The Journal of Supercomputing (2023) 79:7877–7899
https://doi.org/10.1007/s11227-022-04976-5

CC-IFIM: an efficient approach for incremental frequent itemset mining based on closed candidates

Maged Magdy[1] · **Fayed F. M. Ghaleb**[1] · **Dawlat A. El A. Mohamed**[1] · **Wael Zakaria**[1]

Accepted: 22 November 2022 / Published online: 8 December 2022

Abstract

Frequent itemset mining (FIM) is the crucial task in mining association rules that finds all frequent k-itemsets in the transaction dataset from which all association rules are extracted. In the big-data era, the datasets are huge and rapidly expanding, so adding new transactions as time advances results in periodic changes in correlations and frequent itemsets present in the dataset. Re-mining the updated dataset is impractical and costly. This problem is solved via incremental frequent itemset mining. Numerous researchers view the new transactions as a distinct dataset (partition) that may be mined to obtain all of its frequent item sets. The extracted local frequent itemsets are then combined to create a collection of global candidates, where it is possible to estimate the support count of the combined candidates to avoid re-scanning the dataset. However, these works are hampered by the growth of a huge number of candidates, and the support count estimation is still imprecise. In this paper, the Closed Candidates-based Incremental Frequent Itemset Mining approach, or CC-IFIM, has been proposed to decrease candidate generation and improve the accuracy of the global frequent itemsets that are retrieved. The proposed approach is able to prune several produced candidates in earlier steps before performing any further computations. To improve the accuracy of the computation of the support count of the produced candidates, the similarity between partitions has been evaluated using just the local closed candidates rather than all candidates. The experimental findings demonstrated that the CC-IFIM approach is superior to its competitors in terms of efficiency and accuracy.

Keywords Frequent itemsets · Incremental frequent itemsets · Similarity measurements · Closed frequent itemsets

Maged Magdy, Fayed F. M. Ghaleb, Dowlat A. El A. Mohamed and Wael Zakaria have contributed equally to this work.

✉ Wael Zakaria
Wael.Zakaria@sci.asu.edu.eg

Extended author information available on the last page of the article

1 Introduction

Association rule mining (ARM) is one of the key tasks in data mining. It discovers interesting correlations among set of items in large transaction datasets. ARM has developed into a powerful tool with immense potential and wide applications such as market basket analysis, medical diagnosis, bioinformatics, fraud detection and the internet of things.

Formally, consider $\mathcal{I} = \{I_1, I_2, \dots, I_m\}$ is a set of m items and a dataset $\mathcal{D} = \{T_1, T_2, \dots, T_n\}$ is a set of n transactions, where $T_i \subseteq \mathcal{I}$, ARM is a process of mining all strong association rules (AR) by performing the following two tasks [1]. Firstly, generate all frequent k-itemsets ($\mathcal{FI}$); $k = 1, 2, \dots, m$. The k-itemset X (a set of k items) is called frequent if $\sup(X) \geq \text{minsup} \times n$; $\sup(X) = |T_i;X \subseteq T_i|$ and minsup is give threshold (say 80%). Secondly, from the extracted $\mathcal{FI}$, the strong association rules (AR) have been mined; an association $R \in AR$ on the form $R : X \rightarrow Y$; $X \cup Y \in \mathcal{FI}$, $X \cap Y = \phi$. The rule R is strong if $\text{sup}p(R) > \text{minsup}$ and $\text{conf}(R) > \text{minconf}$; $\sup(R) = \frac{\sup(X \cup Y)}{n}$ and $\text{conf}(R) = \frac{\sup(X \cup Y)}{\sup(X)}$, where minsup and minconf are given thresholds.

Frequent itemsets mining (FIM) is the crucial task in AR that finds all $\mathcal{FI}$ among a set of items in $\mathcal{D}$. The exponential complexity of FIM attracts researchers to develop new algorithms to improve it. In huge datasets, the traditional algorithms, such as Apriori [1], FP-growth [9], Eclat [19] and Quick-Apriori [17] require exponential time and memory for mining all $\mathcal{FI}$. In several applications, transaction datasets are growing continuously, so adding incremental transactions to the original dataset requires repeating the mining processes. These kinds of algorithms are considered static because they have no way to merge the new transactions without re-mining the updated datasets. Due to their exponential complexity, traditional mining approaches fail to address the continuously increasing dataset.

To overcome these obstacles, incremental frequent itemset mining (IFIM) is utilized. There are two categories for IFIM approaches: the Apriori-based [4, 5, 20] and the tree-based approaches [9, 10, 13, 15]. Apriori-based approaches suffer from the *I*/*O* overhead of scanning the dataset many times and the cost of computation and memory of generating a huge number of candidates. As well as FP tree-based approaches suffer from complex operations for adjusting trees [18].

Nowadays where the number of transactions is very huge and continuously changing, all the above approaches are not applicable because they cannot get the results in an efficient way. Therefore, some approaches [3, 18] introduced approximated solutions for IFIM. These approaches can be utilized in mining *FI* as well. For IFIM, the FPMSIM approximated approach [18] considers the new transactions as a separate dataset to be mined, from which all local $\mathcal{FI}$ are mined. Subsequently, the extracted local $\mathcal{FI}$s of both the original and the incremental datasets are combined together to produce global candidates. The support count of each candidate is estimated using a kind of statistical model such as Jaccard similarity between the produced candidates of the original and incremental datasets. Although the approach [18] has good results, it produces many candidate itemsets that can be pruned before any further computation. As well as, the estimating of the support count of all candidates is imprecise, which causes the loss of global $\mathcal{FI}$. The purpose of this paper is to address the issues raised in the FPMSIM [18]. CC-IFIM, or closed candidates-based incremental frequent itemset mining, is the name of the proposed approach. The suggested approach makes use of a pruning mechanism to eliminate certain early, pointless candidates. The CC-IFIM approach measures the similarity between just the closed candidates of the original and incremental datasets rather than all produced candidates in order to increase the estimated accuracy of the support count of all candidates. The experimental results on five benchmark datasets showed that the CC-IFIM approach is more efficient and accurate than the FPMSIM approach.

The remainder of this paper is organized as follows: The related works are discussed in Sect. 2, where FPMSIM, which is the source of CC-IFIM, is given specific attention. Section 3 introduces the proposed approach CC-IFIM. Section 4 discusses the experimental results. Finally, Sect. 5 concludes the paper.

2 Related work

In this section, the incremental frequent itemsets mining is classified into two categories: exact approaches that find all $\mathcal{FI}$ (Sect. 2.1), and approximated approach that tries to find approximated $\mathcal{FI}$ denoted by $\mathcal{FI}_{\text{approx}}$; $\mathcal{FI}_{\text{approx}} \subseteq \mathcal{FI}$ (Sect. 2.2).

2.1 Exact approaches

The exact approach can be classified into two major categories: Apriori-based and FP-tree-based as shown in the following two subsections:

2.1.1 Apriori-based

Apriori-based approach firstly scans the incremental dataset to extract its $\mathcal{FI}$s, then re-scanning the original dataset to get the exact support of each candidate, as in FUP [4], FUP2 [5]. However, re-scanning the original dataset causes I/O cost overload.

In 2001, Zhou et al. [20] developed the MAAP algorithm, which utilizes the characteristics of Apriori to improve the overall performance. In the Apriori algorithm, high-level itemsets are joined by low-level itemsets, where in the Apriori algorithm uses high-level itemsets to induct low-level itemsets. MAAP compares frequent patterns with new transactions to generate new association rules. In addition, MAAP can improve the performance of FUP [4]; however, the problem of re-scanning the dataset still exists.

2.1.2 FP tree-based

Rather than employing the generate-and-test strategy of Apriori-based algorithms, the tree-based framework constructs an extended prefix-tree structure, called Frequent Pattern tree (FPtree), to capture the content of the transaction database [9].

In 2008, Hong et al. [10] proposed a fast and effective method for updating the structure of the FP-tree, called the FUFP (Fast Updated Frequent Pattern) algorithm, in which only frequent items are saved in the FUFP-tree. The FUFP algorithm can quickly update and modify the tree by dividing items. When the original large item becomes smaller, it will be directly deleted from the FUFP-tree. Instead, as the original item gets larger, it is added to the end of the head table in descending order. But it needs to re-scan the original dataset to find the transactions of the newly enlarged items and insert them into the FUFP tree.

In 2014, two remarkably efficient algorithms are introduced: FIN [7] and PrePost+ [8] with POC tree and PPC tree, respectively. These two structures are prefix trees and similar to FP-tree, Moreover, both algorithms employ two additional data structures called Nodeset and N-list, respectively, to significantly improve mining speed. However, N-list consumes a lot of memory, and for some datasets, Nodeset's cardinality grows significantly [6].

The preceding issue was resolved in 2016 by Deng, by proposing an algorithm called dFIN [6] that is based on a new data structure called DiffNodeset instead of Nodeset. In contrast to Nodeset, the DiffNodeset of each k-itemset ($k \geq 3$) is extracted by the difference between the DiffNodesets of two (k-1)- itemsets. Numerous investigations demonstrate that DiffNodeset's cardinality is lower than Nodeset's. As a result, the dFIN algorithm is quicker than Nodeset-based algorithms. However, the calculation of the difference between two DiffNodesets can be time-consuming for some datasets.

In 2017, Huynh et al. [11] proposed a tree structure IPPC (Incremental Pre-Post-Order Coding) algorithm which supports incremental tree construction, and an algorithm for incrementally mining frequent itemsets, IFIN (Incremental Frequent Itemsets Nodesets). Through experiments, algorithm IFIN has demonstrated its superior performance compared to FIN [7] and PrePost+ [8]. However, in the case of datasets comprising a large number of distinct items but just a small percentage of frequent items for a certain support threshold, we investigate that IPPC tree becomes to lose its advantage in running time and memory for its construction compared to other trees such as POC and PPC of algorithms FIN and PrePost+.

In 2021, Satyavathi et al. [14] proposed the FIN_INCRE algorithm by enhancing FIN algorithm [7] for efficient mining of incremental association rules which require scanning of the original dataset only once. After scanning the dataset, it generates POC-Tree and from which it produces item sets that occur frequently. Then, they are used to generate association rules. When some new instances are inserted into the original dataset, the algorithm scans only the newly added instances. Then, it updates POC-Tree and frequent itemsets before actually updating the mined association rules.

However, FP tree-based approaches suffer from complex operations for adjusting FP-tree.

Algorithm 1 *FPMSIM* Algorithm

```
Input: Original dataset $\mathcal{D}$, $minsup$
Output: All frequent itemsets $\mathcal{FI}$
begin
Divide $\mathcal{D}$ into partitions/scales $\{P_1, P_2, \ldots, P_k\}$
for each $P_j$, using FP-growth, mine $\mathcal{FI}^j$
$\mathcal{C}_{\mathcal{D}} = \bigcup_{j=1}^{k} \mathcal{FI}^j$
Build a matrix $S$, in which rows represent the support counts of global
    candidates at all partitions.
Calculate the Jaccard Matrix $M \in \mathcal{R}^{k \times k}$, each $M(i,j)$ represents the
    similarity between partitions i and j, where $M(i,j) = \frac{|\mathcal{FI}^i \cap \mathcal{FI}^j|}{|\mathcal{FI}^i \cup \mathcal{FI}^j|}$
for each S(i,j)==0, estimate its support count according to
```

$$S(i,j) = \begin{cases} \frac{1}{m} \sum_{j'=1, j' \neq j}^{k} M(i,j') \times S(i,j') & \textbf{if } S(i,j) \leq minsup * |P_j| \\ minsup * |P_j| & otherwise \end{cases} \tag{1}$$

```
for each candidate $c \in \mathcal{C}_{\mathcal{D}}$ at index $i$, calculate all its global support which
    is $\sum_{j=1}^{j=k} S(i,j)$.
add $C$ to $\mathcal{FI}$ if $supp(c) \geq minsup$
```

2.2 Approximated approaches

Regardless of using Apriori [2] or FP-Tree [9], the key of these approaches are reducing both I/O cost and a complex generating of FP-trees.

In 2017, Li et al. [12] proposed a three-way decision update pattern (TDUP) approach along with a synchronization mechanism for this issue. With two support-based measures, all possible itemsets are divided into positive, boundary, and negative regions. TDUP efficiently updates frequent itemsets and reduces the cost of re-scanning. However, TDUP may miss some potential frequent itemsets with incremental data updates.

In 2021, Xun et al. [18], developed an approach for incremental frequent itemsets mining based on frequent pattern tree and multi-scale called FPMSIM. A partitioning-based FPMSIM is valid for not only parallel programming of mining all $\mathcal{FI}$'s, but also for addressing the problem of incremental frequent itemsets mining. In general, As shown in Algorithm 1, FPMSIM divides the dataset into scales or partitions using a multi-scale concept in which each partition may have the same essence and characteristics. Using FP-growth [2] and for each partition, the local frequent itemsets $\mathcal{FI}$ have been extracted. The union of all local frequent itemsets is considered as global candidates *C*. To estimate the support of global candidates, Jaccard method [16] is used for measuring the similarities between scales or partitions. Finally, the global frequent itemsets are candidates whose candidates with support is greater than or equal to global minsup. In big data, this approach is efficient and scalable which produced an acceptable $\mathcal{FI}_{\text{approx}}$ compared with the exact $\mathcal{FI}$, however, it produces many candidate itemsets that can be pruned before any further computation. Additionally, the estimating of the support count of all candidates is imprecise, which causes the loss of global $\mathcal{FI}$. Consequently, miss some potential frequent itemsets.

3 The proposed approach CC-IFIM

In this section, an improved approach of FPMSIM approach [18] has been introduced. This approach has been called Closed Candidates-based Frequent Itemset Mining and denoted(CC-IFIM). Although FPMSIM and CC-IFIM have some common phases there are a lot of differences between them, The core difference is ignoring the division of the dataset into scales, which requires an additional cost for getting further information as in FPMSIM. This kind of division is not applicable in most cases of real applications.

Algorithm 2 $\mathcal{CC} - IFIM$ Algorithm

```
Input: Original dataset $\mathcal{D}$ and $minsup$
Output: All frequent itemsets $\mathcal{FI}$
begin
Divide $\mathcal{D}$ into partitions $\{P_1, P_2, \ldots, P_k\}$
for each $P_j$, mine $\mathcal{FI}^j$
$\mathcal{C}_{\mathcal{D}} = \bigcup_{j=1}^{k} \mathcal{FI}^j$
Build a matrix S, in which rows represent the support counts of global candidates at all partitions.
Using $S$ and $\mathcal{C}_{\mathcal{D}}$, extract the closed candidates $\mathcal{CC}_{\mathcal{D}}$.
Build a matrix $S_{closed}$, from which the similarity matrix $M_{\mathcal{CC}} \in \mathcal{R}^{k \times k}$ is calculated. Each M(i,j) represents the similarity between partitions i and j. M(i,j) can be calculated using Jaccard, Dice matrix, or cosine similarity.
Using the original matrix $S$ and for each $S(i,j) = 0$, apply the pruning step if applicable, then estimate its support count according to
    $S(i,j) = \begin{cases} sup_{app} = \frac{1}{m(i)} \sum_{j'=1,j' \neq j}^{k} M(j',j) \times S(i,j') & \textbf{if } sup_{app} \leq minsup_j \\ minsup_j & otherwise \end{cases} \qquad (2)$
for each candidate $c \in \mathcal{C}_{\mathcal{D}}$ at index $i$, calculate all its global support which is $\sum_{j=1}^{j=k} S(i,j)$.
add $C$ to $\mathcal{FI}$ if $supp(c) \geq minsup$
```

Consider a transaction dataset $\mathcal{D}$ and minimum support count minsup$_{\mathcal{D}}$. Algorithm 2 shows the steps of extracting $\mathcal{FI}$, where CC-IFIM works as follows:

1. Divide $\mathcal{D}$ into partitions $\mathcal{P}$={$P_1, P_2, \dots, P_k$} where:
   - $\bigcup_{j=1}^{k} P_j = \mathcal{D}$; k the number of partitions.
   - For $i \neq j$, $P_i \cap P_j = \emptyset$.
   - Each P_j has its own minsup called minsup$_{P_j}$.
2. Using Apriori [2] or FP-tree [9] algorithm, $\forall$ $P_j \in \mathcal{P}$ mine its frequent itemsets $\mathcal{FI}^j$ ={f_1^j,f_2^j, … ,$f_{q_j}^j$}. Where, each frequent set $f \in \mathcal{FI}^j$ is a local frequent itemset of $\mathcal{D}$; supp$_j$(f) ≥ minsup$_{P_j}$, while it is called a global candidate itemset of $\mathcal{D}$.
3. Generate a set of candidates denoted by $\mathcal{C}_{\mathcal{D}}$; $\mathcal{C}_{\mathcal{D}}$= $\bigcup_{j=1}^{k} \mathcal{FI}^j$. Let q is the number of generated candidates (i.e., q =| $\mathcal{C}_{\mathcal{D}}$ |). Each itemset $c \in \mathcal{C}_{\mathcal{D}}$ is called a candidate itemset of $\mathcal{D}$.
4. Build a matrix $S \in \mathbb{R}^{q \times k}$; in which the rows represent the candidates c_i; $i = 1, 2, \dots, q$ and columns represent the partitions p_j; $j = 1, 2, \dots, k$. The element $S(i, j)$; $1 \leq i \leq q, 1 \leq j \leq k$ of the matrix S is determined as follows:

$$S(i,j) = \begin{cases} \sup_j(c_i) & \text{if } c_i \in \mathcal{FI}^j \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

5. An extra column $m \in \mathbb{R}^{q \times 1}$ with dimension q is associated to the matrix S as follows:

$$m(i) =\mid p^{+}(i) \mid, i = 1, 2, \dots, q \tag{4}$$

   where $p^{+}(i) = \{j = S(i,j) \neq 0\}$ gives the partition j in which c_i is frequent (i.e., $\sup_j(c_i) \geq 0$).
6. *Extract closed candidates* Using the candidates' $\mathcal{C}_D$ and the matrix S, which contains local frequent itemsets, extract only the closed candidates' $\mathcal{CC}_D$ which contains all local closed frequent itemset (Definition 1).

Definition 1 (local closed frequent itemset) A local frequent itemset $c_i \in \mathcal{C}_D$ is a local closed frequent itemset, if there is no local frequent itemset $c_{i'} \in \mathcal{C}_D$; $c_{i'} \supset c_i$ and $S(i,j) = S(i',j)$;$j = 1, 2, \dots, k$.

7. Using $\mathcal{CC}_D$, build its corresponding new matrix called $S_{\text{closed}} \in \mathbb{R}^{q_c \times k}$;$q_c$ =| $\mathcal{CC}_D$ |, $q_c \leq q$, and $S_{\text{closed}}(i,j) = S(i,j)$; $cc_i \in \mathcal{CC}_D$ and $j = 1, 2, \dots, k$. In CC-IFIM, the matrix S_{closed} is used for creating the similarity matrix $M_{\mathcal{CC}}$ as shown in the next step instead of the matrix S as in FPMSIM.
8. Using $\mathcal{CC}_D$, build a symmetric similarity matrix $M_{\mathcal{C}}\mathcal{C} \in \mathbb{R}^{k \times k}$, where $M_{\mathcal{C}}\mathcal{C}(j,j')$ is the similarity between the partitions P_j and $P_{j'}$ for all $j,j' = 1, 2, \dots, k$ and $j \neq j'$. The three versions of similarity such as Jaccard, Dice, or Cosine [16] has

been used from which the more accurate method will be chosen as a suitable similarity measurement. The similarity Jaccard matrix is calculated as follows:

$$M_{CC}(j,j') = \frac{|c_i \in \mathcal{CC}_D; S_{\text{closed}}(i,j) \neq 0 \text{ and } S_{\text{closed}}(i,j') \neq 0|}{|c_i \in \mathcal{CC}_D; S_{\text{closed}}(i,j) \neq 0 \; or \; S_{\text{closed}}(i,j') \neq 0|} \tag{5}$$

Simply, you can read $S_{\text{closed}}(i,j) \neq 0$ by the candidate c_i is frequent at partition j. Therefore, $M_{CC}(j,j')$ is the number of closed candidates that are frequent in both partitions j and j' divided by the number of candidates that are frequent in at least one partition. In CC-IFIM, we discovered that the Jaccard similarity matrix is not the suitable method for measuring the similarity between partitions as shown in the experimental result section. The similarity Dice matrix is calculated as follows:

$$M_{CC}(j,j') = 2\frac{|c_i \in \mathcal{CC}_D; S_{\text{closed}}(i,j) \neq 0 \; and \; S_{\text{closed}}(i,j') \neq 0|}{|c_i \in \mathcal{CC}_D; S_{\text{closed}}(i,j) \neq 0| + |c_i \in \mathcal{CC}_D; S_{\text{closed}}(i,j') \neq 0|} \tag{6}$$

The Cosine similarity method is calculated as follows:

$$M_C\mathcal{C}(j,j') = \frac{|c_i \in \mathcal{CC}_D; S_{\text{closed}}(i,j) \neq 0 \; and \; S_{\text{closed}}(i,j') \neq 0|}{\sqrt{|c_i \in CC_D; S_{\text{closed}}(i,j) \neq 0|}.\sqrt{|c_i \in \mathcal{CC}_D; S_{\text{closed}}(i,j') \neq 0|}} \tag{7}$$

Based on any version of similarity method, obviously $M_C\mathcal{C}(j,j) = 1$, $M_C\mathcal{C}(j,j') = M_C\mathcal{C}(j',j)$. The value or coefficient $M_C\mathcal{C}(j,j')$ ranges between 0 and 1. $M_C\mathcal{C}(j,j') = 1$ means the partitions j and j' are similar, while $M_C\mathcal{C}(j,j') = 0$ means the partitions j and j' are dissimilar and there is no intersection between them.

9. *Pruning step* After creating matrix $M_C\mathcal{C}$, we back again to $\mathcal{C}_D$ and its corresponding matrix S. Before estimating the global support, a new hypothesis was used in earlier step to predict whether the candidate c_i is globally frequent or not. Therefore, $\forall c_i \in \mathcal{C}_D$, Critical_sup can be calculated as follows:

$$\text{Critical\_sup}(c_i) = \sum_{j=1}^{k} S'(i,j) \tag{8}$$

where

$$S'(i,j) = \begin{cases} \text{minsup}_j - 1 & \text{if } S(i,j) = 0 \\ S(i,j) & \text{otherwise} \end{cases} \tag{9}$$

Since, at $S(i,j) = 0$, c_i is infrequent at partition j. Therefore, the most expected support value should be $\text{minsup}_j - 1$ that is the largest number less than minsup_j. Remove row i from table S, consequently remove candidate c_i from $\mathcal{C}_D$ if its $\text{Critical\_sup}(c_i) < \text{minsup}$. This step reduces the computation and memory cost for building the matrix S.

10. To estimate the support of each candidate $c_i \in \mathcal{C}_D$ at any partition j with $S(i,j) = 0$. The estimated support $S(i,j)$ is computed according to the following equation:

$$S(i,j) = \begin{cases} \text{sup}_{\text{app}} = \frac{1}{m(i)} \sum_{j'=1,j'\neq j}^{k} M(j',j) \times S(i,j') & \text{if } \text{sup}_{\text{app}} \leq \text{minsup}_j \\ \text{minsup}_j & \text{otherwise} \end{cases} \tag{10}$$

11. For each candidate $c_i \in \mathcal{C}_D$, estimate a global support using the following equation:

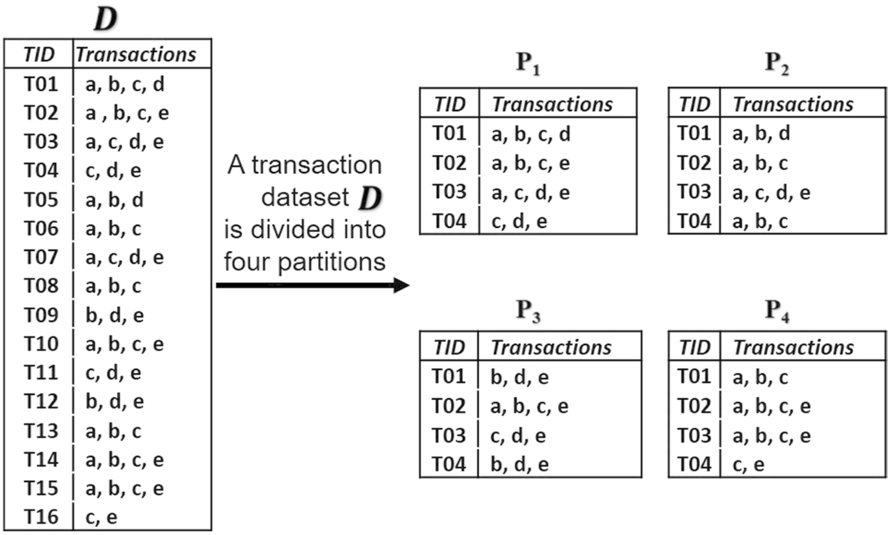
$$\text{sup}(c_i) = \sum_{j=1}^{k} S(i,j) \tag{11}$$

12. Add c_i to the global frequent itemsets $\mathcal{FI}$, if c_i satisfies the following condition:

$$\text{sup}(c_i) \geq \text{minsup}_D \tag{12}$$

The following example is used for discussing each step of CC-IFIM for extracting $\mathcal{FI}$ and the differences between FPMSIM and CC-IFIM. Consider the transaction dataset $\mathcal{D}$ (Fig. 1) that contains sixteen transactions ($n = 16$) and five items $\mathcal{I} = \{a, b, c, d, e\}$ ($|\mathcal{I}| = 5$). And $\text{minsup} = 50\%\ \textit{of}\ \mathcal{D} = 0.5 * 16 = 8$. CC-IFIM works as follows:

1. Firstly, $\mathcal{D}$ is divided into four partitions ($k = 4$): P_1, P_2, P_3, and P_4 (Fig. 1). Where $\text{minsup}_{P_j} = 50\%\textit{of}\ P_j = 0.5 \times 4 = 2; j = 1, 2, 3, 4$.



D

TID	Transactions
T01	a, b, c, d
T02	a , b, c, e
T03	a, c, d, e
T04	c, d, e
T05	a, b, d
T06	a, b, c
T07	a, c, d, e
T08	a, b, c
T09	b, d, e
T10	a, b, c, e
T11	c, d, e
T12	b, d, e
T13	a, b, c
T14	a, b, c, e
T15	a, b, c, e
T16	c, e

A transaction dataset $\boldsymbol{D}$ is divided into four partitions

$\mathbf{P_1}$

TID	Transactions
T01	a, b, c, d
T02	a, b, c, e
T03	a, c, d, e
T04	c, d, e

$\mathbf{P_2}$

TID	Transactions
T01	a, b, d
T02	a, b, c
T03	a, c, d, e
T04	a, b, c

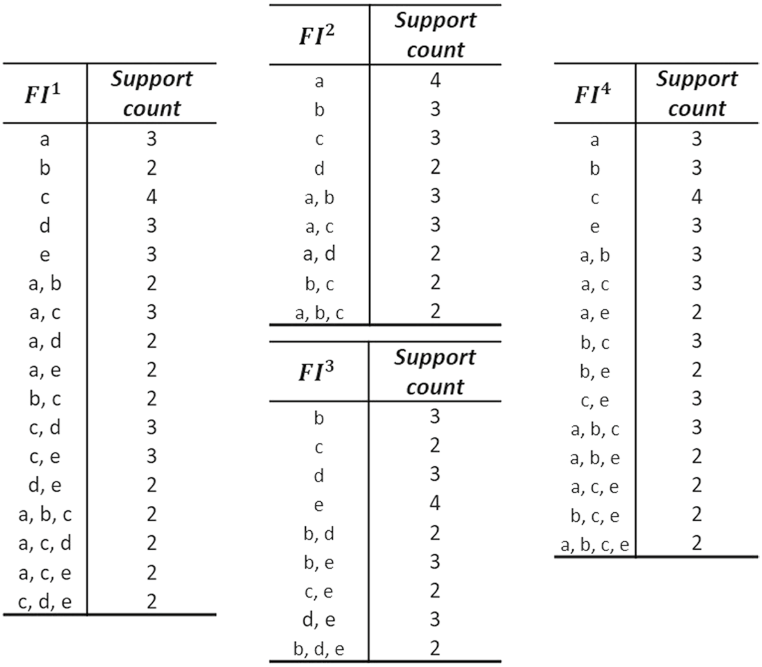
$\mathbf{P_3}$

TID	Transactions
T01	b, d, e
T02	a, b, c, e
T03	c, d, e
T04	b, d, e

$\mathbf{P_4}$

TID	Transactions
T01	a, b, c
T02	a, b, c, e
T03	a, b, c, e
T04	c, e

Fig. 1 Transaction dataset $\mathcal{D}$ and their partitions



FI^1	Support count
a	3
b	2
c	4
d	3
e	3
a, b	2
a, c	3
a, d	2
a, e	2
b, c	2
c, d	3
c, e	3
d, e	2
a, b, c	2
a, c, d	2
a, c, e	2
c, d, e	2

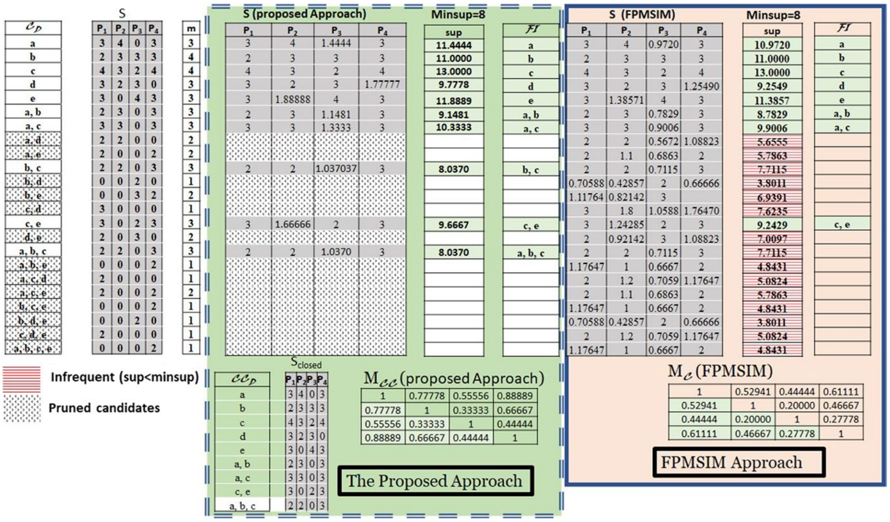
FI^2	Support count
a	4
b	3
c	3
d	2
a, b	3
a, c	3
a, d	2
b, c	2
a, b, c	2

FI^3	Support count
b	3
c	2
d	3
e	4
b, d	2
b, e	3
c, e	2
d, e	3
b, d, e	2

FI^4	Support count
a	3
b	3
c	4
e	3
a, b	3
a, c	3
a, e	2
b, c	3
b, e	2
c, e	3
a, b, c	3
a, b, e	2
a, c, e	2
b, c, e	2
a, b, c, e	2

Fig. 2 The frequent itemsets of each partition $\mathcal{FI}^j$; $j = 1, 2, 3, 4$

2. For each partition P_j, mine its frequent itemsets $\mathcal{FI}^j$ (Fig. 2).
3. Generate the candidates $\mathcal{C}_D = \bigcup_{j=1}^{4} \mathcal{FI}^j$ and its corresponding matrix S (Fig. 3).
4. Build a matrix S (Fig. 3) according to Eq. 3. For instance, the candidate itemset $c_i = \{bc\}$ is frequent at partitions P_1, P_2, and P_4 with $S(i, 1) = 2$, $S(i, 2) = 2$, and $S(i, 4) = 3$. While this itemset $\{bc\}$ is infrequent at partitions P_3, therefore $S(i, 3) = 0$.
5. Adding the column m (Fig. 3) that contains the number of partitions in which the itemset is frequent. Where at candidate $c_i = \{a\}$, $m(i) = 3$ means $\{a\}$ is frequent at three partitions (as shown at partitions 1, 2, 4).
6. Generate the closed candidates $\mathcal{CC}_D$ (Definition 1). For instance, the candidates $c_i = \{ab\}$ and $c_{i'} = \{abc\}$ are frequent and have the same support count at partitions P_1, P_2, and P_4. where, $c_{i'} \supset c_i$, and $S(i,j) = S(i',j); j = 1, 2, 4$. According to Definition 1, $\{bc\}$ is non-closed candidate. Therefore $\{bc\}$ is not included in $\mathcal{CC}_D$ due to the two frequent itemsets $\{bc\}$ and $\{abc\}$ share the same information. So, the closed candidate abc is enough to calculate the similarity between partitions. While, the candidate $c_i = \{ac\}$ is a closed candidate because $c_i = \{ac\}$ and its the only super candidate $c_{i'} = \{abc\}$ not have the same support at corresponding partitions, as shown $S(i,j) \neq S(i',j); j = 1, 2$ which violates Definition 1. Therefore, $\mathcal{CC}_D = \{a, b, c, d, e, ab, ac, ce, abc\}$.



S

$\mathcal{C}_D$	P1	P2	P3	P4	m
a	3	4	0	3	3
b	2	3	3	3	4
c	4	3	2	4	4
d	3	2	3	0	3
e	3	0	4	3	3
a, b	2	3	0	3	3
a, c	3	3	0	3	3
a, d	2	2	0	0	2
a, e	2	0	0	2	2
b, c	2	2	0	3	3
b, d	0	0	2	0	1
b, e	0	0	3	2	2
c, d	3	0	0	0	1
c, e	3	0	2	3	3
d, e	2	0	3	0	2
a, b, c	2	2	0	3	3
a, b, e	0	0	0	2	1
a, c, d	2	0	0	0	1
a, c, e	2	0	0	2	2
b, c, e	0	0	0	2	1
b, d, e	0	0	2	0	1
c, d, e	2	0	0	0	1
a, b, c, e	0	0	0	2	1

🟥 Infrequent (sup<minsup)

▨ Pruned candidates

S (proposed Approach) Minsup=8

P1	P2	P3	P4	sup	$\mathcal{FI}$
3	4	1.4444	3	11.4444	a
2	3	3	3	11.0000	b
4	3	2	4	13.0000	c
3	2	3	1.77777	9.7778	d
3	1.88888	4	3	11.8889	e
2	3	1.1481	3	9.1481	a, b
3	3	1.3333	3	10.3333	a, c
2	2	1.037037	3	8.0370	b, c
3	1.66666	2	3	9.6667	c, e
2	2	1.0370	3	8.0370	a, b, c

S_{closed}

$\mathcal{CC}_D$	P1	P2	P3	P4
a	3	4	0	3
b	2	3	3	3
c	4	3	2	4
d	3	2	3	0
e	3	0	4	3
a, b	2	3	0	3
a, c	3	3	0	3
c, e	3	0	2	3
a, b, c	2	2	0	3

$M_{\mathcal{CC}}$ (proposed Approach)

1	0.77778	0.55556	0.88889
0.77778	1	0.33333	0.66667
0.55556	0.33333	1	0.44444
0.88889	0.66667	0.44444	1

The Proposed Approach

S (FPMSIM) Minsup=8

P1	P2	P3	P4	sup	$\mathcal{FI}$
3	4	0.9720	3	10.9720	a
2	3	3	3	11.0000	b
4	3	2	4	13.0000	c
3	2	3	1.25490	9.2549	d
3	1.38571	4	3	11.3857	e
2	3	0.7829	3	8.7829	a, b
3	3	0.9006	3	9.9006	a, c
2	2	0.5672	1.08823	5.6555	
2	1.1	0.6863	2	5.7863	
2	2	0.7115	3	7.7115	
0.70588	0.42857	2	0.66666	3.8011	
1.11764	0.82142	3		6.9391	
3	1.8	1.0588	1.76470	7.6235	
3	1.24285	2	3	9.2429	c, e
2	0.92142	3	1.08823	7.0097	
2	2	0.7115	3	7.7115	
1.17647	1	0.6667	2	4.8431	
2	1.2	0.7059	1.17647	5.0824	
2	1.1	0.6863	2	5.7863	
1.17647	1	0.6667	2	4.8431	
0.70588	0.42857	2	0.66666	3.8011	
2	1.2	0.7059	1.17647	5.0824	
1.17647	1	0.6667	2	4.8431	

$M_{\mathcal{C}}$ (FPMSIM)

1	0.52941	0.44444	0.61111
0.52941	1	0.20000	0.46667
0.44444	0.20000	1	0.27778
0.61111	0.46667	0.27778	1

FPMSIM Approach

Fig. 3 CC-IFIM versus FPMSIM: the candidates $\mathcal{C}_D$ and its matrix S, the closed candidates $\mathcal{CC}_D$ and its corresponding matrix S_{closed}

7. From a matrix S, extract only a matrix S_{closed} that corresponds to $\mathcal{CC}_D$ (Fig. 3).
8. Using the $\mathcal{CC}_D$ and its matrix S_{closed} (Fig. 3) and based on Jaccard similarity method, calculate the similarity coefficients between each pair of partitions. According to Eq. (5), the matrix $M_{\mathcal{C}}\mathcal{C}$ (Fig. 3) has been created. $M_{\mathcal{C}}\mathcal{C}(1,2) = M_{\mathcal{C}}\mathcal{C}(2,1) = \frac{7}{9} = 0.7778$, means 7 and 9 are the numbers of the intersection and union between two partitions P_1 and P_2, respectively.
9. Apply pruning step (Fig. 3), for the itemset $c_i = \{ad\}$, $S(i,1) = S(i,2) = 2$, $S(i,3) = S(i,4) = 0$, this mean the itemset $\{ad\}$ is infrequent at P_3 and P_4. Then using our hypothesis $S'(i,j') = \text{minsup} - 1 = 2 - 1 = 1; j' = 3,4$ then $\text{Critical\_supp}(ad) = 2 + 2 + 1 + 1 = 6 < \text{minsup}_D = 8$. This means the candidate $\{ad\}$ is a global infrequent itemset. Therefore, in CC-IFIM, the itemset $\{ad\}$ must be ignored from $\mathcal{C}_D$. While the candidate $\{a\}$, its $\text{Critical\_supp}(a) = 3 + 4 + 1 + 3 = 11 > \text{minsup}_D = 8$. Therefore, the candidate a may be valid as a frequent global itemset. Finally, $\mathcal{C}_D = \{a, b, c, d, e, ab, ac, bc, ce, abc\}$. The shaded cells in $\mathcal{CC}_D$ represent the ignored candidates (Fig. 3). As well as, the values that correspond to the pruned candidates are ignored.
10. Using $M_C\mathcal{C}$ and Eq. (5), estimate the support of all candidates $c_i \in \mathcal{C}_D; S(i,j) = 0; j = 1,2,3,4$ (Fig. 3). As shown, for itemset $c_i = \{a\}$, $S(i,3) = \frac{1}{m(i)} \sum_{j=1, j\neq 3}^{4} M_{\mathcal{CC}}(j,3) \times S(i,j) = \frac{1}{3}(M_{\mathcal{CC}}(1,3) \times S(i,1) + M_{\mathcal{CC}}(2,3)$ $\times S(i,2) + M_{\mathcal{CC}}(4,3) \times S(i,4)) =$

    $\frac{1}{3}(0.55556 \times 3 + 0.3333 \times 4 + 0.44444 \times 3) = 1.4444 \cdot$ The resulted S is shown in CC-IFIM partition at the middle of Fig. 3.

11. For each candidate $c \in \mathcal{C}_D$, estimate its global support (Fig. 3). As shown, $sup(a) = 3 + 4 + 1.4444.3 = 11.444 > \text{minsup} = 8$, then the itemset $\{a\}$ is global frequent and has been added to $\mathcal{FI}$.
12. Finally $\mathcal{FI} = \{a, b, c, d, e, ab, ac, bc, ce, abc\}$;$sup(c) \geq \text{minsup}$;$c \in \mathcal{FI}$.

- *Incremental dataset* Finally, this approach will also deal with any incremental datasets as a new partition P_2, where P_1 is the original dataset. Similarly as shown in Algorithm 2, $\mathcal{FI}^1$ and $\mathcal{FI}^2$ are extracted for the partitions P_1 and P_2, respectively. All the steps as shown in Algorithm 2 have been applied to P_1 and P_2. Where, form the candidates $\mathcal{C} = \mathcal{FI}^1 \bigcup \mathcal{FI}^2$. Then form a matrix S for the two partitions P_1 and P_2 as mentioned earlier. Find the closed candidates $\mathcal{CC}$, then the similarity matrix $M_{\mathcal{CC}}$ that is corresponding to the closed candidates from $\mathcal{CC}$ (Definition 1). Apply the pruning step, then Using $M_{\mathcal{CC}}$ estimate the support of zero entries in matrix $S_{\mathcal{C}}$ to find the global support of all candidates. From $\mathcal{FI}$ that satisfies miusup threshold.
- *The Differences between traditional*, FPMSIM,*and* CC-IFIM *approaches* The traditional approach is applied to the transaction dataset $\mathcal{D}$ (Fig. 1), where the extracted $\mathcal{FI}_{\text{Original}} = \{a, b, c, d, e, ab, ac, bc, ce, abc\}$. Using FPMSIM, the similarity matrix $M_{\mathcal{C}}$ (shown at FPMSIM partition at the right of Fig. 3) is measured based on the original matrix S that corresponds $\mathcal{C}_D$ instead of $M_{\mathcal{CC}}$ as in CC-IFIM. In FPMSIM that is based on Jaccard similarity, the matrix $M_{\mathcal{C}}$ is calculated according to the following equation:

$$M_{\mathcal{C}}(j, j') = \frac{|c_i \in \mathcal{C}_D;\ S(i, j) \neq 0 \text{ and } S(i, j') \neq 0|}{|c_i \in \mathcal{C}_D;\ S(i, j) \neq 0 \text{ or } S(i, j') \neq 0|} \tag{13}$$

  The reason of calculating similarity matrix based on $\mathcal{CC}_D$ instead of $\mathcal{C}_D$ returns to the following property: if there are two frequent itemsets X and Y have the same support at all the corresponding partitions, and $X \subset Y$, There are the same and share the same information. Therefore, using X and Y for computing the similarity between two partitions j and j' decrease the coefficient $M(j, j')$ because the denominator is increased twice, one for X and one for Y, consequently, the calculated similarity coefficient is decreased. Instead, we have to ignore X and keep only Y, consequently, the denominator is increased once which leads to increase the similarity coefficient compared to the previous. This means that the computed similarity based only on local closed candidates is more realistic and better. For example, Using $\mathcal{CC}_D$, $M_{\mathcal{CC}}(1, 2) = 7/9 = 0.7778$. Where, the number of itemsets which are frequent in P_1 and P_2 (intersection) is 7. Where, the intersected itemsets in $\mathcal{CC}_D$ are $\{a, b, c, d, ab, ac, abc\}$. while the union is 9 which is $\{a, b, c, d, e, ab, ac, ce, abc\}$. Similarly, but using $\mathcal{C}_D$, $M_{\mathcal{C}}(1, 2) = 9/17 = 0.52941$. As shown in (Fig. 3) the coefficients of matrix $M_{\mathcal{C}}\mathcal{C}$ are greater than the coefficients of matrix $M_{\mathcal{C}}$. Therefore, the estimated support of CC-IFIM is greater than FPMSIM.

Back to the example, it is worth noting that, for calculating the similarity matrix, we consider the whole candidates in $\mathcal{C}_D$ before applying the pruning step as in our

Table 1 Characteristics of datasets

Dataset	Num. of transactions	Num. of items	Avg. length
T10I4D100K	10000	870	10
T40I10D100K	10000	1000	40
Mushroom	8124	120	22
Accidents	340183	468	33.8
Retail	88162	16470	10.3
Kosarak	990002	41270	8.1

proposed approach. Based on M_C, the corresponding S is calculated (shown at the FPMSIM partition at the right of Fig. 3), and the global support is estimated. Finally, the extracted $\mathcal{FI}_{\text{FPMSIM}} = \{a, b, c, d, e, ab, ac, ce\}$. As shown, $\mathcal{FI}_{\text{FPMSIM}} \subset \mathcal{FI}_{\text{Original}}$, where there are two frequent itemsets missing in FPMSIM. While in CC-IFIM, $\mathcal{FI}_{\text{proposed}} = \mathcal{FI}_{\text{Original}}$. Finally, as shown in Fig. 3, it is clear that there is waste of time and memory used for processing useless thirteen candidates as in FPMSIM compared to our proposed approach CC-IFIM.

4 Experimental results

In this section, we compare CC-IFIM with FPMSIM [18]. We have implemented both two approaches using Python. The implemented approaches have been applied to various real datasets and synthetic datasets (as shown in Table 1) (URL:http://fimi.ua.ac.be.) which are widely used for performance evaluation in the pattern mining area.

Accidents and Mushroom are dense datasets, while Retail is a sparse dataset. Generally, a dense dataset is composed of relatively long transactions and a small number of items, and a sparse dataset is characterized by relatively short transactions and a large number of items.

we have used a synthetic sparse dataset T10I4D100K and a real dataset, online news portal click-stream data (Kosarak). The efficiency (running time) of the CC-IFIM compared to FPMSIM is shown in Sect. 4.1. Section 4.2 introduces the rich comparative study that shows the accuracy of CC-IFIM compared to FPMSIM using three measurements; coverage, precision, and average support error.

4.1 Efficiency of CC-IFIM

Using several minsup thresholds on the mentioned datasets:

- *Running time* Since the generation of $\mathcal{FI}$ of each part is similar in both FPMSIM and CC-IFIM, the running time of $\mathcal{FI}$ generation is not included in the comparisons. Figure 4 shows the running time in seconds of CC-IFIM compared to FPMSIM. In CC-IFIM, the running time is calculated starting from pruning $\mathcal{C}_D$,

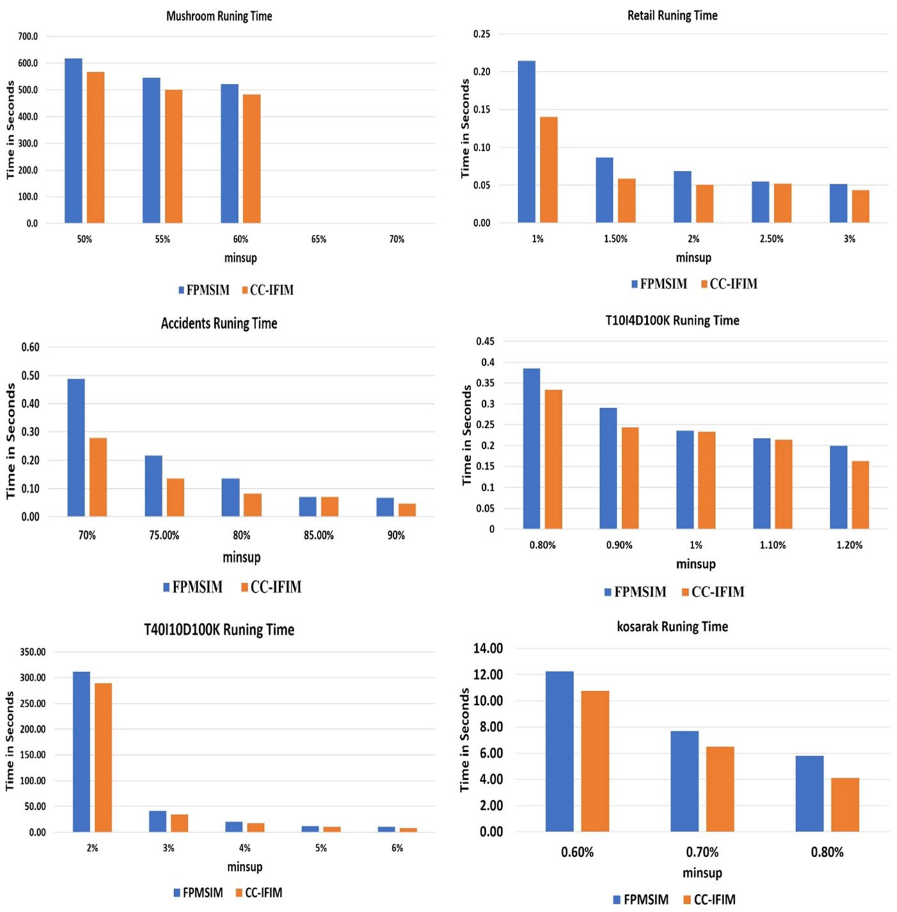


Fig. 4 Execution time between FPMSIM and CC-IFIM for different datasets with varying minsup

getting $\mathcal{CC}_D$, calculating M_{CC}, then estimating the zeros entries in the matrix S. While in FPMSIM, the running time is calculated starting from calculating M_C, then estimating the zeros entries in the matrix S. All results show CC-IFIM is better than FPMSIM. Where in Mushroom, Accidents, Retail, T10I4D100K, and T10I4D100K datasets, CC-IFIM reduced the execution time of FPMSIM around 10–12%, 10–42%, 16–34%, 10–18%, and 10–18%, respectively. In Kosarak dataset, CC-IFIM reduced the execution time of FPMSIM around 10–13%. Although our approach consumes time for extracting the closed candidates, our approach with the pruning step is more efficient than FPMSIM.

- *Memory consumption* Fig. 5 shows the memory consumption of CC-IFIM compared to FPMSIM. All results show CC-IFIM uses memory more efficiently than FPMSIM. As a result, our approach consumed less memory than FPMSIM.

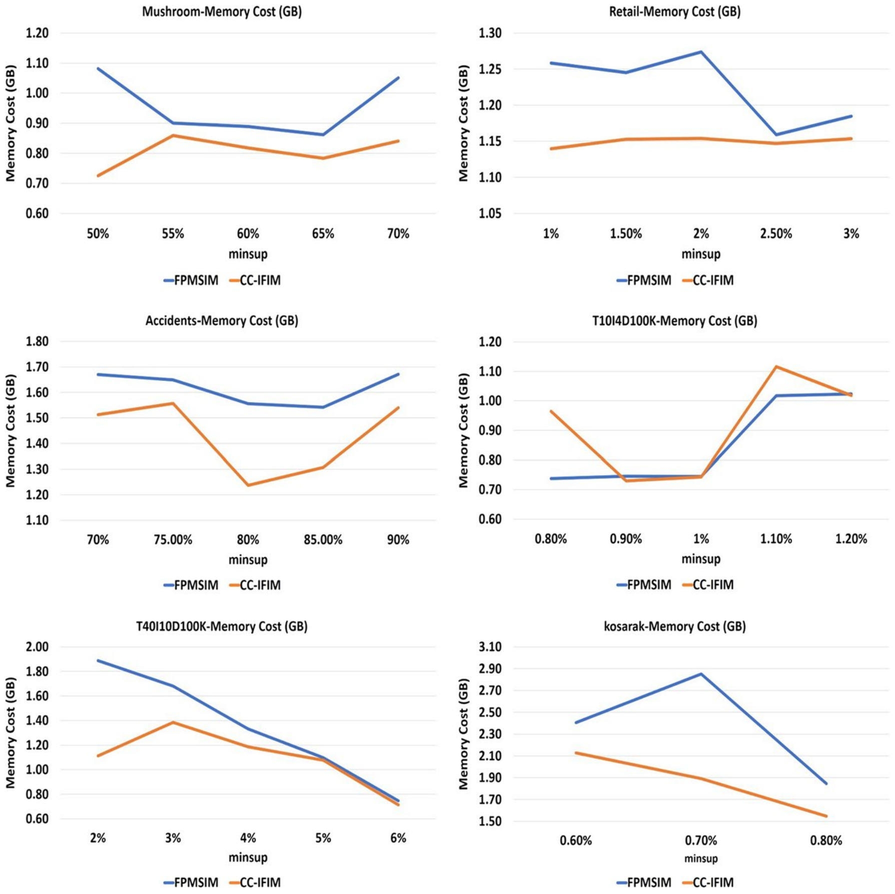


Fig. 5 The memory consumption of FPMSIM and CC-IFIM for different datasets with varying minsup

4.2 Accuracy of CC-IFIM Using M_{CC}

The coverage, precision, and average support error of CC-IFIM are evaluated against FPMSIM [18].

The coverage is calculated using the following equation:

$$\text{coverage} = \frac{|\mathcal{FI}_{\text{app}}| - |f\mathcal{PFI}|}{|\mathcal{FI}|} \tag{14}$$

where $\mathcal{FI}_{app}$ is the approximated $\mathcal{FI}$ using any approach. $f\mathcal{PFI}$ is the false positive $\mathcal{FI}$. The precision reflects the effect of false positive and false negative itemsets on the accuracy of experimental [18], which is defined as:

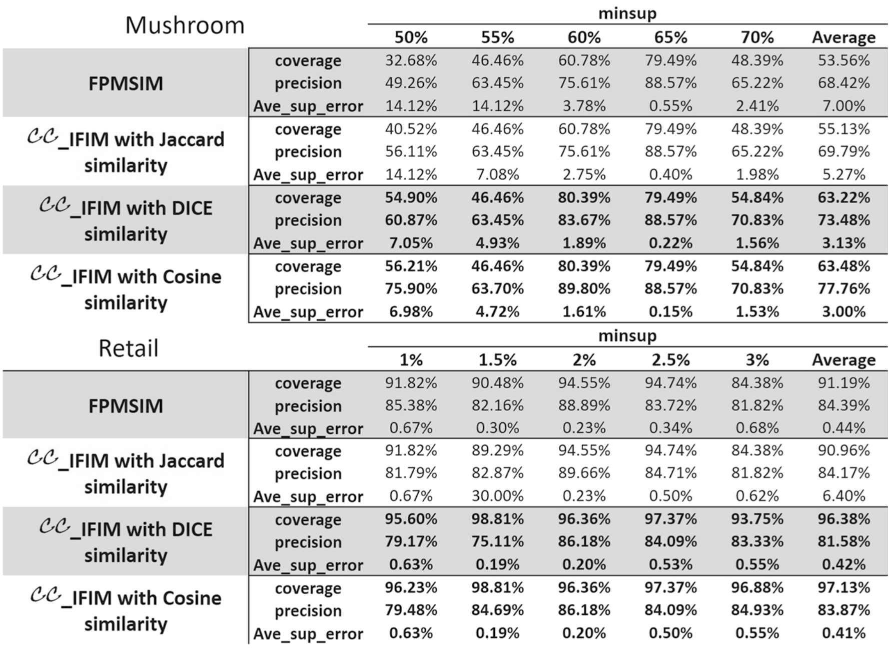
$$\text{precision} = \left(1 - \frac{|fPFI| + |fNFI|}{|\mathcal{FI}_{\text{app}}| + |\mathcal{FI}|}\right) * 100\% \tag{15}$$

$fN\mathcal{FI}$ is the false negative $\mathcal{FI}$.

The support estimation error is the average of the difference between the estimated support $\sup^*(f)$ of the frequent itemsets f in CC-IFIM and the real support $\sup(f)$ of these itemsets as follows:

$$\text{ave\_sup\_error} = \frac{1}{|\mathcal{FI}|} \sum_{f \in \mathcal{FI}} \frac{|\sup^*(f) - \sup(f)|}{|\sup(f)|} * 100\% \tag{16}$$

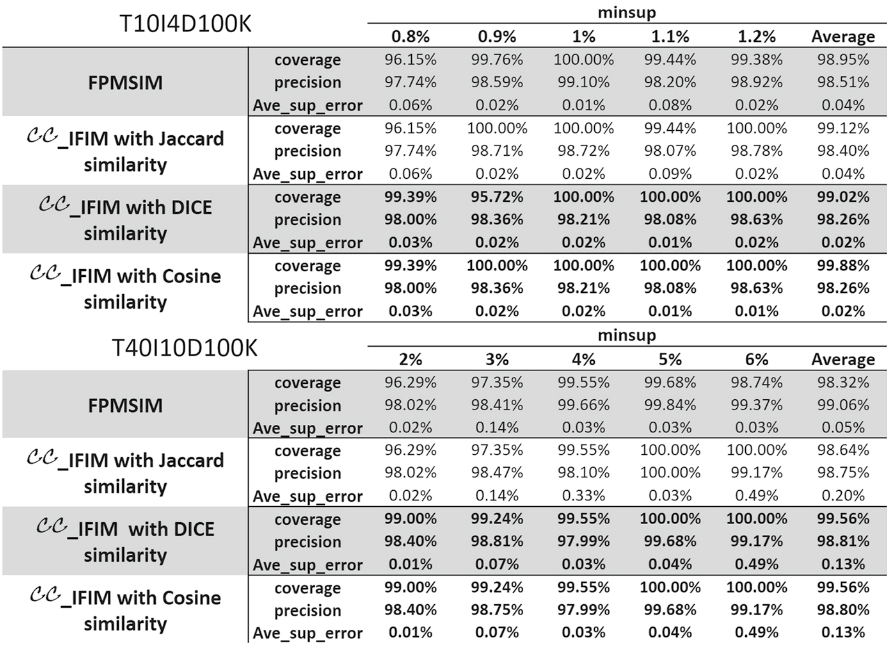
Figures 6 and 7 shows coverage, precision, and avg_sup_error among FPMSIM and three versions of CC-IFIM based on Jaccard, DICE and Cosine, respectively. These results have been tested using several minsup$p$. On Mushroom, Retail, T10I4D100K and T40I10D100K datasets, the overall coverage and precision of CC-IFIM are better than FPMSIM. This means the similarity based on $\mathcal{CC}_D$ is more accurate of estimating the global itemsets support than based on $\mathcal{C}_D$ as in FPMSIM. As shown in the last column, that shows the average calculations, CC-IFIM based on Dice or Cosine similarity is better than using Jaccard similarity. It is worth noting that, although the avg_supp_error is increasing on CC-IFIM, it is very normal due to the increasing of the number of $\mathcal{FI}$. In Mushroom dataset, avg_supp_error recorded 0% at some minsup thresholds, this does not indicate that the estimation support values are



Mushroom		minsup					
		50%	55%	60%	65%	70%	Average
FPMSIM	coverage	32.68%	46.46%	60.78%	79.49%	48.39%	53.56%
	precision	49.26%	63.45%	75.61%	88.57%	65.22%	68.42%
	Ave_sup_error	14.12%	14.12%	3.78%	0.55%	2.41%	7.00%
$\mathcal{CC}$_IFIM with Jaccard similarity	coverage	40.52%	46.46%	60.78%	79.49%	48.39%	55.13%
	precision	56.11%	63.45%	75.61%	88.57%	65.22%	69.79%
	Ave_sup_error	14.12%	7.08%	2.75%	0.40%	1.98%	5.27%
$\mathcal{CC}$_IFIM with DICE similarity	coverage	**54.90%**	**46.46%**	**80.39%**	**79.49%**	**54.84%**	**63.22%**
	precision	**60.87%**	**63.45%**	**83.67%**	**88.57%**	**70.83%**	**73.48%**
	Ave_sup_error	**7.05%**	**4.93%**	**1.89%**	**0.22%**	**1.56%**	**3.13%**
$\mathcal{CC}$_IFIM with Cosine similarity	coverage	**56.21%**	**46.46%**	**80.39%**	**79.49%**	**54.84%**	**63.48%**
	precision	**75.90%**	**63.70%**	**89.80%**	**88.57%**	**70.83%**	**77.76%**
	Ave_sup_error	**6.98%**	**4.72%**	**1.61%**	**0.15%**	**1.53%**	**3.00%**

Retail		minsup					
		1%	1.5%	2%	2.5%	3%	Average
FPMSIM	coverage	91.82%	90.48%	94.55%	94.74%	84.38%	91.19%
	precision	85.38%	82.16%	88.89%	83.72%	81.82%	84.39%
	Ave_sup_error	0.67%	0.30%	0.23%	0.34%	0.68%	0.44%
$\mathcal{CC}$_IFIM with Jaccard similarity	coverage	91.82%	89.29%	94.55%	94.74%	84.38%	90.96%
	precision	81.79%	82.87%	89.66%	84.71%	81.82%	84.17%
	Ave_sup_error	0.67%	30.00%	0.23%	0.50%	0.62%	6.40%
$\mathcal{CC}$_IFIM with DICE similarity	coverage	**95.60%**	**98.81%**	**96.36%**	**97.37%**	**93.75%**	**96.38%**
	precision	**79.17%**	**75.11%**	**86.18%**	**84.09%**	**83.33%**	**81.58%**
	Ave_sup_error	**0.63%**	**0.19%**	**0.20%**	**0.53%**	**0.55%**	**0.42%**
$\mathcal{CC}$_IFIM with Cosine similarity	coverage	**96.23%**	**98.81%**	**96.36%**	**97.37%**	**96.88%**	**97.13%**
	precision	**79.48%**	**84.69%**	**86.18%**	**84.09%**	**84.93%**	**83.87%**
	Ave_sup_error	**0.63%**	**0.19%**	**0.20%**	**0.50%**	**0.55%**	**0.41%**

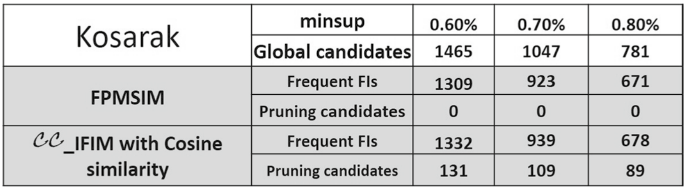
Fig. 6 Coverage, precision and avg_sup_error among FPMSIM and three versions of CC-IFIM based on Jaccard, DICE and Cosine for Mushroom and retail datasets



T10I4D100K		minsup					
		0.8%	0.9%	1%	1.1%	1.2%	Average
FPMSIM	coverage	96.15%	99.76%	100.00%	99.44%	99.38%	98.95%
	precision	97.74%	98.59%	99.10%	98.20%	98.92%	98.51%
	Ave_sup_error	0.06%	0.02%	0.01%	0.08%	0.02%	0.04%
$\mathcal{CC}$_IFIM with Jaccard similarity	coverage	96.15%	100.00%	100.00%	99.44%	100.00%	99.12%
	precision	97.74%	98.71%	98.72%	98.07%	98.78%	98.40%
	Ave_sup_error	0.06%	0.02%	0.02%	0.09%	0.02%	0.04%
$\mathcal{CC}$_IFIM with DICE similarity	coverage	**99.39%**	**95.72%**	**100.00%**	**100.00%**	**100.00%**	**99.02%**
	precision	**98.00%**	**98.36%**	**98.21%**	**98.08%**	**98.63%**	**98.26%**
	Ave_sup_error	**0.03%**	**0.02%**	**0.02%**	**0.01%**	**0.02%**	**0.02%**
$\mathcal{CC}$_IFIM with Cosine similarity	coverage	**99.39%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**99.88%**
	precision	**98.00%**	**98.36%**	**98.21%**	**98.08%**	**98.63%**	**98.26%**
	Ave_sup_error	**0.03%**	**0.02%**	**0.02%**	**0.01%**	**0.01%**	**0.02%**

T40I10D100K		minsup					
		2%	3%	4%	5%	6%	Average
FPMSIM	coverage	96.29%	97.35%	99.55%	99.68%	98.74%	98.32%
	precision	98.02%	98.41%	99.66%	99.84%	99.37%	99.06%
	Ave_sup_error	0.02%	0.14%	0.03%	0.03%	0.03%	0.05%
$\mathcal{CC}$_IFIM with Jaccard similarity	coverage	96.29%	97.35%	99.55%	100.00%	100.00%	98.64%
	precision	98.02%	98.47%	98.10%	100.00%	99.17%	98.75%
	Ave_sup_error	0.02%	0.14%	0.33%	0.03%	0.49%	0.20%
$\mathcal{CC}$_IFIM with DICE similarity	coverage	**99.00%**	**99.24%**	**99.55%**	**100.00%**	**100.00%**	**99.56%**
	precision	**98.40%**	**98.81%**	**97.99%**	**99.68%**	**99.17%**	**98.81%**
	Ave_sup_error	**0.01%**	**0.07%**	**0.03%**	**0.04%**	**0.49%**	**0.13%**
$\mathcal{CC}$_IFIM with Cosine similarity	coverage	**99.00%**	**99.24%**	**99.55%**	**100.00%**	**100.00%**	**99.56%**
	precision	**98.40%**	**98.75%**	**97.99%**	**99.68%**	**99.17%**	**98.80%**
	Ave_sup_error	**0.01%**	**0.07%**	**0.03%**	**0.04%**	**0.49%**	**0.13%**

Fig. 7 Coverage, precision and avg_sup_error among FPMSIM and three versions of CC-IFIM based on Jaccard, DICE, and Cosine for T10I4D100K and T40I10D100K datasets



Kosarak	minsup	0.60%	0.70%	0.80%
	Global candidates	1465	1047	781
FPMSIM	Frequent FIs	1309	923	671
	Pruning candidates	0	0	0
$\mathcal{CC}$_IFIM with Cosine similarity	Frequent FIs	1332	939	678
	Pruning candidates	131	109	89

Fig. 8 Extracted $\mathcal{FI}$s of both FPMISM and CC-IFIM approaches for Kosarak dataset at various minsup

identical to an actual support values. It means CC-IFIM and FPMSIM didn't estimate any support values, where all values of a matrix S is non-zero.

For big data, Kosarak dataset, the traditional algorithm fails to mine all $\mathcal{FI}s$, according to heap memory exception. Therefore, we can not able to calculate coverage, precision and avg_supp_error, while the approximated approaches are the suitable solution to mine $\mathcal{FI}s$. The comparative study has been made between CC-IFIM and FPMISM only. Figure 8 shows that the number of extracted $\mathcal{FI}$s by CC-IFIM is more than $\mathcal{FI}$s by FPMSIM. As shown, at minsup = 0.6% CC-IFIM approach can mine 1332 frequent itemset from 1465 candidates, while FPMISM mines only 1309. Moreover, $CC - IFIM$ approach pruned 131 candidates.

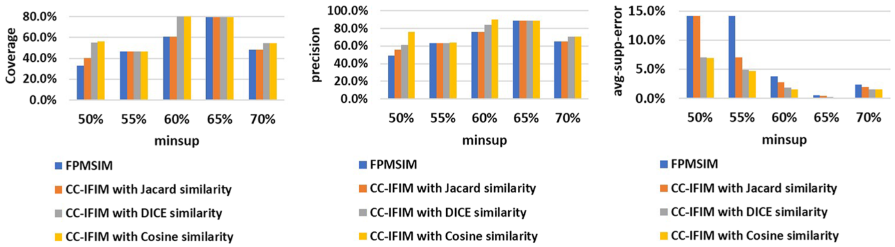


Fig. 9 Coverage, precision and avg_sup_error among FPMSIM and three versions of CC-IFIM based on Jaccard, DICE and Cosine for Mushroom dataset

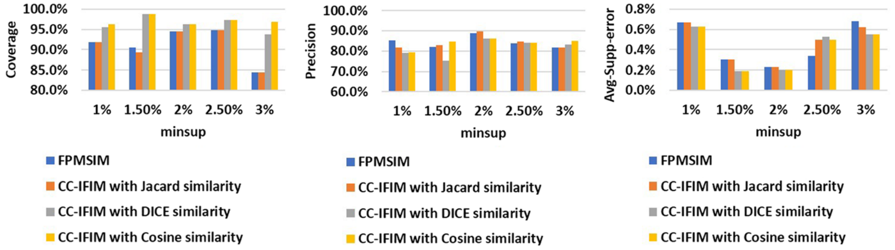


Fig. 10 Coverage, precision and avg_sup_error among FPMSIM and three versions of CC-IFIM based on Jaccard, DICE and Cosine for retail dataset

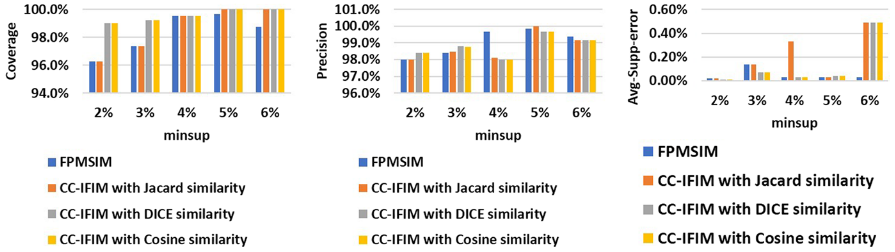


Fig. 11 Coverage, precision and avg_sup_error among FPMSIM and three versions of CC-IFIM based on Jaccard, DICE and Cosine for T40I10D100K dataset

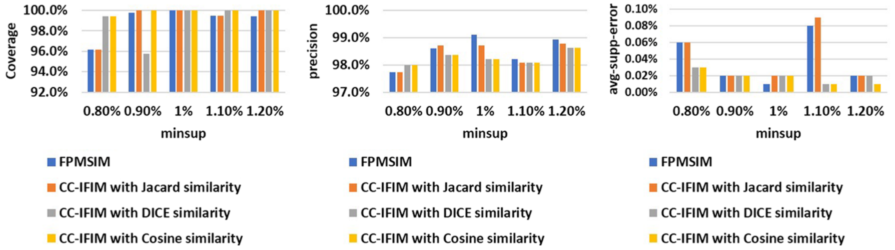
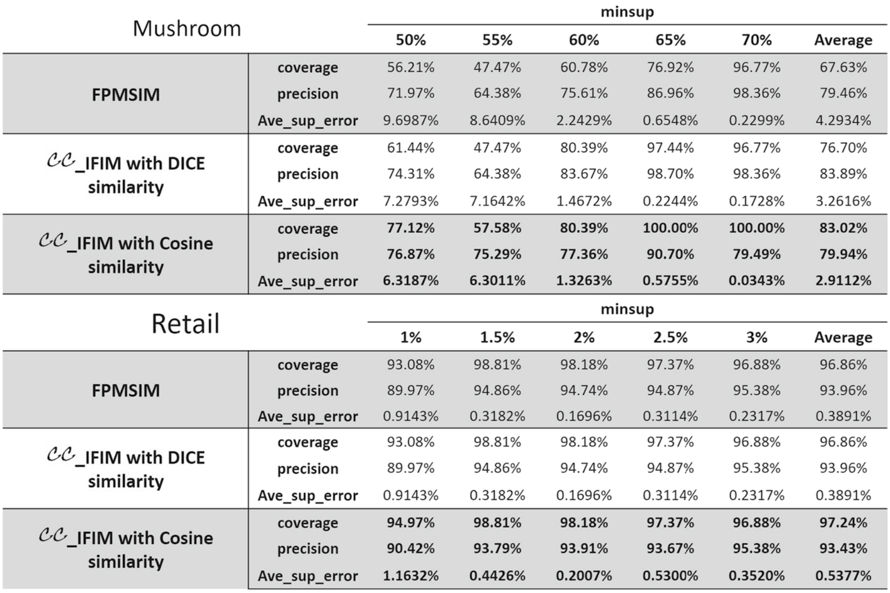


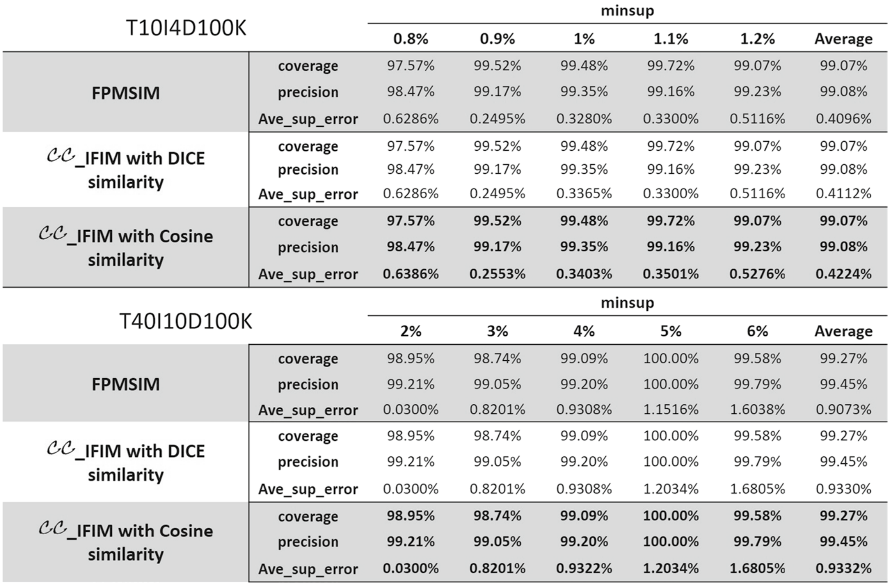
Fig. 12 Coverage, precision and avg_sup_error among FPMSIM and three versions of CC-IFIM based on Jaccard, DICE and Cosine for T10D100K dataset



Mushroom		minsup					
		50%	55%	60%	65%	70%	Average
FPMSIM	coverage	56.21%	47.47%	60.78%	76.92%	96.77%	67.63%
	precision	71.97%	64.38%	75.61%	86.96%	98.36%	79.46%
	Ave_sup_error	9.6987%	8.6409%	2.2429%	0.6548%	0.2299%	4.2934%
CC_IFIM with DICE similarity	coverage	61.44%	47.47%	80.39%	97.44%	96.77%	76.70%
	precision	74.31%	64.38%	83.67%	98.70%	98.36%	83.89%
	Ave_sup_error	7.2793%	7.1642%	1.4672%	0.2244%	0.1728%	3.2616%
CC_IFIM with Cosine similarity	coverage	**77.12%**	**57.58%**	**80.39%**	**100.00%**	**100.00%**	**83.02%**
	precision	**76.87%**	**75.29%**	**77.36%**	**90.70%**	**79.49%**	**79.94%**
	Ave_sup_error	**6.3187%**	**6.3011%**	**1.3263%**	**0.5755%**	**0.0343%**	**2.9112%**

Retail		minsup					
		1%	1.5%	2%	2.5%	3%	Average
FPMSIM	coverage	93.08%	98.81%	98.18%	97.37%	96.88%	96.86%
	precision	89.97%	94.86%	94.74%	94.87%	95.38%	93.96%
	Ave_sup_error	0.9143%	0.3182%	0.1696%	0.3114%	0.2317%	0.3891%
CC_IFIM with DICE similarity	coverage	93.08%	98.81%	98.18%	97.37%	96.88%	96.86%
	precision	89.97%	94.86%	94.74%	94.87%	95.38%	93.96%
	Ave_sup_error	0.9143%	0.3182%	0.1696%	0.3114%	0.2317%	0.3891%
CC_IFIM with Cosine similarity	coverage	**94.97%**	**98.81%**	**98.18%**	**97.37%**	**96.88%**	**97.24%**
	precision	**90.42%**	**93.79%**	**93.91%**	**93.67%**	**95.38%**	**93.43%**
	Ave_sup_error	**1.1632%**	**0.4426%**	**0.2007%**	**0.5300%**	**0.3520%**	**0.5377%**

Fig. 13 Coverage, precision and avg_sup_error among FPMSIM and three versions of CC-IFIM based on Jaccard, DICE and Cosine for Mushroom and Retail Datasets after dividing them into 70% Original dataset and 30% incremental dataset



T10I4D100K		minsup					
		0.8%	0.9%	1%	1.1%	1.2%	Average
FPMSIM	coverage	97.57%	99.52%	99.48%	99.72%	99.07%	99.07%
	precision	98.47%	99.17%	99.35%	99.16%	99.23%	99.08%
	Ave_sup_error	0.6286%	0.2495%	0.3280%	0.3300%	0.5116%	0.4096%
CC_IFIM with DICE similarity	coverage	97.57%	99.52%	99.48%	99.72%	99.07%	99.07%
	precision	98.47%	99.17%	99.35%	99.16%	99.23%	99.08%
	Ave_sup_error	0.6286%	0.2495%	0.3365%	0.3300%	0.5116%	0.4112%
CC_IFIM with Cosine similarity	coverage	**97.57%**	**99.52%**	**99.48%**	**99.72%**	**99.07%**	**99.07%**
	precision	**98.47%**	**99.17%**	**99.35%**	**99.16%**	**99.23%**	**99.08%**
	Ave_sup_error	**0.6386%**	**0.2553%**	**0.3403%**	**0.3501%**	**0.5276%**	**0.4224%**

T40I10D100K		minsup					
		2%	3%	4%	5%	6%	Average
FPMSIM	coverage	98.95%	98.74%	99.09%	100.00%	99.58%	99.27%
	precision	99.21%	99.05%	99.20%	100.00%	99.79%	99.45%
	Ave_sup_error	0.0300%	0.8201%	0.9308%	1.1516%	1.6038%	0.9073%
CC_IFIM with DICE similarity	coverage	98.95%	98.74%	99.09%	100.00%	99.58%	99.27%
	precision	99.21%	99.05%	99.20%	100.00%	99.79%	99.45%
	Ave_sup_error	0.0300%	0.8201%	0.9308%	1.2034%	1.6805%	0.9330%
CC_IFIM with Cosine similarity	coverage	**98.95%**	**98.74%**	**99.09%**	**100.00%**	**99.58%**	**99.27%**
	precision	**99.21%**	**99.05%**	**99.20%**	**100.00%**	**99.79%**	**99.45%**
	Ave_sup_error	**0.0300%**	**0.8201%**	**0.9322%**	**1.2034%**	**1.6805%**	**0.9332%**

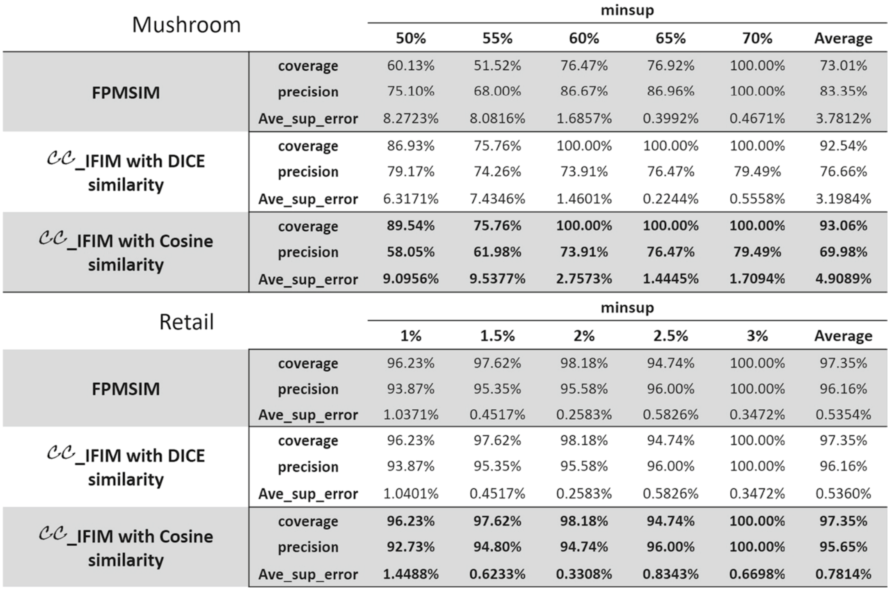
Fig. 14 Coverage, precision and avg_sup_error among FPMSIM and three versions of CC-IFIM based on Jaccard, DICE and Cosine for T10I4D100K and Fig. 11 datasets after dividing them into 70% original dataset and 30% incremental dataset

- *Incremental dataset* Figs. 9, 10, 11 and 12 are another way of visualization that show the comparison of FPMSIM and three versions of CC-IFIM on Mushroom, Retail, T40I4D100K and T10I4D100K datasets, respectively.

Figures13 and 14 shows the accuracy CC-IFIM compared to FPMSIM for the incremental datasets with different partition size to the original. Where the size of incremental dataset is 30% of the dataset. Similarly Figs. 15 and 16 where the size of incremental dataset is 20% of the dataset. These figure shows that CC-IFIM outperforms FPMSIM.

5 Conclusion

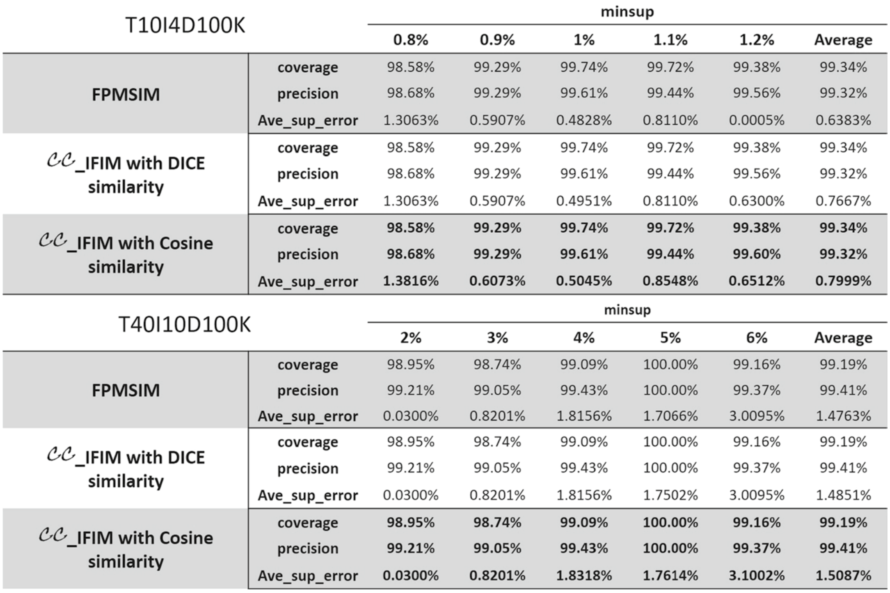
The incremental frequent itemset mining approach is an estimated good choice for the huge datasets that are rapidly expanding by periodically adding new transactions. In which, the approach frequent pattern tree and multi-scale incremental frequent itemset, FPMSIM, is used. It splits the dataset into several components, applies frequent itemset mining separately, then uses Jaccard similarity based on all local frequent itemsets the global frequent itemsets have been mined. However, it faces two problems: performance and efficiency. In this paper, we developed



Mushroom		minsup					
		50%	55%	60%	65%	70%	Average
FPMSIM	coverage	60.13%	51.52%	76.47%	76.92%	100.00%	73.01%
	precision	75.10%	68.00%	86.67%	86.96%	100.00%	83.35%
	Ave_sup_error	8.2723%	8.0816%	1.6857%	0.3992%	0.4671%	3.7812%
CC_IFIM with DICE similarity	coverage	86.93%	75.76%	100.00%	100.00%	100.00%	92.54%
	precision	79.17%	74.26%	73.91%	76.47%	79.49%	76.66%
	Ave_sup_error	6.3171%	7.4346%	1.4601%	0.2244%	0.5558%	3.1984%
CC_IFIM with Cosine similarity	coverage	**89.54%**	**75.76%**	**100.00%**	**100.00%**	**100.00%**	**93.06%**
	precision	**58.05%**	**61.98%**	**73.91%**	**76.47%**	**79.49%**	**69.98%**
	Ave_sup_error	**9.0956%**	**9.5377%**	**2.7573%**	**1.4445%**	**1.7094%**	**4.9089%**

Retail		minsup					
		1%	1.5%	2%	2.5%	3%	Average
FPMSIM	coverage	96.23%	97.62%	98.18%	94.74%	100.00%	97.35%
	precision	93.87%	95.35%	95.58%	96.00%	100.00%	96.16%
	Ave_sup_error	1.0371%	0.4517%	0.2583%	0.5826%	0.3472%	0.5354%
CC_IFIM with DICE similarity	coverage	96.23%	97.62%	98.18%	94.74%	100.00%	97.35%
	precision	93.87%	95.35%	95.58%	96.00%	100.00%	96.16%
	Ave_sup_error	1.0401%	0.4517%	0.2583%	0.5826%	0.3472%	0.5360%
CC_IFIM with Cosine similarity	coverage	**96.23%**	**97.62%**	**98.18%**	**94.74%**	**100.00%**	**97.35%**
	precision	**92.73%**	**94.80%**	**94.74%**	**96.00%**	**100.00%**	**95.65%**
	Ave_sup_error	**1.4488%**	**0.6233%**	**0.3308%**	**0.8343%**	**0.6698%**	**0.7814%**

Fig. 15 Coverage, precision and avg_sup_error among FPMSIM and three versions of CC-IFIM based on Jaccard, DICE and Cosine for Mushroom and retail datasets after dividing them into 80% original dataset and 20% incremental dataset



T10I4D100K		minsup					
		0.8%	0.9%	1%	1.1%	1.2%	Average
FPMSIM	coverage	98.58%	99.29%	99.74%	99.72%	99.38%	99.34%
	precision	98.68%	99.29%	99.61%	99.44%	99.56%	99.32%
	Ave_sup_error	1.3063%	0.5907%	0.4828%	0.8110%	0.0005%	0.6383%
𝒞𝒞_IFIM with DICE similarity	coverage	98.58%	99.29%	99.74%	99.72%	99.38%	99.34%
	precision	98.68%	99.29%	99.61%	99.44%	99.56%	99.32%
	Ave_sup_error	1.3063%	0.5907%	0.4951%	0.8110%	0.6300%	0.7667%
𝒞𝒞_IFIM with Cosine similarity	coverage	**98.58%**	**99.29%**	**99.74%**	**99.72%**	**99.38%**	**99.34%**
	precision	**98.68%**	**99.29%**	**99.61%**	**99.44%**	**99.60%**	**99.32%**
	Ave_sup_error	**1.3816%**	**0.6073%**	**0.5045%**	**0.8548%**	**0.6512%**	**0.7999%**

T40I10D100K		minsup					
		2%	3%	4%	5%	6%	Average
FPMSIM	coverage	98.95%	98.74%	99.09%	100.00%	99.16%	99.19%
	precision	99.21%	99.05%	99.43%	100.00%	99.37%	99.41%
	Ave_sup_error	0.0300%	0.8201%	1.8156%	1.7066%	3.0095%	1.4763%
𝒞𝒞_IFIM with DICE similarity	coverage	98.95%	98.74%	99.09%	100.00%	99.16%	99.19%
	precision	99.21%	99.05%	99.43%	100.00%	99.37%	99.41%
	Ave_sup_error	0.0300%	0.8201%	1.8156%	1.7502%	3.0095%	1.4851%
𝒞𝒞_IFIM with Cosine similarity	coverage	**98.95%**	**98.74%**	**99.09%**	**100.00%**	**99.16%**	**99.19%**
	precision	**99.21%**	**99.05%**	**99.43%**	**100.00%**	**99.37%**	**99.41%**
	Ave_sup_error	**0.0300%**	**0.8201%**	**1.8318%**	**1.7614%**	**3.1002%**	**1.5087%**

Fig. 16 Coverage, precision and avg_sup_error among FPMSIM and three versions of CC-IFIM based on Jaccard, DICE and Cosine for T10I4D100K and T40I10D100K Datasets after dividing them into 80% original dataset and 20% incremental dataset

this approach to be more efficient and accurate. In CC-IFIM, the performance has been increased by applying some prior candidate pruning. While the accuracy has been increased by applying similarity measurements on local closed frequent itemsets instead of local frequent itemsets. As well as using another similarity methods such as Dice or Cosine which is remarkably increasing the accuracy of CC-IFIM. The key contributions of this study are the introduction of a pruning approach for reducing the number of candidates, as well as a novel definition of closed itemsets termed local closed frequent itemsets, on which similarity methods rely. Future research will focus on establishing the scientific rationale for the ideal number of partitions in division processes. Also closed and maximal incremental frequent itemsets will be covered. Additionally, investigate several different similarity models in order to propose a new one.

Acknowledgments The authors acknowledge the role of the Artificial Intelligence and Cognitive Science Laboratory at the Department of Mathematics, Faculty of Science, Ain Shams University, Cairo, Egypt.

Author contributions All authors contributed equally to this work.

Funding Open access funding provided by The Science, Technology & Innovation Funding Authority (STDF) in cooperation with The Egyptian Knowledge Bank (EKB).

Data availability The datasets used during the current study are available in the public (the Frequent Itemset Mining Implementations) repository, [http://fimi.ua.ac.be]. The generated source codes of the proposed approach are available from the corresponding author on reasonable request.

Declarations

Conflict of interest All authors declare that there is no conflict of interest.

Ethical approval Not applicable.

Consent for publication Not applicable.

Consent to participate Not applicable.

References

1. Agrawal R, Imieliński T, Swami A (1993) Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pp. 207–216
2. Agrawal R, Srikant R et al (1994) Fast algorithms for mining association rules. In: Proceedings 20th International Conference Very Large Data Bases. VLDB 1215:487–499
3. Chen R, Zhao S, Liu M (2020) A fast approach for up-scaling frequent itemsets. IEEE Access 8:97141–97151
4. Cheung DW, Han J, Ng VT, Wong C (1996) Maintenance of discovered association rules in large databases: an incremental updating technique. In: Proceedings of the Twelfth International Conference on Data Engineering, pp. 106–114
5. Cheung DW, Lee SD, Kao B (1997) A general incremental technique for maintaining discovered association rules. Database Syst Adva Appl 97:185–194
6. Deng ZH (2016) Diffnodesets: an efficient structure for fast mining frequent itemsets. Appl Soft Comput 41:214–223
7. Deng ZH, Lv SL (2014) Fast mining frequent itemsets using nodesets. Expert Syst Appl 41(10):4505–4512
8. Deng ZH, Lv SL (2015) Prepost+: an efficient n-lists-based algorithm for mining frequent itemsets via children-parent equivalence pruning. Expert Syst Appl 42(13):5424–5432
9. Han J, Pei J, Yin Y, Mao R (2004) Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min Knowl Disc 8(1):53–87
10. Hong TP, Lin CW, Wu YL (2008) Incrementally fast updated frequent pattern trees. Expert Syst Appl 34(4):2424–2435
11. Huynh VQP, Küng J, Dang TK (2017) Incremental frequent itemsets mining with ippc tree. In: International Conference on Database and Expert Systems Applications, Springer, pp. 463–477
12. Li Y, Zhang ZH, Chen WB, Min F (2017) Tdup: an approach to incremental mining of frequent itemsets with three-way-decision pattern updating. Int J Mach Learn Cybern 8(2):441–453
13. Lv D, Fu B, Sun X, Qiu H, Liu X, Zhang Y (2017) Efficient fast updated frequent pattern tree algorithm and its parallel implementation. In: 2017 2nd International Conference on Image, Vision and Computing (ICIVC), IEEE, pp. 970–974

14. Satyavathi N, Rama B (2021) Dynamic and incremental update of mined association rules against changes in dataset. Innov Comput Sci Eng 25:115–121
15. Sun J, Xun Y, Zhang J, Li J (2019) Incremental frequent itemsets mining with FCFP tree. IEEE Access 7:136511–136524
16. Thada V, Jaglan V (2013) Comparison of Jaccard, dice, cosine similarity coefficient to find best fitness value for web retrieved documents using genetic algorithm. Int J Innov Eng Technol 2(4):202–205
17. Wael Z, Yasser K, Elham E, Fayed G (2009) Mining association rule with reducing candidates generation. In: Proceedings of Fourth International Conference on Intelligent Computing and Information Systems (ICICIS2009). ACM
18. Xun Y, Cui X, Zhang J, Yin Q (2021) Incremental frequent itemsets mining based on frequent pattern tree and multi-scale. Expert Syst Appl 163:113805
19. Zaki MJ (2000) Scalable algorithms for association mining. IEEE Trans Knowl Data Eng 12(3):372–390
20. Zhou Z, Ezeife C (2001) A low-scan incremental association rule maintenance method based on the apriori property. In: Conference of the Canadian Society for Computational Studies of Intelligence, Springer, pp. 26–35

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Maged Magdy[1] · **Fayed F. M. Ghaleb**[1] · **Dawlat A. El A. Mohamed**[1] · **Wael Zakaria**[1]

Maged Magdy
Maged.Magdy@sci.asu.edu.eg

Fayed F. M. Ghaleb
fmghaleb@yahoo.com

Dawlat A. El A. Mohamed
dr_dowlatkma@yahoo.com

[1] Department of Mathematics, Faculty of Science, Ain Shams University, Cairo, Egypt