W. Wu, and V. S. Tseng, "Mining top-K association rules," in *Advances in Artificial Intelligence*. Heidelberg, Germany: Springer, 2012, pp. 61–73.

[41] P. Fournier-Viger and V. S. Tseng, "Mining top-K non-redundant association rules," in *Foundations of Intelligent Systems*. Heidelberg, Germany: Springer, 2012, pp. 31–40.
[42] P. Fournier-Viger. (2011). *SPMF: A Sequential Pattern Mining Framework*. [Online]. Available: http://www.philippe-fournier-viger.com/spmf
[43] K. Bache and M. Lichman. (2013). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml

KHUBAIB AMJAD ALAM received the master's degree in computer science from the Comsats Institute of Information Technology, Wah Cantt, Pakistan, in 2013. He is currently pursuing the Ph.D. degree with the Faculty of Computer Science and Information Technology, University of Malaya, Malaysia, under the prestigious Bright Spark Scheme. He has authored several articles in high impact journals and reputed conferences, including the Information Systems, Quality and Quantity, Energy, IJIM, ECM, CloudCom, and FLINS. His current research interests include services and cloud computing, non-functional requirements, enterprise systems, software evolution, decision support systems, knowledge engineering, and applications of soft computing methodologies. He is a member of the ACM SIGSOFT, the IEEE TCSE and TCBIS, and the International Society on MCDM. He has served as an Evaluator for numerous SCI-E indexed journals and international conferences, including the IEEE Access, KBS, IJITDM, IJWSR, ICMNEE, and FIT.

RODINA AHMAD received the master's degree in computer science from the Rensselaer Polytechnic, USA, and the Ph.D. degree in information system management from the National University of Malaysia. She is currently an Associate Professor with the Faculty of Computer Science and Information Technology, University of Malaya. She has over 20 years of experience revolving around teaching, researching, supervising, advising, and consulting in universities and governmental agencies. She has a wealth of academic and cross industry experience in the areas of enterprise analysis and modeling, software requirements engineering, and computer assisted education for computer programming. Programming languages analysis and software development management are her passion. Her research interests are software process improvement, intelligent tutoring system for programming, enterprise analysis, services computing, global software development, computer assisted learning, and Software Requirements Engineering. She received the Outstanding Academic Grant for her studies that include the National Petronas Scholarship and the Sultan Iskandar Scholarship.



KWANGMAN KO received the B.Sc. degree from Wonkwang University in 1991, and the M.Sc. and Ph.D. degrees in computer engineering from Dongguk University, South Korea, in 1993 and 1998, respectively. He is currently a Professor with the Department of Computer Engineering, School of Computer and Information Engineering, Sangji University, South Korea. His research interests include the re-targetable tool suite for embedded systems, virtual machines, and architecture description language.