applied sciences

MDPI

Article

TKIFRPM: A Novel Approach for Topmost-K Identical Frequent Regular Patterns Mining from Incremental Datasets

Saif Ur Rehman [1], Muhammad Altaf Khan [2], Habib Un Nabi [3], Shaukat Ali [1], Noha Alnazzawi [4] and Shafiullah Khan [2,5,*]

1 Department of Computer Science, University of Peshawar, Peshawar 25120, Pakistan
2 Institute of Computing, Kohat University of Science and Technology, Kohat 26000, Pakistan
3 Computer Science Department, Degree College Upper Dir, Dir 18500, Pakistan
4 Department of Computer Science and Engineering, Yanbu Industrial College, Royal Commission for Jubail and Yanbu, Yanbu Industrial City 41912, Saudi Arabia
5 Faculty of Computer and Software Engineering, Huaiyin Institute of Technology, Huai'an 233003, China
* Correspondence: skkust@gmail.com

Abstract: The regular frequent pattern mining (RFPM) approaches are aimed to discover the itemsets with significant frequency and regular occurrence behavior in a dataset. However, these approaches mainly suffer from the following two issues: (1) setting the frequency threshold parameter for the discovery of regular frequent patterns technique is not an easy task because of its dependency on the characteristics of a dataset, and (2) RFPM approaches are designed to mine patterns from the static datasets and are not able to mine dynamic datasets. This paper aims to solve these two issues by proposing a novel top-K identical frequent regular patterns mining (TKIFRPM) approach to function on online datasets. The TKIFRPM maintains a novel synopsis data structure with item support index tables (ISI-tables) to keep summarized information about online committed transactions and dataset updates. The mining operation can discover top-K regular frequent patterns from online data stored in the ISI-tables. The TKIFRPM explores the search space in recursive depth-first order and applies a novel progressive node's sub-tree pruning strategy to rapidly eliminate a complete infrequent sub-tree from the search space. The TKIFRPM is compared with the MTKPP approach, and it found that it outperforms its counterpart in terms of runtime and memory usage to produce designated topmost-K frequent regular pattern mining on the datasets following incremental updates.

Keywords: frequent patterns; regular frequent patterns; data mining; algorithm

Citation: Rehman, S.U.; Khan, M.A.; Nabi, H.U.; Ali, S.; Alnazzawi, N.; Khan, S. TKIFRPM: A Novel Approach for Topmost-K Identical Frequent Regular Patterns Mining from Incremental Datasets. *Appl. Sci.* **2023**, *13*, 654. https://doi.org/10.3390/app13010654

Academic Editor: Giacomo Fiumara

Received: 30 November 2022
Revised: 27 December 2022
Accepted: 30 December 2022
Published: 3 January 2023

1. Introduction

The seminal work for the frequent itemsets (FIs) discovery is named as the Apriori algorithm [1], and its successor procedures are defined to find frequently occurring items in the datasets qualifying user-tuned support threshold parameter. The support threshold is a fundamental parameter which is needed by the FIs' discovery processes before the initiation of mining operations. However, it is a challenging task to select a suitable support threshold parameter for a dataset depending on the characteristics of the dataset [2,3]. Moreover, it is not clear to users how many numbers of patterns could be returned in response to a particular support threshold parameter [4]. The classical frequent itemsets mining (FIM) methods could return many redundant or/and insignificant patterns as itemsets if a low support threshold parameter is provided by users. In addition, providing a low support threshold parameter could result in less pruning of itemsets and high support threshold parameter could prune large number of itemsets [5]. Therefore, several interest-based pattern mining techniques have been proposed to cut down the size of the resultant itemset [5–11].

One of the criterions for the interestingness of the patterns is to consider the periodicity or regularity of the patterns along with their frequency [7]. The periodicity or regularity of a pattern present the occurrence behavior of a pattern (i.e., whether it is found regularly, irregularly, or if it is present in a particular time interval in a given dataset) [7,12]. A pattern is termed as a periodic frequent pattern if its frequency is greater than the support threshold parameter and its periodicity is less than the maximum periodicity [7]. The discovery of periodic frequent patterns is useful in many applications. For example, the analysis of a customer's purchase behavior patterns can result in a better marketing strategy [13] and analysis of the Web users' regular access patterns can help in efficiently browsing a particular website [6]. Similarly, the regular and frequent stock market indices patterns can be pertinent for companies and individuals for future investment [7,14,15]. A variety of methods are proposed by the researchers, motivated by the importance and applications of periodic frequent patterns, such as mining the approximate periodic frequent patterns [11], mining rare periodic frequent patterns [8], using efficient approaches to mine periodic frequent patterns from transactional datasets [9], periodic frequent pattern mining from big data [10], and periodic frequent pattern mining with maximum items support constraints [16]. Moreover, periodicity of patterns is noticed by the data mining researchers in sequential pattern mining, such as discovering periodic high-utility itemsets in a discrete sequences [17] and an efficient mining of the periodic high-utility sequential patterns [18]. However, all methods need support threshold parameter to be defined by users, and the mining operation could produce large number of patterns, most of which would not be of any interest to a user.

To address the issues, a simple approach could be to ask a user about the number of topmost frequent regular patterns, which are required initially in place of a support threshold. Following this notion, a number of top-K frequent regular pattern mining methods are introduced in [2,11,12,19]. All the top-K frequent regular patterns classically perform breadth-first traversal of the search space to generate the required patterns. The top-K frequent regular patterns maintain a top-K list to record the topmost-K frequent regular patterns in descending support order during exploration of a search space. In top-K frequent regular pattern mining, different support calculation procedures are adopted, such as the top-K frequent regular pattern transaction identifiers (TIDs) intersection method [2], compressed TIDs [12,19], using dataset partitioning and support estimation [19,20], and a partitioned dynamic bit vector [21]. However, the available methods have the following limitations:

(a) The top-K frequent regular pattern mining methods adopt a best-fit strategy for the patterns that are radically using the Apriori method for pattern generation. The top-K frequent regular pattern mining methods perform excessive candidate pattern generation, which is inessential for computing. These techniques use no pruning methods to prune the unnecessary candidates before counting the supports of frequent regular itemsets.

(b) The top-K frequent regular patterns are applied on offline-datasets. However, the datasets are online nowadays, and are continuously growing and remain in the updating stage. Therefore, it is mandatory to have an online synopsis or summary data structure to maintain the consolidated information of an already present dataset portion and the current dataset chunk retrieved so far.

To solve the problems, this research work has proposed an algorithm, namely top-K identical frequent regular pattern mining (TKIFRPM). The TKIFRPM produces top-K identical frequent regular patterns (FRPs) from the online incremental datasets. The TKIFRPM maintains a novel data structure with item support index tables (ISI-tables) to keep summarized information about the updated dataset. Subsequently, the mining operation can be launched at any time interval to discover the top-K regular frequent patterns from the ISI-tables. The TKIFRPM explores the search space (arranged in itemset enumeration tree) in recursive depth-first order. The TKIFRPM applies a novel progressive node's sub-tree pruning strategy to rapidly eliminate a complete infrequent sub-tree from the search space. Hence, the TKIFRPM is computationally efficient as compared to the best-fit methods of top-K frequent regular pattern mining, since the TKIFRPM method does not perform support checking on the nodes of the infrequent subtrees unlike other top-K frequent regular pattern mining methods. The TKIFRPM is compared with a state-of-the-art relevant MTKPP approach in terms of runtime (i.e., execution) and memory usage. The approaches are evaluated on three different dataset (i.e., mushroom, accident, and chess) and the performance gain on each dataset is recorded. In all experimental scenarios, the TKIFRPM has outperformed MTKPP in terms of low runtime and memory usage. The proposed TKIFRPM algorithm could have several real-time applications, including web mining, stream mining, real-time fraud detection, and stock exchange marketing.

The paper is organized in the following sections. Section 2 presents a detailed review of the available frequent regular pattern approaches and highlights their advantages and disadvantages. Section 3 discusses the preliminaries required for the top-K identical frequent regular pattern mining. The Section 4 presents details of the methodology and approach used in the proposed TKIFRPM algorithm. The experimental results are presented in Section 5. In the last section, the conclusion of this research and potential future work are presented.

2. Literature Review

This section provides a comprehensive overview of the methods proposed by the researchers for the topmost frequent regular pattern mining algorithms. However, these methods suffer from several issues. Firstly, these methods are computationally intensive because they follow the classical Apriori approach for the required pattern mining. Secondly, the methods are inadequate because they are applicable on static datasets and have no synopsis structure available to find patterns from online datasets. Thirdly, the K semantics of a parameter where all the topmost regular frequent patterns does not consider the distinct K support count as topmost-K frequent patterns; rather, if the topmost-K patterns with the same support are found, it stops mining further distinct K topmost-K support patterns. Fourthly, none of these methods apply pruning methods, other than the least support threshold from the top-K list. A detailed review of the topmost-K regular frequent pattern mining research can be found in [15]. However, a top-level comparison of the available relevant methods, along with the proposed method, is presented in Table 1 and discussed in the following paragraphs.

Table 1. Top-level comparison of the topmost frequent regular pattern mining methods.

S. No	Method Name	Support Threshold/Top-K	Dataset (Online/Offline)	Summary Data Structure
1	MTKKP [2]	Top-K	Offline	No
2	TR-CT [12]	Top-K	Offline	No
3	TKRIMPE [19]	Top-K	Offline	No
4	TFRC-Mine [21]	Top-K	Offline	No
5	PF-ECLAT [22]	Support threshold	Offline	No
6	IMTFRI [20]	Top-K	Online	Yes
7	TKFIM [15]	Top-K	Offline	No
8	TKIFIs Miner [14]	Top-K	Offline	No
8	TKIFRPM (Proposed)	Top-K	Online	Yes

Amphawan et al. [2] have proposed top-K regular frequent patterns from transactional datasets, namely MTKPP. This approach performs a single scan of a given dataset to record the TIDs in which the corresponding items appear. The MTKPP uses a top-K list for ranking and recording top-K frequent regular patterns. The procedure discovers the patterns by performing breadth-first traversal of the search space. The support of every pattern is determined by TIDs' intersection mechanism. The patterns with their support and regularity are checked against the minimum support of the patterns in the top-K List and the user-given regularity threshold. A discovered pattern is merged with the patterns in the top-K list if the pattern's support is greater than the minimum support and its regularity is less than or equal to a user's given regularity threshold. However, the main issues with MTKPP are the excessive candidate generations and heavy memory consumptions due to the in-memory placement of the TIDs of all the items without compression, which could compromise the performance of MTKPP on sparse and dense datasets.

Amphawan et al. [12] have proposed an approach in another research work on the top-K frequent regular pattern mining, namely TR-CT (top-K regular frequent itemset mining based on compressed TIDs). The TR-CT approach uses a top-K list and works using a best-fit strategy, similar to MTKPP. The TR-CT merges the itemsets (based on common prefixes) to present in the top-K list and finds their support by intersecting their compressed TIDs. The discovered itemsets, along with their support and regularity, are ranked in the top-K list. The performance of TR-CT on sparse datasets is similar to that of MTKPP. However, TR-CT is found to be more efficient than the MTKPP technique on dense datasets by requiring less memory due to the compression of the consecutive TIDs.

In another work, Amphawa et al. [19] have proposed another approach, namely TKRIMPE (top-K regular-frequent itemset mining with dataset partitioning and support estimation), which is more efficient than TR-CT on sparse datasets. The TKRIMPE works in two phases. In the first phase, the dataset is partitioned, and each partition is scanned to collect the support, regularity, and TIDs of a single itemset. The itemsets are ranked in the top-K list. The second phase is reserved for the itemset mining purpose. This phase is performed using a best-fit strategy over itemsets available in the top-K list. The itemsets with highest support and common prefixes are considered first for candidate formation. Once the candidate is formed, the support estimation method is applied to determine its estimated support. If the candidate itemset has lower estimated support than the least support of the itemset in the top-K list, it is dropped from the search space.

The top-K frequent regular pattern discovery methods avoid a large number of patterns. However, these methods still produce a result set containing redundant patterns. To avoid the redundancy in the result set, top-K frequent regular closed patterns method with minimum length constraint is proposed by Amphawan et al. [21], namely TFRC-Mine. The TFRC-Mine follows a two-step process, as with TKRIMPE. The first step is to scan and initialize the top-K list and the second step is to use best-fit strategy for search space exploration and pattern mining. For support calculation partition, a dynamic bit vector approach is applied in TFRC-Mine. The TFRC-Mine method is found to be effective to prune the candidate itemsets and to count support of the patterns. The TFRC-Mine is reported to be more computationally efficient than MTKPP [2], TR-CT [12], and TKRIMPE [19] for large value of K over dense datasets.

Ravikumar et al. [22] has proposed periodic frequent-equivalence class transformation (PF-ECLAT) for periodic frequent itemset mining. The PF-ECLAT uses frequent itemset mining ECLAT (equivalence class clustering and bottom-up lattice traversal) algorithm for periodic frequent itemset mining. The PF-ECLAT requires minimum support threshold and maximum periodicity as parameter in advance from users. Firstly, the PF-ECLAT finds one-length patterns and converts the given dataset into a vertical format by applying the PFP-List data structure. Secondly, the PF-ECLAT works on the PFP-List in depth-first order and applies a down word closure property to effectively prune itemsets from the search space. The PF-ECLAT algorithm is found to be more computationally and memory-usage efficient than other periodic frequent itemset mining algorithms. However, the PF-ECLAT has the limitations of requiring a support threshold and application on the offline datasets.

Bandit et al. [20] has introduced a single-pass incremental miner of top-k frequent-regular itemset (IMTFRI). The algorithm is described in two main steps. In the first step, the incoming batch of transaction is read one by one to record the frequency information of items. In the second step, the entire top-K frequent regular items list is computed using the frequency information of the items in the first step. The IMTFRI approach increments the frequency counts of already discovered top-K frequent regular items. The new top-K frequent regular itemsets are also discovered from the updated dataset chunk. For the frequency counting, it uses a partition dynamic bit vector (PDBV) data structure. This approach uses breadth-first best-fit traversal to mine top-K frequent regular itemsets. No pruning mechanism is utilized during traversal to prune the itemsets whose frequency is decreasing due to the baseline frequency updation mechanism. In other words, IMTFRI uses only single criteria to merge itemsets whose prefixes are the same rather than also considering their frequency, which could drop due to baseline frequency updation.

Iqlab et al. [15] has proposed a top-K frequent itemset mining (TKFIM) algorithm for detecting FIs without a user-defined support threshold. The TKFIM uses the idea of equivalence classes of set theory where each class represents an independent group of itemsets. The support of the itemsets is counted and the procedure is applied to the vertical database structure containing transaction IDs and items. The class-based strategy is used to determine itemsets of highest support, and the joining process is used for mining class-based candidates. The current class stops generating candidate itemsets if the least support in the top-K list is greater than the support of an itemset, and the next class joining is used. This process is repeated as long as frequent or candidate itemsets are found. However, the TKFIM does not merge itemsets, and uses a class-based strategy for counting support for mining, which can work on the static offline datasets but not on the incremental dynamic online datasets.

Rehman et al. [14] have recently introduced a methodology to mine identical frequent patterns without a user support threshold, namely the TKIFIs Miner. This approach generates patterns from offline datasets. However, the patterns it produces are not regular patterns. The method also accepts K as a parameter from the user finding K topmost support identical patterns.

3. Background

This section provides detailed preliminary background information to understand the frequent regular identical itemsets and top-K frequent regular identical itemsets which are used and presented in the proposed approach in the next section. Several definitions are presented along with their mathematical equations to provide the background knowledge of the area and proposed method. The notions used in the equations are presented and explained in Table 2.

Table 2. Introduction and explanation of the notions used in the mathematic equations.

S. No	Symbol/Term	Explanation
1	tr	Incoming transaction
2	t_b/T_b	Transaction buffer
3	θ_r	Regularity threshold
4	θ_s	Support threshold
5	$ISI - Tables$	Item support index table
6	P	Possible set of candidate itemsets
7	F_l	Frequent regular patterns or patterns generated at the parent node
8	C_l	Candidate itemsets generated or available at the parent node
9	F_{l+1}	New generated frequent pattern
10	C_{l+1}	New set of candidate itemsets/patterns
11	I	Set of all items in a dataset
12	X	Itemset or pattern
13	t_j	Individual transaction
14	I_{FR}	Set of frequent regular patterns

Let I = $\{i_1, i_2, \ldots \ldots \ldots .i_n\}$ present the set of all items. A set X = $\{i_j, \ldots \ldots \ldots .i_k\} \subseteq I$ j $\leq k, j, k \in [1, n]$ is called an itemset or pattern. A transaction dataset T = $\{t_1, t_2, \ldots, t_m\}$ over dataset D contain m number of total transactions. Every transaction $t_i \in$ T is 2-tuple (i.e., t_i = (TID, Y) where TID $\in [1, m]$ and Y is a pattern). If a pattern X $\subseteq Y$ is found in transaction $t_i \in T$, then the transaction ID (i.e., TID) of the corresponding transaction is written as $t_j^x,\ j \in [1, m]$. The support count of pattern X is the set of all corresponding TIDs where transactions contained X as patterns (i.e., T^x = $\left\{t_j^x, \ldots \ldots, t_k^x\right\}$, $j, k \in [1, m]\ j \leq k$). The period of pattern X is the number of transactions where X as a pattern does not appear between any two consecutive transections $t_j,\ t_{j+1} \in T,\ J \in [1, m]$. The regularity of pattern X is the maximum number of transactions between two consecutive transactions $t_{j,}t_{j+1} \in T^x\ j \in [1, m]$ where X does not occur in the transaction dataset (i.e., $P^x = max\left(P_1^x, P_2^x, \ldots .P^x_{|T^x|-1}, P^x_{|T^x|}\right)$). On a user-supplied support threshold and regularity threshold, R is the set of patterns qualifying aforementioned constraint (i.e., $R = \{\forall i = 1 \cdots n\ X_i | \forall i = 1 \cdots n\ support(X_i) \geq \theta_s \wedge \forall i = 1 \cdots n\ Reg(X_i) \leq \theta_r\}$). The support count of all the patterns in R can be collected in a set of distinct support counts $S = \{\forall i = 1, \ldots \ldots, m\ S_i = \text{max\_supp}_\text{i}(R)\}$. The patterns in R can be classified into identical frequent regular (IFR) patterns and top-K identical frequent regular patterns.

3.1. Definition 1—Identical Frequent Regular (IFR) Itemsets

One or more itemsets are called (I_{FR}), if and only if every itemset belongs to a set of frequent regular patterns R and all the patterns have same support count $s \in S$, where S is the set of support counts of itemsets belong to R, as shown in the following Equation (1):

$$I_{FR} = \{\forall i = 1, \ldots, m\ , 1 \leq m \leq nX_i \in R \mid \forall i = 1, \ldots, m, Support(X_i) = swhere\ s \in S\} \quad (1)$$

For example, if R is a set of frequent regular patterns with user-given regularity = 3, then any itemset in Table 3 can be considered as a member of R and, hence, is I_{FR}.

Table 3. Transaction dataset batch A.

TID No.	Transactions
1	a, b, d
2	b, c, d, e
3	a, b, e, g, h
4	b, e, g, h
5	a, d, e, g
6	c, d, g, h
7	b, c, d, e, g, h
8	a, b, e, h
9	b, d, e, g, h
10	a, b, c, g, h

3.2. Definition 2—Top-1 IFR Itemsets

One or more itemsets are called *top-1 IFR* (I_{FR_1}), if and only if every itemset belongs to a set of frequent regular patterns R, and all the patterns have a similar highest support count $s_1 \in S$, where S is the set of support counts of itemsets belong to R, as shown in the following Equation (2):

$$I_{FR_1} = \{\forall i = 1, \ldots, m\ ,\ 1 \leq m \leq n\ \ X_i \in R \mid \forall i = 1, \ldots, m,\ \ \exists s_1 \in S\ Support(X_i) = s_1\}. \quad (2)$$

For example, an itemset A $= \{\text{b}\}$ positioned at serial No. 1 in Table 3 with support (A) = 8 and regularity (A) = 3 is termed as I_{FR_1}. This itemset has the first highest support in R, and it also satisfies the user-given regularity value that is 3.

3.3. Definition 3—Top-Kth IFR Itemsets

One or more itemsets are called top-K IFR (I_{FR_K}), if and only if every itemset belongs to a set of frequent regular patterns R and all the patterns have the same Kth highest support count $s_k \in S$, where S is the set of support counts of itemsets which belong to R, as shown in the following Equation (3):

$$I_{FR_K} = \{\forall i = 1, \ldots, m\ ,\ 1 \le m \le n\ \ X_i \in R \mid \forall i = 1, \ldots, m,\ \ \exists s_k \in S\ Support(X_i) = s_k\} \tag{3}$$

For example, any itemset in Table 3 is a member of a frequent regular pattern and is referred as Top-Kth IFR.

In the incremental framework, an online transaction buffer TB is a collection of m transactions received online in a particular time t and is denoted as $TB_t = \{tid_1, tid_2, tid_3, \ldots, tid_m\}$. Then, a transactional set at any current time is a collection of TBs received so far (i.e., $T = \{TB_1,\ TB_2,\ TB_3, \ldots,\ TB_t\}$). On a user-provided parameter that is the number of required patterns K and a regularity threshold σ_r, the problem of *frequent* regular pattern mining is to find the topmost $-K$ identical frequent regular patterns set from the online transactions dataset.

3.4. Definition 4—Topmost-K IFR Set

A set of IFRs (TK_IFR_s) of highest support to the Kth support is found from a given online transaction dataset D, as is stated in the following Equation (4), where R is the set of frequent and regular itemsets:

$$TK\_IFR_s = \left\{\forall \mathrm{j} = 1,\ 2,\ 3,\ \ldots,\ K\ \ I_{FR\_j} \in R\ \middle|\ \forall i = 1,\ 2,\ 3,\ \ldots\ ,\ K\text{–}1,\ support\ \left(I_{FR\_j}\right) > support\ \left(I_{FR\_j+1}\right)\right\} \tag{4}$$

3.5. Case Example

Consider a dataset consisting of 8 items and 20 transactions. The dataset is bifurcated into two groups of transactions (shown in Tables 3 and 4, respectively) received in an online transaction buffer. Mining operation is performed two times on the dataset with the topmost patterns $K = 5$ and regularity $\theta_r = 3$.

Table 4. Transaction dataset batch B.

TID No.	Transactions
11	a, d, f
12	b, d, f
13	a, b, c, d, e
14	a, c, d
15	a, b, d
16	a, c, d, e
17	b, c, d, f
18	c, e, f
19	a, b, c, d
20	a, c, d, f

Tables 3 and 4 represents the transaction datasets bifurcated into batch A and batch B, respectively. After performing a mining operation on the transaction dataset in batch A, the topmost-5 *IFR* sets with a regularity less than or equal to 3 are shown in Table 5. The results produced by performing a mining operation on the online dataset given in Table 5. Subsequently, the dataset is updated with transaction dataset batch B, as shown in Table 4. After the update, the mining operation is performed again for finding the topmost-5 *IFR* set with a regularity less than or equal to 3. The results produced for the topmost-5 *IFR* set is shown in Table 6, which shows that, after the update, the results produced for the topmost-5 IFR set are different than those shown in Table 5.

Table 5. Top-5 *IFR* set after mining batch A.

K	Itemsets	Regularity/Periodicity	Support
1	b	3	8
2	e, g, h	2	7
3	d, be, bh gh	3 2	6
4	a, bg, eh, beh, bhg eg	3 2	5
5	ed, beg, egh, begh gd	3 2	4

Table 6. Top-5 *IFR* set after mining batch A and B.

K	Itemsets	Regularity/Periodicity	Support
1	d	3	15
2	b	3	13
3	a	3	12
4	g, h	2	7
5	bh, ac, gh	3 2	6

4. TKIFRPM Approach

This section provides a detailed discussion of the proposed top-*K* identical frequent regular pattern mining (TKIFRPM) approach for producing topmost-*K* IFRs. The approach is able to perform mining on online datasets. The overall TKIFRPM approach/methodology and algorithm are shown in Figure 1 and Algorithm 1 respectively. The TKIFRPM algorithm receives online committed transaction *tr* in step 1. The committed transaction *tr* is copied into transactions buffer *tb* in step 2. Step 3 checks whether the transactions buffer *tb* is full or not. If the transaction buffer *tb* is full, step 4 applies an online synopsis or summary data structure (explained in Section 4.1), and the synopsis data structure ISI-table is updated when the transaction buffer *tb* is found to be full. Step 5 receives all distinct support K 1-itemsets from the ISI-table in the candidate list C_l. Step 6 calls the top-*K* IFRs mining search procedure to return *K* topmost distinct support and regular itemsets of a size greater than 1. Step 8 and 9 return the itemsets in the top-*K* list for the current buffer *tb*.

Algorithm 1 Overall TKIFRPM

```
r:—$Incoming\ transaction$
Input:
$t_b$:—$Trasaction\ buffer$
$\theta_r$—Regularity threshold
$\theta_s$: —Support threshold, initially zero
ISI − tables :—Item support index table
(0) flag = 0 // for the merge sort to be performed for the first time on ISI-Tables
(1)    $while\ ((t_r = read\ (\ ))! = \varnothing)$
(2)            $t_b = t_b \cup t_r$
(3)    $if\ (t_b\ is\ full)$
(4)        $update\_ISI\_tables(ISI - tables, t_b,\ flag)$
(5)        $flag = 1$ // to perform binary search when tb is full
(6)        $C_l \leftarrow \varnothing,\ F_l \leftarrow \varnothing,$
(7)        $Select\_K\_item(\ ISI - tables,\ K,\ C_l\ ,\ \theta_r\ )$
(8)        Gen_Top–$KIFRs(\ F_l,\ C_l,\ \theta_r,\ \theta_s,\ top - K\ List)$
(9) Output $top - K\ List;$
```

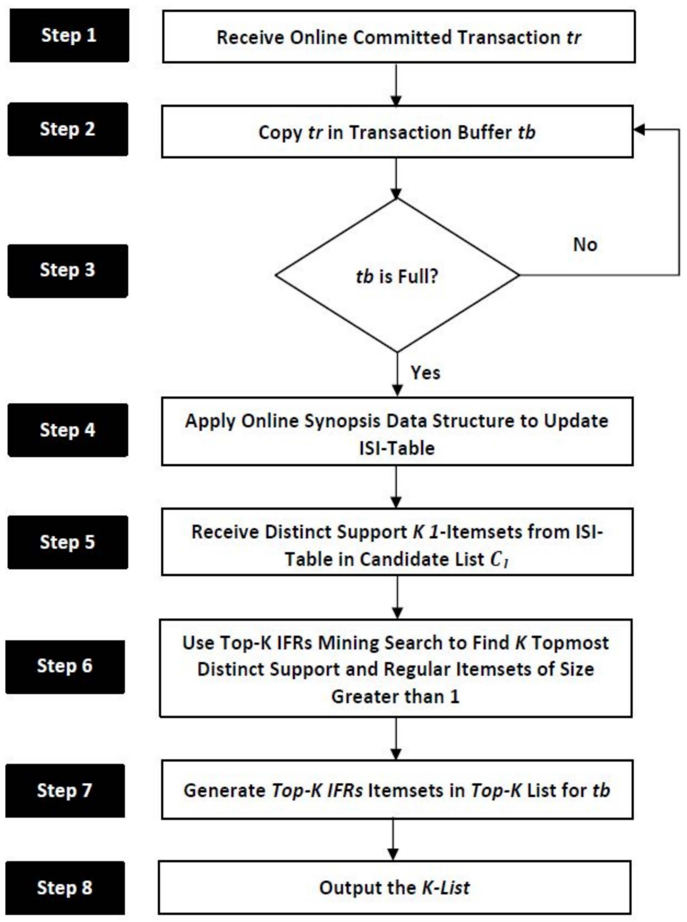

Step 1
Receive Online Committed Transaction *tr*
Step 2
Copy *tr* in Transaction Buffer *tb*
Step 3
tb is Full?
No
Yes
Step 4
Apply Online Synopsis Data Structure to Update ISI-Table
Step 5
Receive Distinct Support *K* 1-Itemsets from ISI-Table in Candidate List C_1
Step 6
Use Top-K IFRs Mining Search to Find *K* Topmost Distinct Support and Regular Itemsets of Size Greater than 1
Step 7
Generate *Top-K IFRs* Itemsets in *Top-K* List for *tb*
Step 8
Output the *K-List*

Figure 1. The overall TKIFRPM approach and methodology.

4.1. Item Support Index Tables (ISI-Tables)

The classical FIM procedures are designed to perform mining on static datasets. The static dataset is permanently stored on secondary storage media. These procedures scan offline static datasets more than once to compute all of the frequent patterns. On the other hand, the size of an online dataset is not fixed, and the online dataset continuously remains in the updation mode, which is simultaneously updated when an online committed transaction arrives or when the online transaction buffer *tb* is full. Therefore, there is no specific time for online dataset updation. The classical frequent pattern mining approaches when applied on an online dataset will inherently perform multiple scans of entire online dataset stored in the secondary storage media, resulting in a large response time. The pattern mining routine should have reasonably short response time and perform only one pass for the selected items on an online dataset when applied recurrently [23,24]. This motivates the designing of a summary data structure. The TKIFRPM has proposed a novel summary data structure, namely the items support index tables (ISI-tables). The structure of the ISI-tables is shown in Figure 2. The ISI-tables have a dynamic data structure which is maintained in the primary memory area. The ISI-tables data structure stores items along with their support and TIDs. In addition, the ISI-tables data structure stores items in an order (ascending or descending) automatically after it is updated. The proposed mining routine will perform only one pass on the selected items in the ISI-tables rather than scanning the entire data structure for the generation of topmost frequent patterns. The ISI-tables' data structure supports one-time and incremental mining operations.

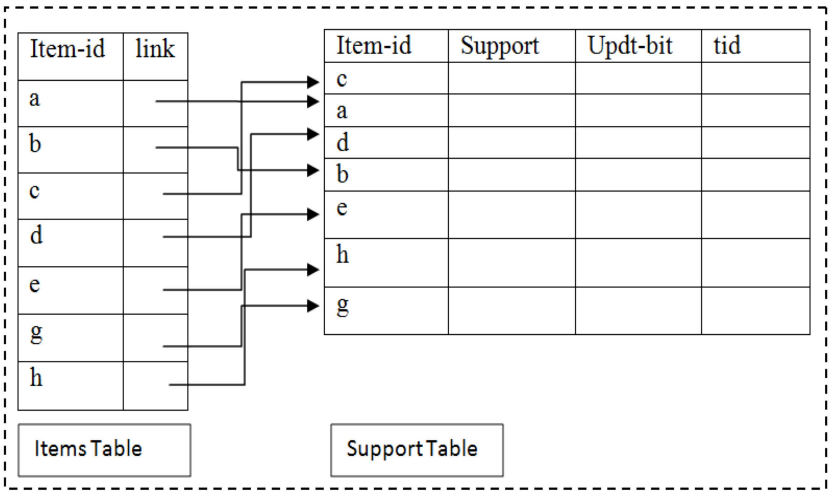


Figure 2. Structure of ISI-tables.

4.2. Update ISI-Tables

The ISI-tables are updated using the "update_ISI_tables" algorithm as shown in Algorithm 2. The algorithm receives the following three parameters: *ISI-tables*, transaction buffer T_b, and *flag*. The *ISI-tables* are updated with the transactions received in the transaction buffer T_b. Each transaction in the T_b is scanned, and for every item in a selected transaction from T_b, the corresponding row in the support table is adjusted. Every item in the selected transaction is hashed and stored in the items table to find the address of the corresponding row $(f_1 - ptr)$ in the support table. The address of an item is used to update the support, Updt-bit, and TID fields in the support table. At the end, the value of the *flag* is checked. If it is 1, the support table is sorted in descending support order using a merge-sort. The *flag* value 1 indicates that the *ISI-tables* are populated for the first time, and the *flag* value 0 occurs every time the transaction buffer is copied to the *ISI-tables*, and that updation of the *ISI-tables* is needed. The *ISI-tables* maintain items in the support descending order using a binary search. The items with their support values are searched in the support table using a binary search to find the suitable place for its adjustment. This way, *ISI-tables* maintain the order of the items without performing sorting operations on all items in a dataset.

Algorithm 2 update_ISI_tables (ISI-tables, T_b, flag)

```
Input:
$ISI - tables : -Items\ Support\ Index\ Tables$
$T_b : -transaction\ buffer$
$flag - to\ perform\ Merge\ sort\ \ or\ binary\ search$
(1) $foreach\ tr \in T_b$
(2) $foreach\ item \in t_r$
(3)         $f_1 - ptr$ = ISI − tables.items¯table[item]
(4)         ISI − tables.Support¯table[$f_1 - ptr$].Support + +
(5)         ISI − tables.Support¯table[$f_1 - ptr$].Updt_bit ← 1
(6)         ISI − tables.Support¯table[$f_1 - ptr$].id ∪ tr.tid
(7) If (flag==1)
(8)    Merge-sort (ISI-tables) // to sort the support table in descending order of the support
(9) else
(10)      Binary-search (ISI-tables) // to place the updated items in their correct order in
support table
(11) Return
```

The "select-K-itemset" algorithm shown in Algorithm 3 copies a 1-itemset from the ISI-tables. The ISI- tables may contain more than one 1-itemset with a similar support. The similar support of all these are treated as distinct and selected if their support is above or equal to the *k*th topmost support. This procedure considers three parameters, as follows: *ISI-tables*, *K*, and C_l. The "select-K-itemset" algorithm counts the unique support of items in *ISI-tables* and select the corresponding items until the total number of unique supports is equal to *K*.

Algorithm 3 *select_K_item* $(ISI\text{-}tables,\ K,\ C_l)$

```
Input:
$\cup_{spt}$ $\leftarrow$ $unique\ support$
$prev - spt \leftarrow previous\ support$
(1) $i \leftarrow 1,\ \cup_{spt} \leftarrow 0,\ K \leftarrow 1,\ C_l \leftarrow\ \varnothing$
(2) New item
(3) $item.id \leftarrow ISI - tables.support\bar{\ }table.item\_\ id[i]$
(4) $item.tid \leftarrow ISI - tables.support\bar{\ }table.tid[i]$
(5) $C_l \leftarrow\ C_l\ \cup item$
(6) $prev - spt \leftarrow support\ (item.tid)$
(7) $repeat\ Until\ \ \cup_{sprt} <= k$
(8) $i++$
(9) $New\ item$
(10) $item.id \leftarrow ISI - tables.item - id[i]$
(11) $item.tid \leftarrow ISI - tables.support\bar{\ }tabel.item - tid[i]$
(12) $C_l \leftarrow\ C_l\ \cup item$
(13) $if\ (prev - spt\ != support(item.tid))$
(14) $\cup_{sprt}++$
(15) $prv - spt \leftarrow support(item.tid)$
(16) $if\ \left(\cup_{sprt} \geq k\right)$
(17) $break$
(18) Return $C_l$
```

4.3. Topmost-K IFRP Miner

The top-K IFRP set miner is a derivation of the classical depth-first procedure for the FIS mining [25]. The method finds the set of topmost-K IFRPs using the "gen_top_KIFRs" algorithm shown in Algorithm 4. The proposed method traverses the search tree using a recursive depth-first strategy. In every iteration, it forms a new node of a tree, and every new node is logically divided into two parts, i.e., the head (F_{l+1}) and tail (C_{l+1})). The head of every new node presents a frequent pattern (F_{l+1}) greater than the current minimum threshold. This head *p* of a new node (F_{l+1}) is an extension of parent node head potion F_l with any item $\in C_l$, where C_l is candidate item list (i.e., a tail of a parent node). The tail of a new node C_{l+1} is formed in two steps. Firstly, all items with support less than the item combined with previous F_l for F_{l+1} are collected in *P* (possible candidate). Secondly, the support of every item along F_{l+1} is checked against the current minimum threshold θ_s.

4.4. Computational Complexity

The frequent pattern mining approach is comprised of candidate set generation and the support for finding of candidates. The more candidate set generation is performed, the more time is consumed for their support calculation. For the support finding operation of the candidate itemsets, the TKIFRPM performs TIDs intersection of two sets of TIDs. The TKIFRPM completes one TID intersection in *nlogn* time. As it performs a binary search to find whether the TID exist in another set or not, this operation requires *nlogn* time for finding one TID. If the total number of transactions received are *m* (maximum TIDs received), then the total number of operations required to complete one support calculation operation is $m(nlogn)$. The TKIFRPM will create maximum 2^n nodes in the worst case (i.e., to find all frequent itemsets in K) scenario, where $n + 1$ nodes will not be needed in the support calculation operations. Therefore, the total number of nodes to be created will be $2^n - (n + 1)$. Thus, the total number of operations required for frequent regular pattern mining in the worst case will be $((2^n - (n + 1))*m\ (nlogn))$, and it will be the same for every case of mining when the mining operation "Gen_top_KIFRs" is performed on the data received from the ISI-tables. In every case, m transaction will be received in transaction buffer T_b, and if every transaction contains n items, four constant time operations will be required to be completed. Thus, the time required to complete all operations in updating ISI-tables will be *4mn*. The binary search or merge-sort is performed for enforcing order on the items and will require *nlogn* time. Thus, the total time required to update the ISI-tables will be *4mn+nlogn*, and the time required for selecting K itemsets for mining will be n. Therefore, the total time taken or the computational complexity of the TKIFRPM is shown in Equation (5), as follows:

$$Computational\ Complexity = \{((2^n - (n+1)) * m(nlogn)) + 4mn + nlogn + n\} \quad (5)$$

Algorithm 4 Gen_top-KIFRs ($F_l,\ C_l,\ \theta_r,\ \theta_s,\ TK\_\ IFRs$)

```
Input
K:          Topmost-K regular frequent itemsets
$F_{l+1}$:  New frequent pattern extended with highest support item from $C_l$
$C_{l+1}$:  Set of candidate items found frequent with the frequent pattern $F_{l+1}$
$F_l$:      Frequent pattern or head portion of the parent node
$C_l$:      Candidate itemset list of parent node
$\theta_r$: User-given regularity threshold
$\theta_s$: Support threshold
$P$: Possible candidate set
(1) Foreach $x \in C_l$
(2) $F_{l+1}$ = $F_l$ $U$ $\{ x \}$
(3) $TK\_\ IFRs = TK\_\ IFRs$ U $F_{l+1}$
(4) $\theta_s$ = min_$support(TK\_\ IFRs\ )$
(5) $P = \{ i\ :\ i \in C_l\ and\ \ support(i) \leq support(x)\}$
(6) $C_{l+1} = \varnothing$
(7) Foreach $y \in P$
(8)       If ($support\ (F_{l+1}\ U\ \ y\ ) \geq\ \theta_s\ and$ Periodicity $(F_{l+1}\ U\ \ y\ ) \leq \theta_r$ )
(9)             $C_{l+1} = C_{l+1} U\ y$
(10) Gen_Top-k_IFRs ( $F_{l+1},\ \ C_{l+1},\ \ l,\ \ \theta_r,\ \ \theta_s,\ TK\_\ IFRs$ )
```

5. Evaluation and Discussion

The experiments are conducted to evaluate and compare the performance trends of the proposed TKIFRPM approach with the available approaches. The TKIFRPM method can be compared with all of the available topmost-K identical frequent pattern mining approaches. However, due to limitations in resources and time, the comparison of the TKIFRPM is limited to the MTKPP only. The reason for selecting MTKPP is its operational and structural relevancy to the proposed TKIFRPM to provide better and widely acceptable results. However, the conclusion derived can be theoretically applied to other relevant approaches. The performance of both of the approaches is recorded in terms of runtime (i.e., execution) and memory usage to find/mine the topmost-K identical frequent regular patterns using incremental online datasets.

5.1. Experimental Setup and Datasets

The TKIFRPM and MTKPP approaches are implemented in Python 3.0 to perform the comparative evaluation. The three benchmark datasets, namely mushroom, accident, and chess, are used for both algorithms. The detailed characteristics of the used datasets are presented in Table 7. The comparative evaluation is carried out using K, θ_r, and T_b, where K presents the topmost patterns, θ_r represents is periodicity, and T_b represents the transaction buffer size. The different K = {100, 200, 300, 400, 500] values are used in the experimental setup to discover the topmost patterns. The fixed value of periodicity $\theta_r = 15$ is used in every experiment for each of the datasets to compare the runtime on each iteration. The transaction buffer T_b = {3K for mushroom, 30K for accident, and 1.5K for chess} sizes (i.e., size and number of transactions of the dataset) for recording online committed transactions is used to simulate an online dataset environment. The experiments are performed in three iterations on every dataset. The variable size of the transaction buffer is used to conclude every experiment in each of the three iterations due to the size of the datasets, which varies in terms of the number of transactions. The transaction buffer T_b size is kept at 3K transactions for the mushroom dataset, 30K transactions for the accident dataset, and 1.5K transactions for the chess dataset. An individual transaction is read from the dataset and recorded in the transaction buffer in every transaction, and when the transaction buffer is found full it is copied to the ISI-tables.

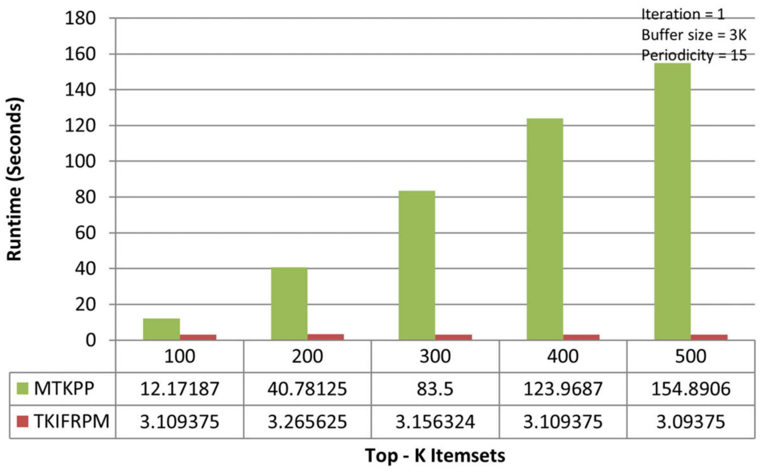
Table 7. Benchmark datasets description.

Datasets	Number of Items	Average Pattern Length	Number of Transactions
Mushroom	119	23	8, 124
Accident	468	33.8	340, 183
Chess	76	37	3, 196

5.2. Runtime Evaluation and Comparision

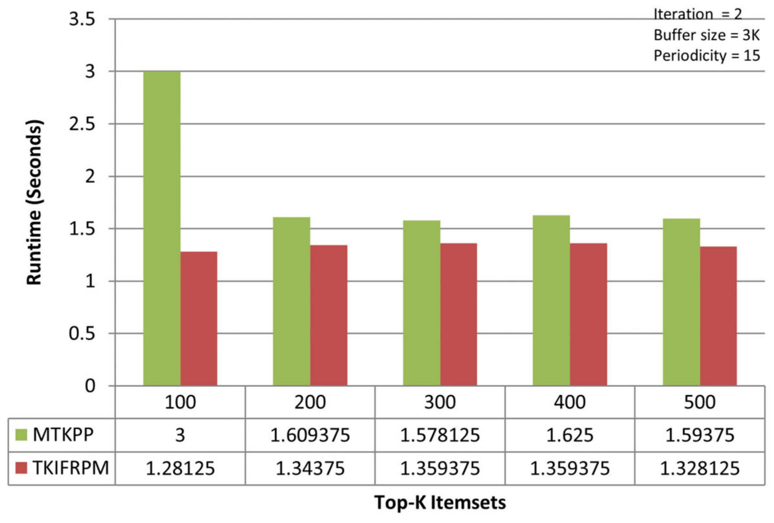
It is observed that TKIFRPM (i.e., average runtime = 3.15 s) takes less time to find and produce designated topmost-K patterns then MTKPP (i.e., average runtime = 83.16 s) in the first iteration, as shown in Figure 3a. However, if a dataset size is large, the MTKPP method is expected to consume more time to find the topmost-K patterns due to its multiple scans and support calculation architecture. In the second iteration, it is observed that TKIFRPM (i.e., average runtime = 1.33 s) is more efficient, performance-wise, at producing designated topmost-K patterns than the MTKPP (average runtime = 1.98 s), as shown in Figure 3b. However, in the third iteration, as the dataset size is increased, the performance of TKIFRPM (i.e., average runtime = 1.60 s) is a little slower to produce designated topmost-K patterns than the MTKPP (average runtime = 0.29 s), as shown in Figure 3c. It is due to the fact that in these datasets, the patterns could be of the similar/same support, and that the MTKPP architecturally skips the multiple patterns of the same support during its computation process for producing designated topmost-K patterns and results in a shorter time. However, the TKIFRPM do not skip multiple patterns of the similar/same support, and ranks them properly to produce designated topmost-K patterns because of its identical characteristics, in order to pick and process all the identical patterns.

On the accident dataset, the TKIFRPM performs better to produce designated topmost-K patterns then the MTKPP in all three iterations. In the first iteration, the TKIFRPM has an average runtime of 104.86 s, and the MTKPP has an average runtime 2061.55 s, as shown in Figure 4a. In the second iteration, the TKIFRPM has an average runtime of 225.13 s, and the MTKPP has average runtime of 3303.69 s, as shown in Figure 4b. In the third iteration, the TKIFRPM has average runtime of 382.95 s, and the MTKPP has average runtime of 4637.52 s, as shown in Figure 4c. For all iterations, it is due to the fact that MTKPP performs multiple scans for support calculation to produce designated topmost-K patterns, which results in a long runtime, whereas TKIFRPM does not perform multiple scans, which results in low runtime and makes TKIFRPM more efficient than MTKPP in terms of performance.



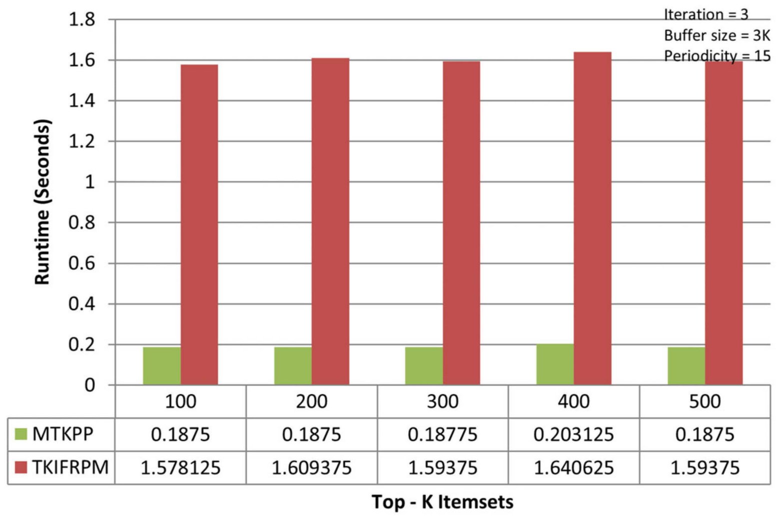
	100	200	300	400	500
🟩 MTKPP	12.17187	40.78125	83.5	123.9687	154.8906
🟥 TKIFRPM	3.109375	3.265625	3.156324	3.109375	3.09375

(**a**)



	100	200	300	400	500
🟩 MTKPP	3	1.609375	1.578125	1.625	1.59375
🟥 TKIFRPM	1.28125	1.34375	1.359375	1.359375	1.328125

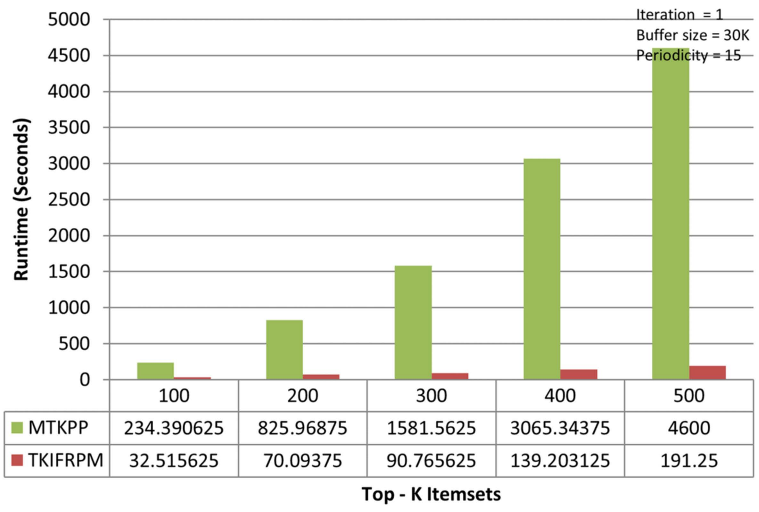
(**b**)



	100	200	300	400	500
🟩 MTKPP	0.1875	0.1875	0.18775	0.203125	0.1875
🟥 TKIFRPM	1.578125	1.609375	1.59375	1.640625	1.59375

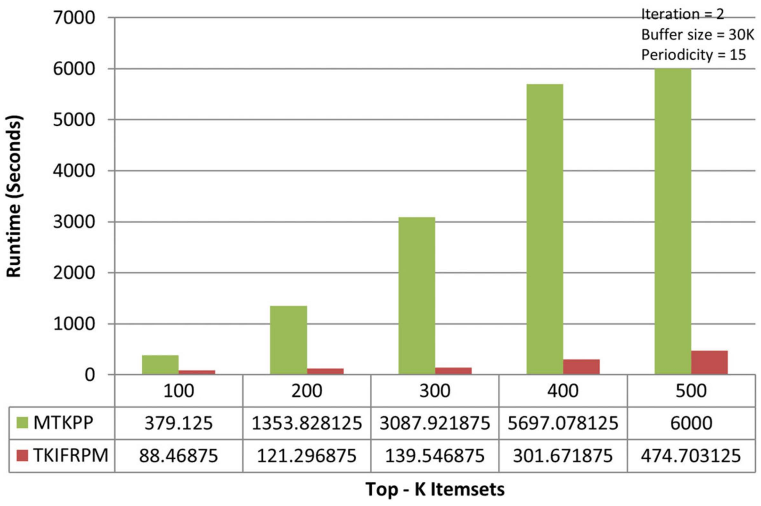
(**c**)

Figure 3. (**a**) Performance trends of the first iteration on the mushroom dataset. (**b**) Performance trends of the second iteration on the mushroom dataset. (**c**) Performance trends of the third iteration on the mushroom dataset.



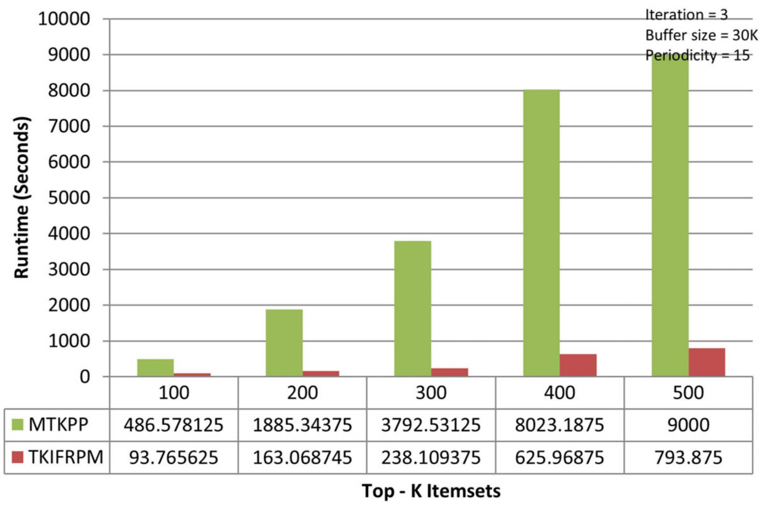
	100	200	300	400	500
🟩 MTKPP	234.390625	825.96875	1581.5625	3065.34375	4600
🟥 TKIFRPM	32.515625	70.09375	90.765625	139.203125	191.25

(**a**)



	100	200	300	400	500
🟩 MTKPP	379.125	1353.828125	3087.921875	5697.078125	6000
🟥 TKIFRPM	88.46875	121.296875	139.546875	301.671875	474.703125

(**b**)



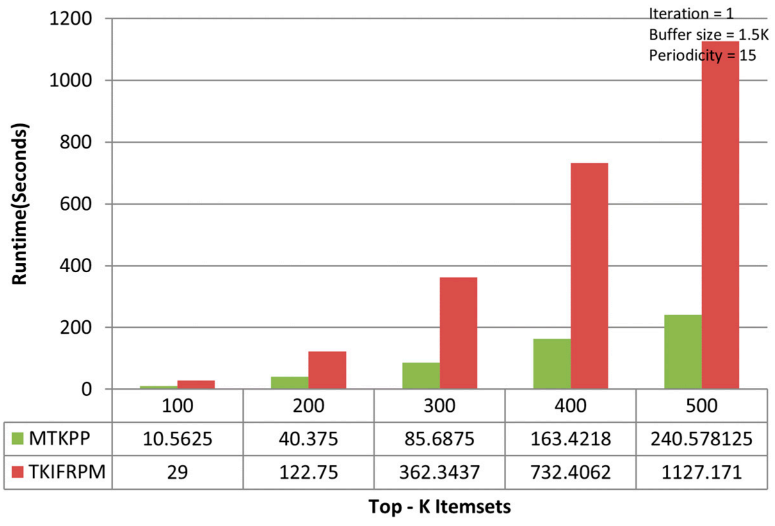
	100	200	300	400	500
🟩 MTKPP	486.578125	1885.34375	3792.53125	8023.1875	9000
🟥 TKIFRPM	93.765625	163.068745	238.109375	625.96875	793.875

(**c**)

Figure 4. (**a**) Performance trends of the first iteration on the accident dataset. (**b**) Performance trends of the second iteration on the accident dataset. (**c**) Performance trends of the third iteration on the accident dataset.

On the chess dataset, the MTKPP takes less time (i.e., average runtime = 108.12 s) than TKIFRPM (i.e., average runtime = 474.73) to produce designated topmost-K patterns in the first iteration, as shown in Figure 5a. This is because the size of the dataset is initially small. Having a small number of multiple patterns of the same support, as discussed earlier, means that MTKPP skips multiple patterns of the same support, while TKIFRPM do not skip them and processes them properly. In addition, due to being a small dataset, the MTKPP picks the patterns with a length equal to 1 of the highest support, whereas TKIFRPM picks multiple patterns of variable length and of the K highest support simultaneously. However, as the size of the dataset grows, the TKIFRPM performs better to produce designated topmost-K patterns as compared to MTKPP in iterations two and three, respectively. In the second iteration, the TKIFRPM has average runtime of 51.22 s, and the MTKPP has average runtime of 131.12 s, as shown in Figure 5b. In the third iteration, the TKIFRPM has average runtime of 10.64 s, and the MTKPP has average runtime of 121.51 s, as shown in Figure 5c. This is because as the size of a dataset increases, the existence of the patterns of highest support is similarly increased, and could be of variable length; therefore, the MTKPP performs multiple scans to find the support of multiple patterns, and the TKIFRPM does not perform multiple scans for support calculation.

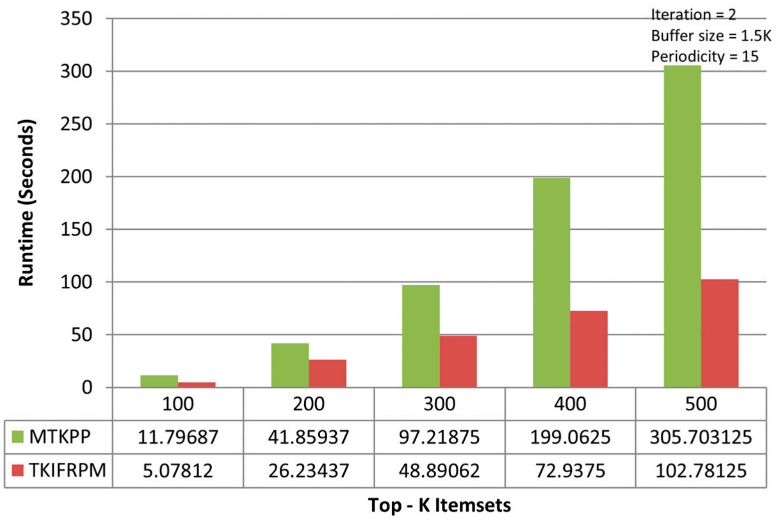
Figure 6a,b shows the consolidated results of all iterations for TKIFRPM and MTKPP on the mushroom dataset, where TKIFRPM is faster than MTKPP. The TKIFRPM completes the mining operation in maximum 3 s, whereas MTKPP consumes a maximum 150 s to complete the same mining operation. The results on the accidents dataset are shown in the Figure 7a,b, and the results on the chess dataset are shown in the Figure 8a,b, which signifies that the performance differences are quite clear for the TKIFRPM and MTKPP. The TKIFRPM is significantly faster in the topmost-K IFR itemset mining than MTKPP. This is due to that fact that MTKPP is simply using a breadth-first approach, similar to Apriori, and does not apply any pruning method in the candidate generation. Thus, all the candidates who have common prefixes are merged, and their support is intersected using TIDs sets. However, the TKIFRPM strategy of working with candidates is more effective. The TKIFRPM method first checks the support against the minimum support and the periodicity against the maximum periodicity. If a candidate has less support than the minimum support or periodicity greater than maximum periodicity, then the candidate is pruned from further consideration. Thus, the entire sub-tree with the candidate as root node is removed from the search space. Therefore, the TKIFRPM method uses a pruning method, which makes it working faster than the MTKPP method.



	100	200	300	400	500
🟩 MTKPP	10.5625	40.375	85.6875	163.4218	240.578125
🟥 TKIFRPM	29	122.75	362.3437	732.4062	1127.171

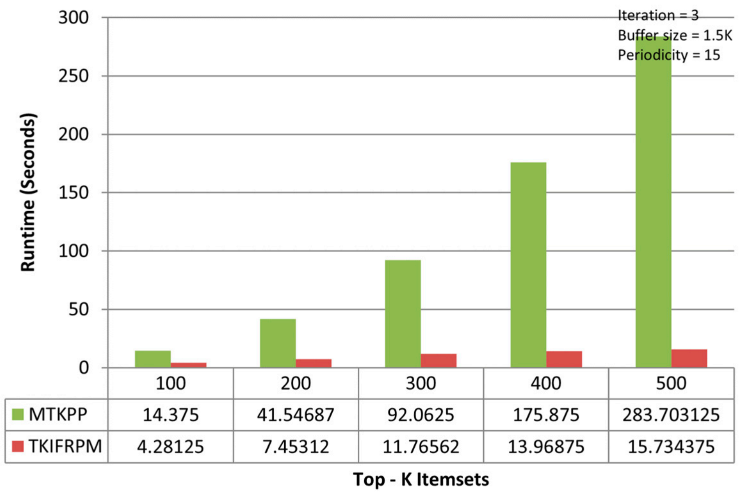
(**a**)

Figure 5. *Cont.*



	100	200	300	400	500
🟩 MTKPP	11.79687	41.85937	97.21875	199.0625	305.703125
🟥 TKIFRPM	5.07812	26.23437	48.89062	72.9375	102.78125

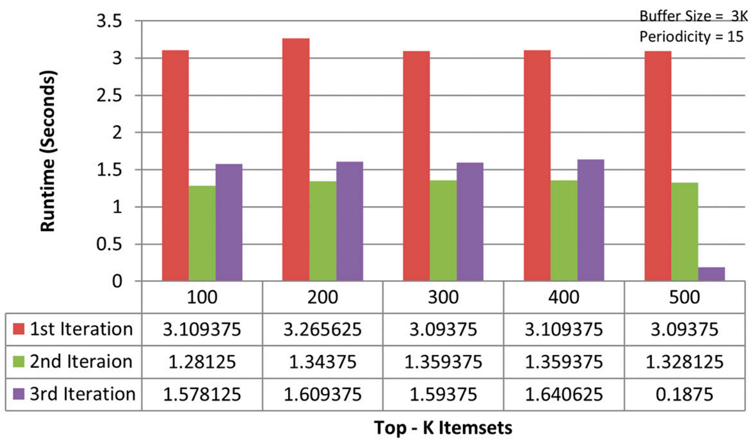
(**b**)



	100	200	300	400	500
🟩 MTKPP	14.375	41.54687	92.0625	175.875	283.703125
🟥 TKIFRPM	4.28125	7.45312	11.76562	13.96875	15.734375

(**c**)

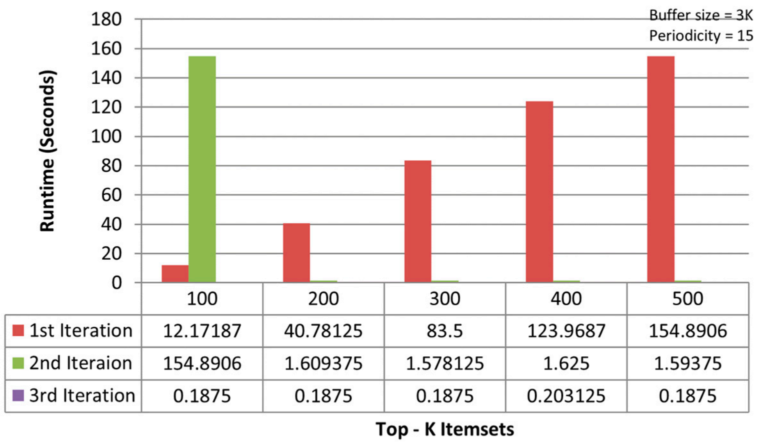
Figure 5. (**a**) Performance trends of the first iteration on the chess dataset. (**b**) Performance trends of the second iteration on the chess dataset. (**c**) Performance trends of the third iteration on the chess dataset.



	100	200	300	400	500
🟥 1st Iteration	3.109375	3.265625	3.09375	3.109375	3.09375
🟩 2nd Iteraion	1.28125	1.34375	1.359375	1.359375	1.328125
🟪 3rd Iteration	1.578125	1.609375	1.59375	1.640625	0.1875

(**a**)

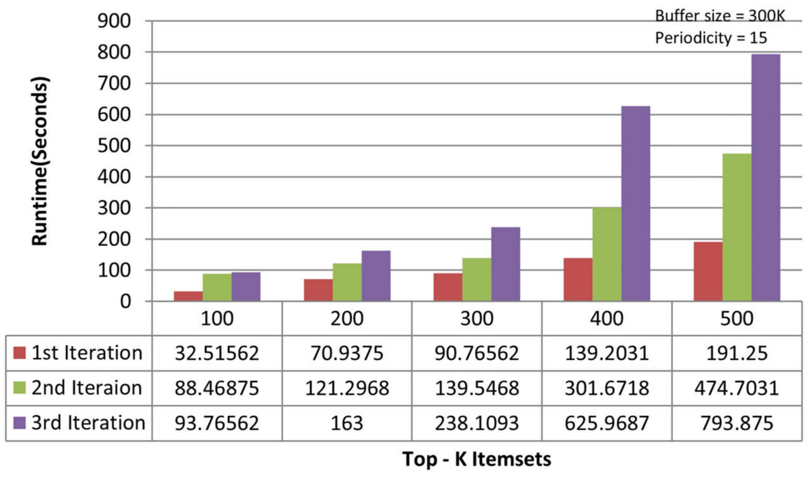
Figure 6. *Cont*.



	100	200	300	400	500
🟥 1st Iteration	12.17187	40.78125	83.5	123.9687	154.8906
🟩 2nd Iteraion	154.8906	1.609375	1.578125	1.625	1.59375
🟪 3rd Iteration	0.1875	0.1875	0.1875	0.203125	0.1875

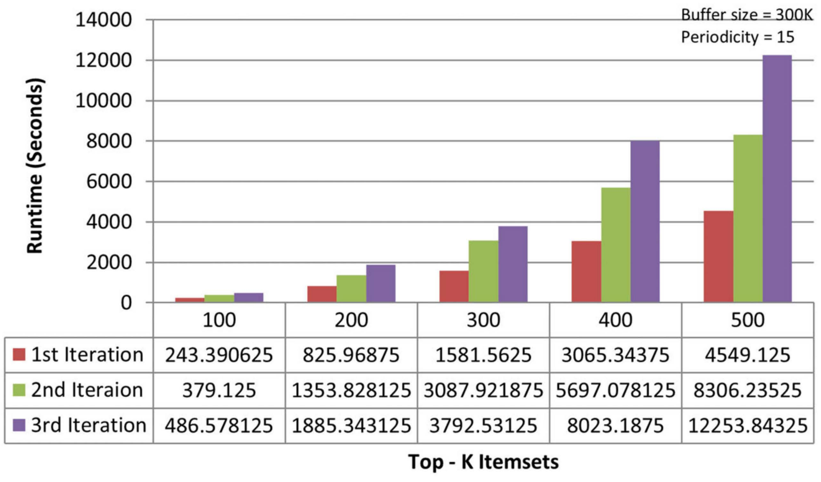
(**b**)

Figure 6. (**a**) Results of three iterations of TKIFRPM on the mushroom dataset. (**b**) Results of three iterations of MTKPP on the mushroom dataset.



	100	200	300	400	500
🟥 1st Iteration	32.51562	70.9375	90.76562	139.2031	191.25
🟩 2nd Iteraion	88.46875	121.2968	139.5468	301.6718	474.7031
🟪 3rd Iteration	93.76562	163	238.1093	625.9687	793.875

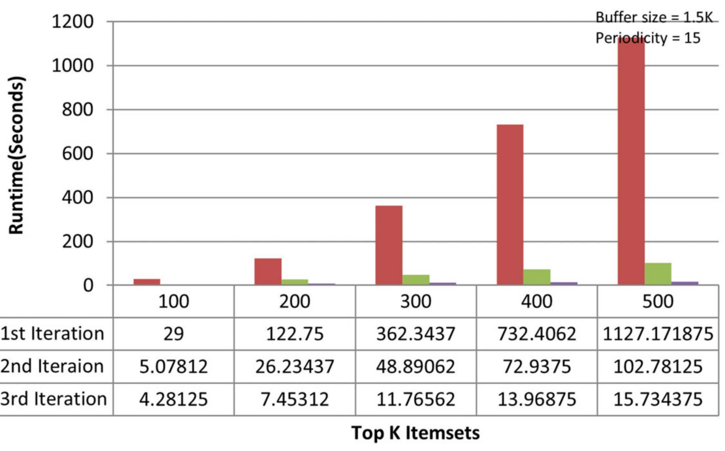
(**a**)



	100	200	300	400	500
🟥 1st Iteration	243.390625	825.96875	1581.5625	3065.34375	4549.125
🟩 2nd Iteraion	379.125	1353.828125	3087.921875	5697.078125	8306.23525
🟪 3rd Iteration	486.578125	1885.343125	3792.53125	8023.1875	12253.84325

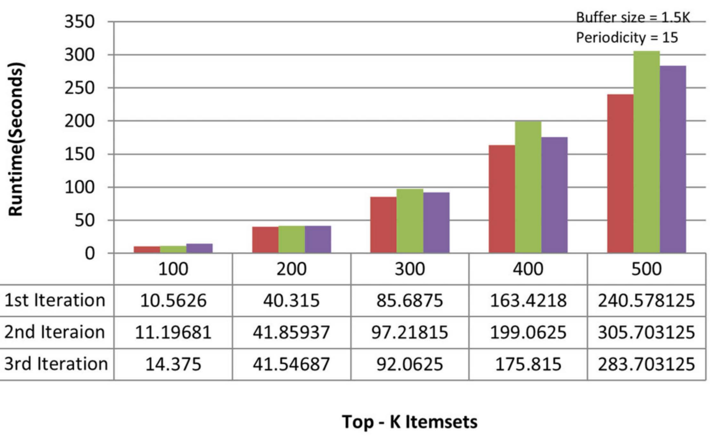
(**b**)

Figure 7. (**a**) Results of three iterations of TKIFRPM on the accident dataset. (**b**) Results of three iterations of MTKPP on the accident dataset.



	100	200	300	400	500
🟥 1st Iteration	29	122.75	362.3437	732.4062	1127.171875
🟩 2nd Iteraion	5.07812	26.23437	48.89062	72.9375	102.78125
🟪 3rd Iteration	4.28125	7.45312	11.76562	13.96875	15.734375

(**a**)



	100	200	300	400	500
🟥 1st Iteration	10.5626	40.315	85.6875	163.4218	240.578125
🟩 2nd Iteraion	11.19681	41.85937	97.21815	199.0625	305.703125
🟪 3rd Iteration	14.375	41.54687	92.0625	175.815	283.703125

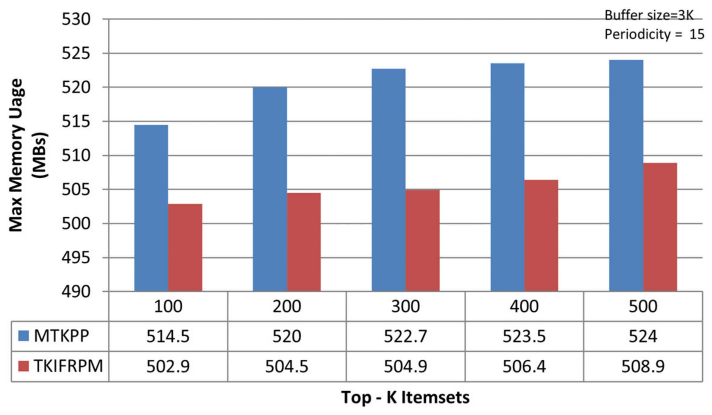
(**b**)

Figure 8. (**a**) Results of three iterations of TKIFRPM on the chess dataset. (**b**) Results of three iterations of MTKPP on the chess dataset.

Conclusively, the performance of TKIFRPM is better than the MTKPP on both small and large datasets, such as mushroom, accident, and chess. It is also observed that on very small dataset, the MTKPP requires only a slight lead in runtime as compared to TKIFRPM. However, as the size of the dataset grows (as in the chess dataset), the performance of MTKPP deteriorates significantly.

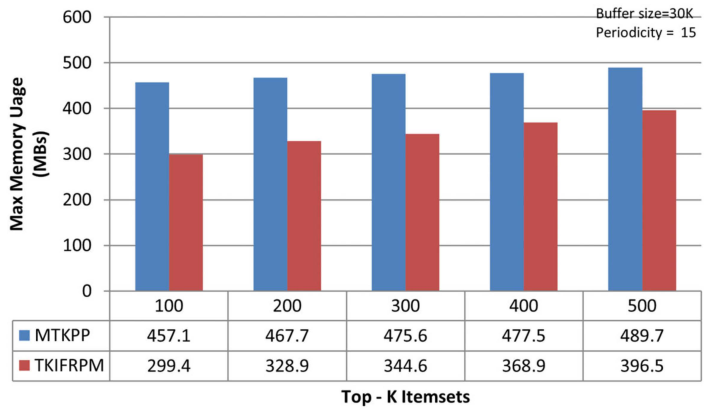
5.3. Memory Usage Evaluation and Comparision

The memory usage evaluations of TKIFRPM and MTKPP methods is performed for the different datasets discussed earlier (i.e., mushroom, accident, and chess) using K, θ_r and T_b values (i.e., K = {100, 200, 300, 400, 500}, K, θ_r = 15 and K, θ_r and T_b = {3K for mushroom, 30K for accident, and 1.5K for chess}. The memory usage evaluations are conducted in same way as runtime evaluations (discussed in Section 5.2) by recording the memory usage of the methods in three iterations for each of the datasets. The average memory usage of the methods for the different values of K and each of the datasets are shown in Figure 9a (mushroom dataset), 9b (accident dataset), and 9c (chess dataset), respectively. The results depict that TKIFRPM is, memory-wise, more efficient than MTKPP in all scenarios. This is because MTKPP is an Apriori-based algorithm and follows no candidate pruning strategy and accepts the Apriori property. Therefore, it keeps large amount of data in the tree in the memory, which simultaneously grows with the increase in a dataset size. Thus, the memory usage of MTKPP is the highest not only compared to TKIFRPM but among all topmost-K frequent pattern mining techniques. On the other hand, TKIFRPM is based on the depth-first strategy and, apart from the threshold-based pruning, the TKIFRPM also follows the look-ahead pruning strategy. This strategy allows TLIFRPM to prune entire subtrees and eliminate them from the mining process. Therefore, it saves the memory which could be wasted in the candidate generations and subsequently used in the support calculation of the TKIFRPM. Conclusively, the TKIFRPM results in less memory usage than the MTKPP in all experiments.



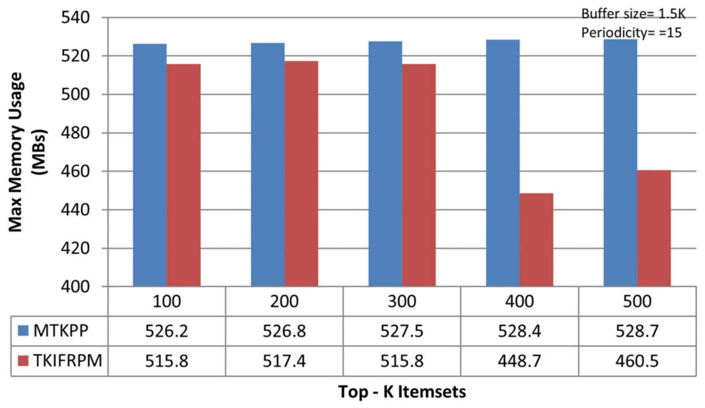
	100	200	300	400	500
🟦 MTKPP	514.5	520	522.7	523.5	524
🟥 TKIFRPM	502.9	504.5	504.9	506.4	508.9

(**a**)



	100	200	300	400	500
🟦 MTKPP	457.1	467.7	475.6	477.5	489.7
🟥 TKIFRPM	299.4	328.9	344.6	368.9	396.5

(**b**)



	100	200	300	400	500
🟦 MTKPP	526.2	526.8	527.5	528.4	528.7
🟥 TKIFRPM	515.8	517.4	515.8	448.7	460.5

(**c**)

Figure 9. (**a**) Memory usage estimations of TKIFRPM and MTKPP on the mushroom dataset. (**b**) Memory usage estimations of TKIFRPM and MTKPP on the accident dataset. (**c**) Memory usage estimations of TKIFRPM and MTKPP on the chess dataset.

6. Conclusions and Future Work

Several top-K frequent regular pattern mining methods are introduced using a support threshold defined by users to determine the number of topmost frequent regular patterns. The top-K frequent regular patterns maintain the top-K list to record the topmost-K frequent regular patterns in descending support order during exploration of a search space. The top-K frequent regular patterns classically perform breadth-first traversal of the search space to generate the required patterns. In top-K frequent regular pattern mining, different support calculation procedures are adopted. However, the available methods have the following limitations: (1) top-K frequent regular pattern mining methods adopt a best-fit strategy for the patterns, perform excessive candidate pattern generation which is inessential for computing, and use no pruning methods to prune the unnecessary candidates before counting the support of the frequent regular itemsets, and (2) top-K frequent regular patterns are applied on offline-datasets, which are not feasible for the online datasets, which continuously remain in the updating stage.

This paper has presented a novel top-K identical frequent regular pattern mining (TKIFRPM) method producing top-K identical frequent regular patterns from the online transactional and incremental datasets. The TKIFRPM maintains a novel performance-wise effective data structure with item support index tables (ISI-tables) to keep summarized information about the updated datasets. The mining operation can be launched at any time interval to discover top-K regular frequent patterns from the ISI-tables at any instant of time. The TKIFRPM explores the search space (arranged in an itemset enumeration tree) in recursive depth-first order. The TKIFRPM applies a novel progressive node's sub-tree pruning strategy to rapidly eliminate a complete infrequent sub-tree from the search space. Hence, the TKIFRPM is computationally efficient compared to the best-fit methods of top-K frequent regular pattern mining, since TKIFRPM does not perform support checking on the nodes of the infrequent subtrees, unlike other top-K frequent regular pattern mining methods. The experimental results have shown that the TKIFRPM method performs better than its counterpart methods (e.g., MTKPP) on the datasets following incremental updates.

In the future, we expect to provide detailed comparisons of the proposed TKIFRPM method with other top-K frequent regular pattern mining methods (i.e., TR-CT, TKRIMPE, TFRC, etc.) using available large-sized online datasets to derive more accurate and widely acceptable results.

Author Contributions: Methodology, S.U.R.; Software, M.A.K., H.U.N. and S.A.; Validation, S.U.R. and M.A.K.; Investigation, S.U.R., H.U.N., S.A. and N.A.; Resources, N.A. and S.K.; Writing—original draft, S.U.R. and N.A.; Writing—review & editing, S.K.; Supervision, S.K.; Project administration, S.K.; Funding acquisition, N.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research has received no external funding.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created.

Acknowledgments: The authors of this paper would like to offer thanks for the research facilities provided by the Huaiyin Institute of Technology, China.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Agrawal, R.; Imieliński, T.; Swami, A. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*; ACM: New York, NY, USA, 1993.
2. Amphawan, K.; Lenca, P.; Surarerks, A. Mining top-k periodic-frequent pattern from transactional databases without support threshold. In Proceedings of the International Conference on Advances in Information Technology, Bangkok, Thailand, 1–5 December 2009; Springer: Berlin/Heidelberg, Germany, 2009.
3. Ashraf, J.; Habib, A.; Salam, A. Top-K Miner: Top-K identical frequent itemsets discovery without user support threshold. *Knowl. Inf. Syst.* **2016**, *48*, 741–762.

4. Salam, A.; Khayal, M. Mining top-k frequent patterns without minimum support threshold. *Knowl. Inf. Syst.* **2012**, *30*, 57–86. [CrossRef]
5. Fournier-Viger, P.; Lin, C.W.; Duong, Q.H.; Dam, T.L.; Ševčík, L.; Uhrin, D.; Voznak, M. PFPM: Discovering periodic frequent patterns with novel periodicity measures. In *Proceedings of the 2nd Czech-China Scientific Conference 2016*; IntechOpen: London, UK, 2017.
6. Shyu, M.-L.; Haruechaiyasak, C.; Chen, S.C.; Zhao, N. Collaborative filtering by mining association rules from user access sequences. In Proceedings of the International Workshop on Challenges in Web Information Retrieval and Integration, Tokyo, Japan, 8–9 April 2005.
7. Tanbeer, S.K.; Ahmed, C.F.; Jeong, B.S.; Lee, Y.K. Discovering periodic-frequent patterns in transactional databases. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*; Springer: Berlin/Heidelberg, Germany, 2009.
8. Kiran, R.U.; Reddy, P.K. Mining rare periodic-frequent patterns using multiple minimum supports. In Proceedings of the 15th International Conference on Management of Data, Providence, RI, USA, 29 June–2 July 2009; pp. 7–8.
9. Surana, A.; Kiran, R.U.; Reddy, P.K. An efficient approach to mine periodic-frequent patterns in transactional databases. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*; Springer: Berlin/Heidelberg, Germany, 2011.
10. Kiran, R.U.; Kitsuregawa, M. Discovering quasi-periodic-frequent patterns in transactional databases. In *International Conference on Big Data Analytics*; Springer: Cham, Switzerland, 2013.
11. Amphawan, K.; Surarerks, A.; Lenca, P. Mining periodic-frequent itemsets with approximate periodicity using interval transaction-ids list tree. In Proceedings of the 2010 Third International Conference on Knowledge Discovery and Data Mining, Phuket, Thailand, 9–10 January 2010.
12. Amphawan, K.; Lenca, P.; Surarerks, A. Efficient mining top-k regular-frequent itemset using compressed tidsets. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*; Springer: Berlin/Heidelberg, Germany, 2011.
13. Chen, M.-C.; Chiu, A.-L.; Chang, H.-H. Mining changes in customer behavior in retail marketing. *Expert Syst. Appl.* **2005**, *28*, 773–781. [CrossRef]
14. Rehman, S.U.; Alnazzawi, N.; Ashraf, J.; Iqbal, J.; Khan, S. Efficient Top-K Identical Frequent Itemsets Mining without Support Threshold Parameter from Transactional Datasets Produced by IoT-Based Smart Shopping Carts. *Sensors* **2022**, *22*, 8063. [CrossRef] [PubMed]
15. Iqbal, S.; Shahid, A.; Roman, M.; Khan, Z.; Al-Otaibi, S.; Yu, L. TKFIM: Top-K frequent itemset mining technique based on equivalence classes. *PeerJ Comput. Sci.* **2021**, *7*, e385. [CrossRef] [PubMed]
16. Kiran, R.U.; Reddy, P. Mining periodic-frequent patterns with maximum items' support constraints. In *Proceedings of the Third Annual ACM Bangalore Conference*; ACM: New York, NY, USA, 2010.
17. Fournier-Viger, P.; Wu, Y.; Dinh, D.T.; Song, W.; Lin, J.C.W. Discovering periodic high utility itemsets in a discrete sequence. In *Periodic Pattern Mining*; Springer: Singapore, 2021; pp. 133–151.
18. Dinh, D.-T.; Le, B.; Fournier-Viger, P.; Huynh, V.N. An efficient algorithm for mining periodic high-utility sequential patterns. *Appl. Intell.* **2018**, *48*, 4694–4714. [CrossRef]
19. Amphawan, K.; Lenca, P.; Surarerks, A. Mining top-k regular-frequent itemsets using database partitioning and support estimation. *Expert Syst. Appl.* **2012**, *39*, 1924–1936. [CrossRef]
20. Tagmatcha, B.; Amphawan, K. Mining top-k frequent-regular itemsets from incremental transactional database. In Proceedings of the 2018 5th International Conference on Advanced Informatics: Concept Theory and Applications (ICAICTA), Krabi, Thailand, 14–17 August 2018.
21. Amphawan, K.; Lenca, P. Mining top-k frequent-regular closed patterns. *Expert Syst. Appl.* **2015**, *42*, 7882–7894. [CrossRef]
22. Ravikumar, P.; Likhitha, P.; Raj, B.V.V.; Uday Kiran, R.; Watanobe, Y.; Zettsu, K. Efficient Discovery of Periodic-Frequent Patterns in Columnar Temporal Databases. *Electronics* **2021**, *10*, 1478. [CrossRef]
23. Li, H.-F.; Shan, M.-K.; Lee, S.-Y. DSM-FI: An efficient algorithm for mining frequent itemsets in data streams. *Knowl. Inf. Syst.* **2008**, *17*, 79–97. [CrossRef]
24. Li, H.-F. MHUI-max: An efficient algorithm for discovering high-utility itemsets from data streams. *J. Inf. Sci.* **2011**, *37*, 532–545. [CrossRef]
25. Gouda, K.; Zaki, M.J. Genmax: An efficient algorithm for mining maximal frequent itemsets. *Data Min. Knowl. Discov.* **2005**, *11*, 223–242. [CrossRef]