IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 2, 2020, accepted May 14, 2020, date of publication May 19, 2020, date of current version June 4, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2995719

A Fast Approach for Up-Scaling Frequent Itemsets

RUNZI CHEN[1], SHULIANG ZHAO[2,3,4], AND MENGMENG LIU[5]

[1]College of Mathematics and Information Science, Hebei Normal University, Shijiazhuang 050024, China
[2]College of Computer and Cyber Security, Hebei Normal University, Shijiazhuang 050024, China
[3]Hebei Provincial Key Laboratory of Network and Information Security, Shijiazhuang 050024, China
[4]Hebei Provincial Engineering Research Center for Supply Chain Big Data Analytics and Data Security, Shijiazhuang 050024, China
[5]Library, Zhangjiakou University, Zhangjiakou 075000, China

Corresponding author: Shuliang Zhao (zhaoshuliang@hebtu.edu.cn)

ABSTRACT With the rapid growth of data scale and diversification of demand, people have an urgent desire to extract useful frequent itemset from datasets of different scales. It is no doubt that the traditional method can solve the problem. However, the relationships among datasets of different scales are not fully utilized. A fast approach proposed in this paper is as follows: the frequent itemsets on the large-scale data are directly inferred based on the frequent itemsets that are belonged small-scale datasets, instead of mined from the large-scale dataset again on condition that the frequent itemsets on the small-scale datasets have been mined. We conduct extensive experiments on one synthetic data and four UCI data sets. The experimental results show that our algorithm is significantly faster and consumes less memory than these leading algorithms.

INDEX TERMS Up-scaling, up-scaling frequent itemsets, frequent itemset mining, data mining.

I. INTRODUCTION

To analyze customer's buying behavior-based transactions database, Agrawal *et al.* first presented frequent itemset mining in 1993 [1], that is one of the critical data mining tasks and has widely used in many other significant data mining tasks including mining associations and correlations, classifying, clustering, etc. Since then, frequent itemset mining has been a hot field that has attracted a great deal of attention of researcher. After Apriori proposed, there are several improved algorithms because Apriori needs to scan the database repeatedly. These algorithms have a common feature: generating candidate itemsets. So, filtering candidate itemset is a challenging task. FP-growth algorithm is a classic representative that does not generate candidate itemsets and compresses the database representing frequent items into FP-tree, which retains the itemset association information [2]. In recent years, to enhance the efficiency of mining frequent itemset, three kinds of the data structure are presented by Deng *et al.*, named Node-list, N-list, and Nod-eset. FIN based Nodeset consumes less memory because the Nodeset structure requires only the pre-order(or post-order) [3]. Despite the above advantage of Nodeset, two data structures (DiffNodeset [3] and NegNodeset [4]) are proposed by Deng *et al.* and Aryabarzan *et al.*, and there are two algorithms named dFIN and negFIN based the former data structures respectively. Extensive experimental results show that dFIN and negFIN have the same speed but ming frequent itemset faster, compared with the state-of-the-art algorithms [4].

The same problem or system can be perceived at different levels of specificity (detail), depending on the complexity of the problem, available computing resources, and particular needs to be addressed [5]. Assuming you have nationwide patient information, you need to give both city and country managers some advice on which diseases that the same person suffers from, where the data onto each city is named small-scale data, and the data onto the whole country is named large-scale data. So, there is an urgent need to mine frequent itemsets on different scale datasets. It would be a pity that the traditional mining algorithm is first applied to discover frequent itemsets from the small-scale datasets, and then to from the large-scale dataset later. On the other hand, the scheme does not use the relationship between small-scale data and large-scale data. In this paper, a new framework(up-scaling) is proposed: the frequent itemsets on a small-scale datasets is used to directly infer the frequent itemsets on a large-scale dataset, instead of secondary mining on a large-scale dataset.

The contributions of this paper are listed as follows:

1) This paper presents a novel framework for addressing the issue that one mines frequent itemsets from different scale datasets.

The associate editor coordinating the review of this manuscript and approving it for publication was Xin Luo.

2) We propose the new algorithm mining frequent itemsets from the large-scale dataset, which depends on the frequent itemsets belonged to the small-scale datasets, not original data.

3) Experimental results show that the framework and the algorithm are efficient and violently reducing memory consumption with similar accuracy, especially in the case of short frequent itemsets, and the framework is better than the current optimal algorithm.

The rest of this paper is organized as follows: Section II discusses related work to mine frequent itemsets. Section III describes our problem and the framework is proposed in section IV. Section V shows the experimental results. The conclusions and some future research directions are given in section VI.

II. RELATED WORK

Frequent itemset mining is the first and foremost step of association rule mining [6]. In association rules mining and Frequent itemset mining literature, frequent Itemset mining methods are mainly divided into two main categories: 1) algorithms that mine frequent itemset that takes advantage of the horizontal data format. 2) algorithms that mine frequent itemset that takes advantage of the vertical data format.

In the first categorized algorithms, apriori, the basic algorithm for finding frequent itemsets, is firstly proposed by Agrawal and Srikant [7], which motivates many researchers to study this field. It adopts an iterative approach known as a level-wise search, where k-itemsets are used to explore (k + 1)-itemsets. Apriori algorithm often needs multiple passes over dataset and produces many candidate itemsets that are eventually pruned. For the problem that Apriori generates too many candidate itemsets, Park introduces the DHP algorithm that can reduce the number of candidate itemsets and improve efficiency by using a hash, function and bit vector [8]. To reduce the number of scanning the dataset, Savasere proposes a partitioning algorithm [9], which can obtain all frequent itemsets by scanning the dataset twice. First, The partitioning algorithm divides the dataset into several non-overlapping partitions and the frequent itemsets for each partition are computed. Then, another pass over the dataset is performed to acquire the support of the candidates and the frequent itemsets can be discovered. Y. Djenouri *et al.* propose SSFIM [10], which scans the transactional database when discovering frequent itemsets once. It has a unique feature to allow the generation of a fixed number of candidate itemsets, independently from the minimum support threshold, which intuitively allows to reduce the cost in terms of runtime for large databases. Toivonen presents the Sampling algorithm [11] based on the fact that trade off some degree of accuracy against efficiency. It uses the sampling method to extract an appropriate number of samples from the original dataset, and then mine frequent itemsets from the samples. The above all algorithms need to generate candidate itemsets. The classic and basic algorithm that doesn't need to generate candidate itemsets is the FP-growth algorithm [2]. It stores essential information about frequent itemset in a tree-based data structure, namely frequent pattern tree (FP-tree). Like the FP-growth algorithm, other algorithms [12]–[14] employ the pattern growth method to discover frequent itemsets.

In the second categorized algorithms, Generating frequent k-itemsets by intersecting the TID sets of every pair of frequent (k-1)-itemsets, which is the essence of the Eclat algorithm [15] by Zaki 2000. Eclat algorithm can recursively partition large classes into smaller ones until each class can be maintained entirely in the memory. Then, each class is processed independently in the breath-first fashion to compute the frequent itemsets. The main problem of Eclat is that when the intermediate results of vertical TID lists can become too large to be store in the memory. Burdick *et al.* propose to MAFIA [16] that converts the original data into binary vectors, and obtains the support through the ''and operation'', so as to improve the operation speed. When the data set is dense, these algorithms will generate a lot of redundant items, Pasquier *et al.* [17] propose closed frequent itemsets where frequent itemsets are computed. With the introduction of diffset technology, the algorithm [18] by Zaki that its memory requirements were reduced. It only keeps track of differences in the tids of a candidate pattern from its generating frequent patterns. The diffsets drastically cut down the size of memory required to store intermediate results.

Recently, to mine frequent itemsets in the presence of missing items and overcome these limitations of FT-Apriori, Shariq Bashir proposes FT-PatternGrowth [19], which adopts a divide-and-conquer technique and projects a big database into several databases and mines FT frequent itemsets in each small database. EAFIM [20] that uses the Apache Spark framework to achieve parallelism is an improved version of the apriori algorithm. Yasir, Muhammad, *et al.* propose the HARPP [21], which adopt the concern of pow set and dictionary data structures, and the D-GENE [22], which suspends the process of ITTL generation till the completion of transaction pruning phase, discovering frequent itemsets from sparse datasets.

The drawback of these methods is that it requires excessive time consumption or construct complex data structures [23] or dominates only in a specific scenario, so its efficiency needs to be improved. In this paper, we introduce the method(up-scaling) that computes frequent itemsets of the large-scale dataset depending on the frequent itemsets which belonged to small-scale datasets, not original data. So, our method is efficient and requires less memory consumption.

III. PROBLEM DESCRIPTION

Before presenting our problem statement, let's start with enlisting some necessary notations used in this paper.

A dataset D of size |D| consists of disjoint subsets D_1,D_2, . . . , D_k, whose size are $|D_1|$, $|D_2|$, . . . , $|D_k|$ respectively, where D is named large-scale dataset and D_i is named small-scale datasets. Let $T = \{I_1, I_2, \ldots, I_m\}$ be an itemset. every transaction element S of D is not empty such that

S ⊆ T. supND is defined occurrence frequency of some itemset S in D and support is the percentage of S appearing, that to say support = supND / |D|.

Definition 1 (Frequent Itemset of Small-Scale Datasets): if ∃x Let x ⊆ t, t ∈ D_i, $Tsupport_i$ computed by (1) that comes from [11] is minimum support threshold, p is probability parameter and adjustable and x satisfies x.surpport ≥ $Tsupport_i$, where Tsupport is minimum support threshold on large-scale dataset D, D_i is a small-scale datasets of D and x.supND$_i$ is occurrence frequency of x in D_i, x.supportD_i = x.supND$_i$ / |D_i|, all x form frequent itemsets of D_i, notated $SCFI_i$, otherwise, x does not satisfy x.surpportD_i ≥ $Tsupport_i$, x is infrequent itemset of small-scale datasets D_i.

$$Tsupport_i = Tsupport - \sqrt{\frac{1}{2|D_i|}.\ln\frac{1}{p}} \tag{1}$$

Definition 2 (The Estimated Value of Infrequent Itemset of the Small-Scale Datasets): if itemset x is infrequent of D_i but frequent of D_j, x's estimate value of occurrence frequency in D_i is as follows

$$\overline{x.\sup ND_i} = \sum\nolimits_{j\neq i, x\in SCFI_j} W_{ij} \times x.\sup ND_j \tag{2}$$

where W_{ij} is the similarity weight between D_i and D_j and $\sum W_{ij} \leq 1$

Definition 3 (Frequent Itemset of the Large-Scale Dataset): let the x's value of occurrence frequency in D is computed according to (3), if x.supND/|D| > Tsupport, then x is defined as frequent in D and all x form frequent itemsets of D, notated LCFI.

$$\text{x.}\sup ND = \sum_{i=1, x\in SCFI_i}^{n} x.\sup ND_i + \sum_{j=1, x\notin SCFI_j}^{n} \overline{x.\sup ND_j} \tag{3}$$

According to [24], it is obvious that if itemset x is frequent in D then there is at least one D_i that is small-scale datasets and x is also frequent in D_i.

Definition 4 (Potential Large-Scale Frequent Itemset(PLSFI)): Let PLSFI = $\cup_1^n${x is frequent in D_i}. then frequent itemset of the large-scale dataset(LSFI) LSFI ⊆ PLSFI.

Definition 5: (Similarity of Two Sets): if A and B are two definite sets, their similarity M(A, B) is as follows

$$M(A, B) = |A \cap B|/|A \cup B| \tag{4}$$

where |A| is the number of the elements in A.

Problem 1: let LSFI is the set of the frequent itemset in D. The ultimate objective of this paper is finding the function up-scaling (PLSFI, Tsupport) which can reveal all large-scale frequent itemsets in D, that is to say LSFI = up-scaling (PLSFI, Tsupport).

Fig. 1 shows two methods that can get the frequent itemsets of the large-scale dataset. It is the intuitional method that translating small-scale datasets into the large-scale dataset, then getting frequent itemsets from the large-scale dataset. Another method is finding frequent itemsets from the small-scale datasets, then translating the frequent itemsets which

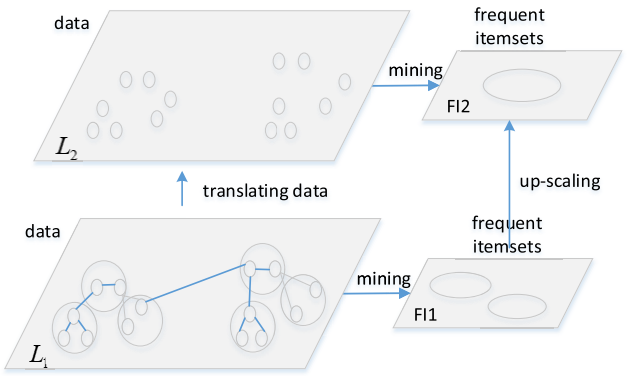


FIGURE 1. Two methods of acquiring frequent itemsets of the large-scale dataset.

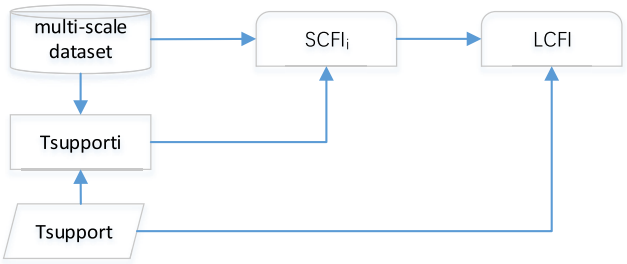


FIGURE 2. The overall framework of mining frequent itemsets of multi-scale dataset.

are mined from small-scale datasets into the final result that is frequent itemsets of the large-scale dataset. Our method is the latter. Specifically, we propose the up-scaling algorithm that depends on the frequent itemsets that have been mined from, the small-scale dataset, not the raw large-scale dataset.

IV. PROPOSED FRAMEWORK AND THE PROPOSED ALGORITHM

A. THE OVERALL FRAMEWORK

The ultimate objective of this paper is to reveal all frequent itemsets belonged large-scale dataset basing on the frequent itemsets which have been mined on small-scale datasets. We can observe the overall framework shown in fig. 2. In this paper, discovering the frequent itemsets of large-scale dataset is divided into the following five steps:

1) Mining the frequent itemsets for each small-scale dataset.

2) Calculating the similarity between small-scale datasets.

3) Constructing the potential frequent itemset of large-scale dataset.

4) Estimating the support value of some itemsets on small-scale datasets where they are infrequent.

5) Filtering frequent itemsets for the large-scale dataset.

B. PROPOSED ALGORITHM

Based on three formulas in section 3 and the overall framework, up-scaling frequent itemsets of the large-scale dataset is described in Algorithm 2. In particular, we use the

Algorithm 1 (Frequent Itemsets Mining of Small-Scale Datasets)

Input: $D_i(i = 1, \ldots, k)$, Tsupport, $p(0 < p < 1)$
Output: frequent itemsets on D_i $SCFI_i(i = 1, \ldots, k)$
1: foreach D_i do begin
2: computing $Tsupport_i$ according (1)
3: endfor
4: foreach D_i begin
5: $SCFI_i$ = getFrequentItemsets(D_i, $Tsupport_i$)
6: endfor

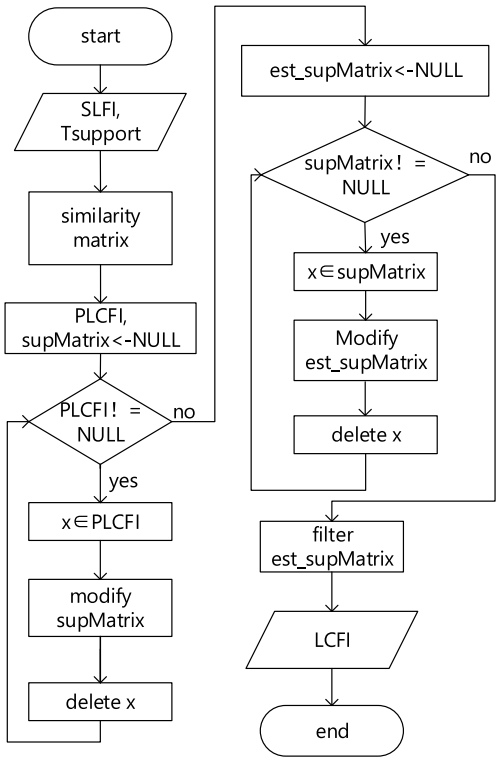


FIGURE 3. The flow diagram of the up-scaling algorithm.

similarity of frequent itemsets of the small-scale datasets instead of their similarity, in this algorithm.

Algorithm 1 products with the frequent itemsets of small-scale datasets using traditional data mining methods, which is the input data source of Algorithm 2.

The flow diagram of up-scaling frequent itemsets can be seen in Fig. 3. It is visible that the time consumption of the constructing supportMatrix and est_supMatrix is dominant in the up-scaling algorithm. The time complexity of them is same, $\Theta(m \times k)$, where m is the cardinality of the set of PLSFI, k is constant and $k \ll m$ on a specific scenario. memory consumption mainly consists of inputting small-scale frequent itemsets and constructing supportMatrix and est_supMatrix, so the space complexity of the up-scaling algorithm is Θ(ma-x(kFI, $m \times k$)), where kFI $= \sum_{i=1}^{k} |LCFI_i|$ and $|LCFI_i|$ is the cardinality of the set of $LCFI_i$.

Algorithm 2 (Up-Scaling Frequent Itemsets)

Input: $SCFI_i(i = 1, \ldots, k)$, Tsupport
Output: frequent itemsets on D(LSFI)
1: foreach $SCFI_i$, $SCFI_j$ do begin
2: computing $M_{ij} = M(SCFI_i, SCFI_j)$ according
3: endfor
4: PLCFI $= \bigcup_{i=1}^{k} SCFI_i$
5: foreach x ∈ PLCFI do begin
6: foreach $SCFI_i$ do begin
7: if x ∈ $SCFI_i$ then do $supx_i = x.supNDi$
8: else $supx_i = 0$
9: endfor
10: num = the number of that supxi is nozero
11: list(num, $supx_1, \ldots, supx_k$)
12: add list to supportMatrix
13: endfor
14: foreach x ∈ supportMatrix do begin
15: sum = 0
16: foreach $supx_i$ in x do begin
17: if supxi = 0 then do
18:

$$\text{est\_supxi} = |D_i| \times \frac{|D_i|}{m} \times \sum_{j=1}^{n} M_{ij} \times \frac{\sup xj}{|D_j|}$$

19: if $est\_supx_i \geq Tsupport_i \times |D_i|$ then do $est\_supx_i = Tsupport_i \times |D_i|$
20: else $est\_supx_i = supx_i$
21: sum = sum + est_supx_i
22: endfor
23: list(sum,$est\_supx_1, \ldots, est\_supx_n$)
24: add list to est_supMatrix
25: endfor
26: foreach x ∈ PLCFI, list_item in est_supMatrix do begin
27: add sum/|D| to list_item
28: if sum/|D| ≥ Tsupport then do add x to LCFI
29: endfor

C. CASE STUDY

To familiarize the readers with the proposed algorithm, we demonstrate the algorithm through the transactional datasets which consist of D_1, D_2, D_3 and D_4, and are shown by fig.4.

Let Tsupport = 0.6, p = 0.5, according to (1), we compute the support threshold of the small-scale datasets ($Tsupport_1 = 0.39$, $Tsupport_2 = 0.40$, $Tsupport_3 = 0.41$, $Tsupport_4 = 0.38$). In the first step, frequent itemsets of D_i are mined by apriori algorithm, as shown by fig.5.

In the second step, based on the four frequent itemsets as shown by fig.5 and according to Definition 5, the similarity matrix is computed as follows:

$$\begin{pmatrix} 1.0, 0.77, 0.64, 0.75 \\ 0.77, 1.0, 0.69, 0.54 \\ 0.64, 0.69, 1.0, 0.54 \\ 0.75, 0.54, 0.54, 1.0 \end{pmatrix}$$

TID	List of Item_IDs
T01	I1,I2,I4
T02	I1,I2,I3,I6
T03	I1,I2,I3,I4
T04	I1,I2
T05	I1,I2,I3,I4
T06	I1,I2,I4,I5
T07	I3,I4,I5
T08	I1,I2,I3,I4

D1

TID	List of Item_IDs
T01	I2,I3,I4
T02	I1,I2,I6
T03	I1,I2,I4
T04	I1,I2,I3,I4
T05	I1,I2,I3,I4
T06	I1,I2,I5
T07	I1,I2,I4
T08	I1,I2,I4
T09	I1,I2,I3,I4,I5

D2

TID	List of Item_IDs
T01	I1,I3,I4
T02	I1,I3,I4
T03	I1,I2,I3,I4
T04	I2,I3,I4
T05	I1,I3,I4
T06	I1,I2,I4
T07	I1,I2,I3,I4
T08	I1,I2,I4,I5
T09	I2,I3,I4,I5
T10	I2,I3,I4,I5

D3

TID	List of Item_IDs
T01	I1,I3,I4
T02	I1,I2,I3,I6
T03	I1,I2,I4
T04	I1,I2,I3
T05	I1,I2,I3,I4
T06	I1,I2
T07	I1,I2,I3,I5

D4

FIGURE 4. Sample of the transactions datasets.

ItemSet	SupportND1
{I1}	7
{I2}	7
{I3}	5
{I4}	6
{I1, I2}	7
{I1, I3}	4
{I1, I4}	5
{I2, I3}	4
{I2, I4}	5
{I3, I4}	4
{I1, I2, I3}	4
{I1, I2, I4}	5

ItemSet	SupportND2
{I1}	8
{I2}	9
{I3}	4
{I4}	7
{I1, I2}	8
{I1, I4}	6
{I2, I3}	4
{I2, I4}	7
{I3, I4}	4
{I1, I2, I4}	6
{I2, I3, I4}	4

ItemSet	SupportND3
{I1}	7
{I2}	7
{I3}	8
{I4}	10
{I1, I3}	5
{I1, I4}	7
{I2, I3}	5
{I2, I4}	7
{I3, I4}	8
{I1, I3, I4}	5
{I2, I3, I4}	5

ItemSet	SupportND4
{I1}	7
{I2}	6
{I3}	5
{I4}	3
{I1, I2}	6
{I1, I3}	5
{I1, I4}	3
{I2, I3}	4
{I1, I2, I3}	4

FIGURE 5. The support count of the small-scale data.

In the third step, based on every frequent itemsets of the small-scale datasets and according to Definition 4, the potential frequent itemset of the large-scale dataset(PLSFI) is constructed as follows: {{I1},{I2},{I3},{I4},{I1,I2},{I1,I3},{I1,I4},{I2,I4}, {I2,I3}, {I3,I4},{I1,I2,I3},{I1,I2,I4},{I1, I3,I4}, {I2,I3,I4}}

In the fourth step, we design the data structure which can save the support value of itemsets on every small-scale datasets, and estimate the support value of some itemsets of the small-scale datasets where they are infrequent, which are shown in Table 1 and Table 2.

In the last step, based on the support of the large-scale dataset in Table 2 and according to Definition 3, the final frequent itemsets of the large-scale dataset are held as follows: {{I1},{I2},{I3},{I4},{I1,I2},{I1,I4},{I2,I4}}.

V. EXPERIMENTS

To prove the effectiveness and efficiency of the up-scaling approach, we conducted two groups of experiments. The purpose of the first group of experiments is to compare the performance of the up-scaling algorithm against the dFIN algorithm [3] and negFIN algorithm [4], which are leading mining algorithms in the field of mining frequent itemsets at present. In the second experiment, we select Apriori [1] that is a classic frequent itemset mining algorithm as the baseline algorithm to verify the accuracy of the up-scaling algorithm.

A. DATA PREPROCESSING

Comparison experiments are assessed on five datasets, which consist of one synthetic dataset and four real datasets. The description of these datasets is shown in Table 3. To obtain the small-scale datasets, we divide every dataset into four non-overlapping partitions in two methods. The first method is in terms of equal interval, i.e., the first partition consists of record 1, record 5, and so on, the second partition consists of record 2, record 6, and so on, and the second method is

TABLE 1. The support value of itemsets on every small-scale datasets.

no	potential itemsets	supND1	supND2	supND3	supND4	num of Nozero
1	{I1}	7	8	7	7	4
2	{I2}	7	9	7	6	4
3	{I3}	5	4	8	5	4
4	{I4}	6	7	10	3	4
5	{I1,I2}	7	8	0	6	3
6	{I1,I3}	4	0	5	5	3
7	{I1,I4}	5	6	7	3	4
8	{I2,I3}	4	4	5	4	4
9	{I2,I4}	5	7	7	0	3
10	{I3,I4}	4	4	8	0	3
11	{I1,I2,I3}	4	0	0	4	2
12	{I1,I2,I4}	5	6	0	0	2
13	{I1,I3,I4}	0	0	5	0	1
14	{I2,I3,I4}	0	4	5	0	2

TABLE 2. The support value and estimate support value of the itemsets on every small-scale datasets.

no	potential itemsets	supND1	supND2	supND3	supND4	support
1	**{I1}**	7	8	7	7	**0.85**
2	**{I2}**	7	9	7	6	**0.85**
3	**{I3}**	5	4	8	5	**0.65**
4	**{I4}**	6	7	10	3	**0.76**
5	**{I1,I2}**	7	8	4.14	6	**0.74**
6	{I1,I3}	4	3.35	5	5	0.51
7	**{I1,I4}**	5	6	7	3	**0.61**
8	{I2,I3}	4	4	5	4	0.5
9	**{I2,I4}**	5	7	7	2.64	**0.64**
10	{I3,I4}	4	4	8	2.44	0.54
11	{I1,I2,I3}	4	3.12	3.15	4	0.42
12	{I1,I2,I4}	5	6	4.14	2.64	0.52
13	{I1,I3,I4}	2.57	3.12	5	1.88	0.37
14	{I2,I3,I4}	2.65	4	5	1.78	0.4

that the data are directly quartered. Then frequent itemsets are mined from the partitions basing different minimum support threshold and parameter p.

B. EXPERIMENTAL SETTINGS

We compare the performance that is runtime and memory consumption of our method against the negFIN algorithm.

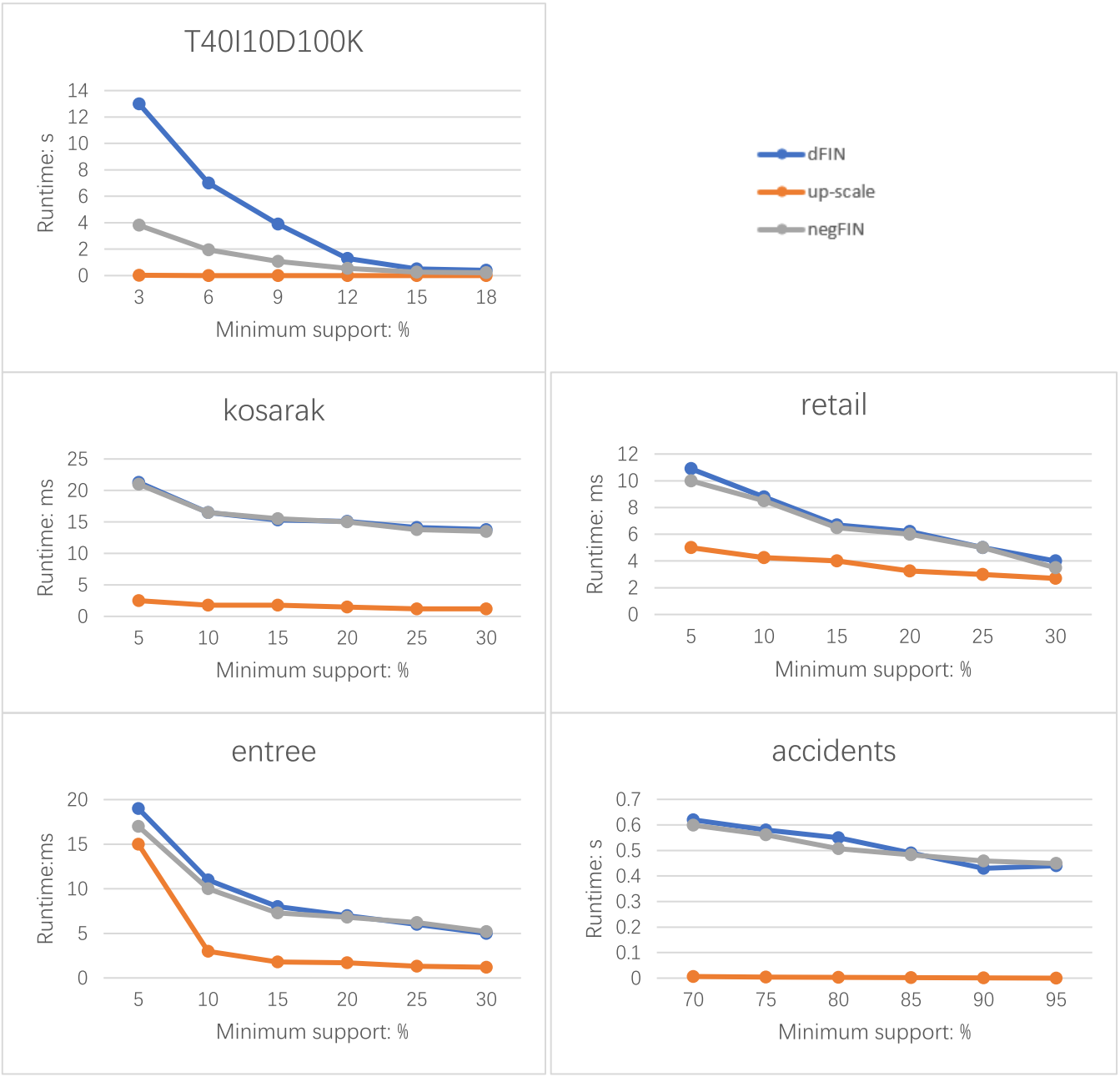


FIGURE 6. Runtime comparison of the up-scaling against negFIN and dFIN.

TABLE 3. Description of the datasets.

NO	Name of dataset	Type	|T|	|D|	Size of dataset (KB)
1	accidents	Real	34	340183	34678
2	entree	Real	11	4160	182
3	kosarak	Real	8	990002	31279
4	retail	Real	10	88162	4070
5	T40I10D100K	Synthetic	40	100000	15213

To make a fair comparison, these two algorithms have been run on the same hardware and software conditions. Our computer has the configuration of Inter(R) Core(TM) i7 Dual-Core processors running at 2.8GHz and 16G RAM, with the windows 10 x64 Home operating system. All algorithms are coded in C/C++.

C. RUNTIME COMPARISON

The runtime comparison of up-scaling against negFIN and dFIN is shown in fig. 6. In these figures, the X and Y axes are the minimum support threshold and runtime, respectively. As we know, with the increment in the value of the minimum support threshold, there is the corresponding decrement in execution time for these three algorithms. However, except for entree dataset, the running time of up-scaling on the

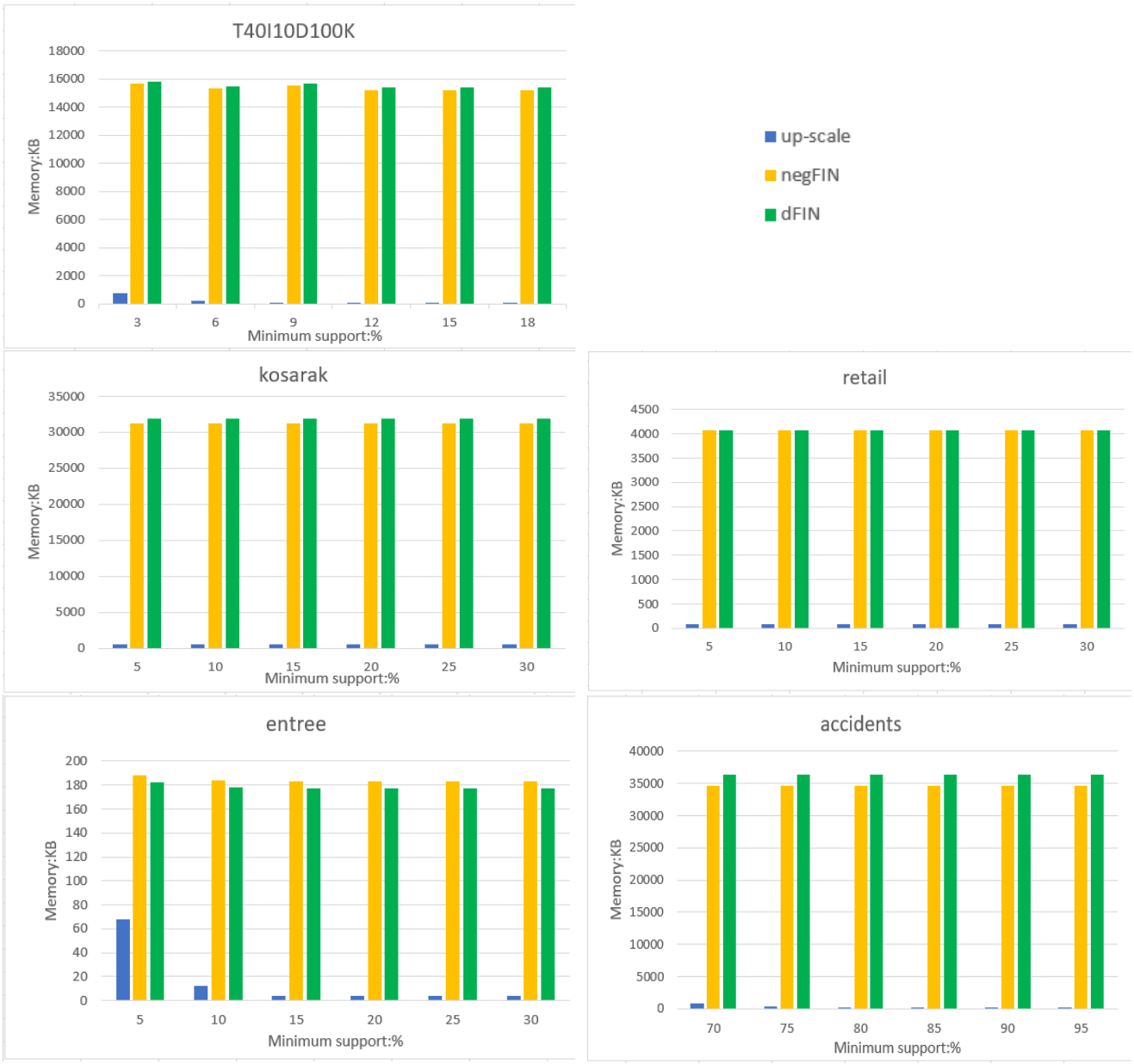


FIGURE 7. Memory consumption comparison of the up-scaling against negFIN and dFIN.

other four datasets vary very small. The main reason are that the number of the frequent itemsets that are belonged small-scale datasets on these datasets is small and the time overhead of our algorithm mainly depend on the frequent itemsets that are belonged small-scale datasets, not original data. Up-scaling faintly outperforms both the algorithms for the entree dataset when the minimum support value is set to 5%. Because, the number of the frequent itemsets severed our algorithm as input is rough the same as that of raw data in that case. NegFIN runs faster than dFIN on T40I10D100K dataset for lower minimum support, and the two algorithms spend almost the same time on the other four datasets. The reason is that the amount of which the negFIN drive NegNodeset is more than that the dFIN drives the DiffNodeset on T40I10D100K dataset, and the their amount is same roughly on other datasets.

As we can see in these figures, it is evident that up-scaling is more efficient than negFIN and dFIN. It should be noticed that runtime of up-scaling means the total execution time, which is the period between input and output of algorithm 2. Given different values of p, experiment 1 evaluates the performances of up-scaling on varying minimum support threshold, where our method can improve the CPU perfor-mance by an average of 73 percent against negFIN and 75 percent against dFIN in our experiments. In addition, we also notice the fact that for two different data partitioning methods, the running

T40I10D100K	min_sup:%	3	6	9	12	15	18
	p:%	5	10	15	20	25	30
kosarak	min_sup:%	5	10	15	20	25	30
	p:%	5	10	15	20	25	30
retail	min_sup:%	5	10	15	20	25	30
	p:%	5	10	15	20	25	30
entrée	min_sup:%	5	10	15	20	25	30
	p:%	5	10	15	20	25	30
accidents	min_sup:%	70	75	80	85	90	95
	p:%	5	10	15	20	25	30

FIGURE 8. Parameter values on five datasets.

min_sup / accur / p	accidents					
	70	75	80	85	90	95
5	100%	100%	100%	100%	100%	100%
10	100%	100%	100%	100%	100%	100%
15	100%	100%	100%	100%	100%	100%
20	100%	100%	100%	100%	100%	100%
25	100%	100%	100%	100%	100%	100%
30	100%	100%	100%	100%	100%	100%

FIGURE 9. The accuracy of the up-scaling algorithm on accidents, using the first partition method.

min_sup / accur / p	T40I10D100K					
	3	6	9	12	15	18
5	100%	100%	100%	100%	100%	100%
10	100%	100%	100%	100%	100%	100%
15	100%	100%	100%	100%	100%	100%
20	100%	100%	100%	100%	100%	100%
25	100%	100%	100%	100%	100%	100%
30	100%	100%	100%	100%	100%	100%

FIGURE 10. The accuracy of the up-scaling algorithm on T40I10D100K, using the second partition method.

time of the up-scaling algorithm is roughly equal. So, we do not distinguish the two cases in fig. 5.

D. MEMORY CONSUMPTION COMPARISON

Fig. 7 compares up-scaling with negFIN and dFIN. As we can see in this figure, the memory consumption of our algorithm is much less than negFIN and dFIN. Because our algorithm is mining the frequent itemsets of large-scale dataset from the frequent itemsets which are belonged to small-scale datasets, while the negFIN algorithm and the dFIN algorithm are directly mining on the raw dataset and our algorithm needs to construct the supportMatrix and the est_supMatrix that is main components of memory consumption of up-scaling when negFIN constructing set_enumeration_tree and frequent_itemset_tree [4] and

min_sup / accur / p	kosarak					
	5	10	15	20	25	30
5	100%	100%	100%	100%	100%	100%
10	100%	100%	100%	100%	100%	100%
15	100%	100%	100%	100%	100%	100%
20	100%	100%	100%	100%	100%	100%
25	100%	100%	100%	100%	100%	100%
30	100%	100%	100%	100%	100%	100%

FIGURE 11. The accuracy of the up-scaling algorithm on kosarak, using the second partition method.

min_sup / accur / p	retail					
	5	10	15	20	25	30
5	94%	100%	86%	75%	100%	100%
10	94%	100%	86%	75%	100%	100%
15	94%	100%	86%	75%	100%	100%
20	94%	100%	86%	75%	100%	100%
25	88%	100%	86%	75%	100%	100%
30	88%	100%	86%	75%	100%	100%

FIGURE 12. The accuracy of the up-scaling algorithm on retail, using the second partition method.

dFIN constructing PPC-tree [3]. It takes about the approximate space to construct data structures frequent_itemset_tree and PPC-tree, as we can see in this figure, the memory consumption of both algorithms is roughly the same. It is obvious that the frequent itemsets on these datasets are much smaller than the original dataset. In particular, the result that minimum support threshold is 5% on the entree dataset in fig. 7. shows that up-scaling consumes much more memory than other minimum support thresholds, that is because minimum support threshold is set to 5%, the entree dataset produces much more frequent itemsets which are belonged to small-scale datasets than for the other mini-mum support thresholds. It should be noted that for two different data partitioning methods, the up-scaling algorithm consumes almost the same memory. Therefore, we use one histogram depicting the two cases in fig. 7.

E. VALIDITY OF EFFECTIVENESS

In this part, we give parameter p and minimum support threshold six different values with regard to the same dataset, respectively, as shown by fig. 8. For the same dataset and every partition method, we used up-scaling and Apriori to conduct 36 experiments, respectively. Fig. 9. gives the specific results from accidents in terms of the first partition method. The accur is expressed as the percentage of $|A|/|B|$, where A and B are computed by up-scaling and Apriori, respectively. The accuracy on the other four datasets datasets partitioned by the first method is also 100% and the figure display is omitting here. That is to say that up-scaling

min_sup / accurp	entree					
	5	10	15	20	25	30
5	95%	95%	86%	93%	82%	57%
10	93%	93%	86%	93%	82%	57%
15	93%	92%	86%	93%	82%	57%
20	91%	90%	86%	93%	82%	57%
25	91%	89%	86%	93%	82%	57%
30	90%	89%	86%	93%	82%	57%

FIGURE 13. The accuracy of the up-scaling algorithm on entree, using the second partition method.

min_sup / accurp	accidents					
	70	75	80	85	90	95
5	91%	87%	85%	100%	100%	100%
10	90%	87%	83%	100%	100%	100%
15	89%	87%	83%	100%	100%	100%
20	89%	87%	83%	100%	100%	100%
25	89%	87%	83%	100%	100%	100%
30	89%	86%	83%	100%	100%	100%

FIGURE 14. The accuracy of the up-scaling algorithm on accidents, using the second partition method.

and Apriori discover the same frequent itemsets, which confirms the result generated by up-scaling in our experiments is effective.

For the second partition method, the accuracy of our algorithm is shown from fig. 10 to fig. 14. That is noticeable that up-scaling is 100% accurate on two datasets, partially 100 % on the other two datasets, and performs poorly on the entree. We guess the reason is the data distribution of the entree is very uneven.

VI. CONCLUSION AND FUTURE RESEARCH DIRECTIONS

Based on the requirements of mining frequent itemsets from different scale datasets, this paper proposes a new frequent itemsets mining framework. It just needs to mine all frequent itemsets on small-scale datasets, and then according to the frequent itemsets generate large-scale dataset's frequent itemsets, but no looking for frequent itemsets from the large-scale dataset, thereby reducing the operating costs. Because frequent itemsets decrease with the increase of minimum support threshold, our algorithm that needs to input frequent itemsets of small-scale datasets is especially suitable for situations with a high threshold. Experimental results show that the framework is feasible and effective.

In the future, our research directions as follows: (1) up-scaling the cluster centers in clustering tasks, (2) up-scaling function moving trends in regression analysis, (3) solving some problem of industrial recommender systems basing collaborative filtering (CF) [25]–[27].

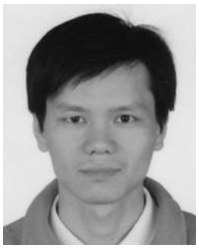
REFERENCES

[1] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *ACM SIGMOD Rec.*, vol. 22, no. 2, pp. 207–216, Jun. 1993.
[2] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Mining Knowl. Discovery*, vol. 8, no. 1, pp. 53–87, Jan. 2004.
[3] Z.-H. Deng, "DiffNodesets: An efficient structure for fast mining frequent itemsets," *Appl. Soft Comput.*, vol. 41, pp. 214–223, Apr. 2016, doi: 10.1016/j.asoc.2016.01.010.
[4] N. Aryabarzan, B. Minaei-Bidgoli, and M. Teshnehlab, "NegFIN: An efficient algorithm for fast mining frequent itemsets," *Expert Syst. Appl.*, vol. 105, pp. 129–143, Sep. 2018.
[5] J. T. Yao, A. V. Vasilakos, and W. Pedrycz, "Granular computing: Perspectives and challenges," *IEEE Trans. Cybern.*, vol. 43, no. 6, pp. 1977–1989, Dec. 2013.
[6] M. Yasir, M. A. Habib, M. Ashraf, S. Sarwar, M. U. Chaudhry, H. Shahwani , M. Ahmad, and C. M. N. Faisal, "TRICE: Mining frequent itemsets by iterative TRimmed transaction LattICE in sparse big data," *IEEE Access*, vol. 7, pp. 181688–181705, Dec. 2019.
[7] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases (VLDB)*, Sep. 1994, pp. 1–2.
[8] J. S. Park, M.-S. Chen, and P. S. Yu, "An effective hash-based algorithm for mining association rules," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, vol. 95, 1995, pp. 175–186.
[9] S.-Y. Wur and Y. Leu, "An effective Boolean algorithm for mining association rules in large databases," in *Proc. 6th Int. Conf. Adv. Syst. Adv. Appl.*, vol. 5, 1995, pp. 432–444.
[10] Y. Djenouri, D. Djenouri, J. C.-W. Lin, and A. Belhadi, "Frequent itemset mining in big data with effective single scan algorithms," *IEEE Access*, vol. 6, pp. 68013–68026, 2018.
[11] H. Toivonen, "Sampling large databases for association rules," in *Proc. 22th Int. Conf. Very Large Data Bases*, 1996, pp. 134–145.
[12] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang, "H-mine: Hyper-structure mining of frequent patterns in large databases," in *Proc. IEEE Int. Conf. Data Mining*, San Jose, CA, USA, 2001, pp. 441–448.
[13] G. Liu, H. Lu, Y. Xu, and J. X. Yu, "Ascending frequency ordered prefix-tree: Efficient mining of frequent patterns," in *Proc. 8th Int. Conf. Database Syst. Adv. Appl. (DASFAA)*, Kyoto, Japan, 2003, pp. 65–72.
[14] G. Grahne and J. Zhu, "Fast algorithms for frequent itemset mining using FP-trees," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 10, pp. 1347–1362, Oct. 2005.
[15] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 372–390, May 2000.
[16] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, and T. Yiu, "MAFIA: A maximal frequent itemset algorithm," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 11, pp. 1490–1504, Nov. 2005.
[17] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," in *Proc. ICDT*. Berlin, Germany: Springer, 1999.
[18] M. J. Zaki and K. Gouda, "Fast vertical mining using diffsets," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2003, p. 326.
[19] S. Bashir, "An efficient pattern growth approach for mining fault tolerant frequent itemsets," *Expert Syst. Appl.*, vol. 143, Apr. 2020, Art. no. 113046.
[20] S. Raj, D. Ramesh, M. Sreenu, and K. K. Sethi, "EAFIM: Efficient apriori-based frequent itemset mining algorithm on Spark for big transactional data," *Knowl. Inf. Syst.*, Apr. 2020, doi: 10.1007/s10115-020-01464-1.
[21] M. Yasir, "HARPP: HARnessing the power of power sets for mining frequent itemsets," *Inf. Technol. Control*, vol. 48, no. 3, pp. 415–443, 2019.
[22] M. Yasir, M. A. Habib, M. Ashraf, S. Sarwar, M. U. Chaudhry, H. Shahwani, M. Ahmad, and C. M. N. Faisal, "D-GENE: Deferring the GENEration of power sets for discovering frequent itemsets in sparse big data," *IEEE Access*, vol. 8, pp. 27375–27392, 2020, doi: 10.1109/ACCESS.2020.2971834.
[23] X. Han, X. Liu, J. Chen, G. Lai, H. Gao, and J. Li, "Efficiently mining frequent itemsets on massive data," *IEEE Access*, vol. 7, pp. 31409–31421, Mar. 2019.
[24] L. M. Goyal, M. M. S. Beg, and T. Ahmad, "An efficient framework for mining association rules in the distributed databases," *Comput. J.*, vol. 61, no. 5, pp. 645–657, May 2018.

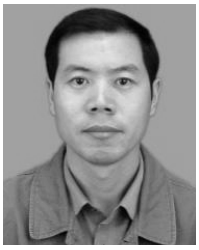
[25] X. Luo, M. Zhou, Z. Wang, Y. Xia, and Q. Zhu, "An effective scheme for QoS estimation via alternating direction method-based matrix factorization," *IEEE Trans. Services Comput.*, vol. 12, no. 4, pp. 503–518, Jul. 2019.

[26] X. Luo, M. Zhou, S. Li, Y. Xia, Z.-H. You, Q. Zhu, and H. Leung, "Incorporation of efficient second-order solvers into latent factor models for accurate prediction of missing QoS data," *IEEE Transactions on Cybernetics*, vol. 48, no. 4, pp. 1216–1228, Apr. 2018.

[27] X. Luo and M. Zhou, "Unconstrained non-negative factorization of high-dimensional and sparse matrices in recommender systems," in *Proc. IEEE 14th Int. Conf. Automat. Sci. Eng. (CASE)*, Munich, Germany, Aug. 2018, pp. 1406–1413.



RUNZI CHEN was born in Xinyang, Henan, China, in 1981. He received the B.S. degree in computer science and technology from Xinyang Normal University, China, in 2006, and the M.S. degree in software and theory of computer from Central China Normal University, China, in 2009. He is currently pursuing the Ph.D. degree in computational mathematics with the College of Mathematical and Information Science, Hebei Normal University, China. His current research interests include data mining and intelligent information processing.



SHULIANG ZHAO was born in Cangzhou, Hebei, China, in 1967. He received the B.S. degree in computer science and technology from East China Normal University, China, in 1990, the M.S. degree in software theory and method from Beihang University, China, in 2002, and the Ph.D. degree in computer science from the Beijing University of Technology, China, in 2006. He is currently a Professor with the School of Computer and Cyber Security, Hebei Normal University, China. His research interests include data mining and intelligent information processing. He is a member of CCF.

MENGMENG LIU was born in Zhangjiakou, Hebei, China, in 1988. She received the B.S. degree in software engineering and the M.S. degree in computer science and technology from Hebei Normal University, China, in 2012 and 2015, respectively. She is currently a Researcher with Zhangjiakou University. Her current research interests include data mining and intelligent information processing.