IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON INNOVATION AND APPLICATION OF INTELLIGENT PROCESSING OF DATA, INFORMATION AND KNOWLEDGE AS RESOURCES IN EDGE COMPUTING

Received November 7, 2019, accepted November 21, 2019, date of publication November 29, 2019,
date of current version December 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2956823

Incremental Association Rule Mining Based on Matrix Compression for Edge Computing

DONGDAI ZHOU[1], MENG OUYANG[1], ZHEJUN KUANG[2], ZHEN LI[1], JIN PENG ZHOU[3], AND XIAOCHUN CHENG[4], (Senior Member, IEEE)

[1]School of Information Science and Technology, Northeast Normal University, Changchun 130021, China
[2]College of Computer Science and Technology, Changchun University, Changchun 130012, China
[3]Division of Engineering Science, University of Toronto, Toronto, ON M5S2E8, Canada
[4]Faculty of Science and Technology, Middlesex University, London NW4 4BT, U.K.

Corresponding authors: Zhejun Kuang (kuangzhejun@ccu.edu.cn) and Xiaochun Cheng (x.cheng@mdx.ac.uk)

This work was supported in part by the National Natural Science Foundation of China under Grant 61977015, in part by the Twelfth Five-Year Plan of Jilin Provincial Department of Education under Grant 557, and in part by the EU FP7 Programme under Contract FP7-IP-608142.

ABSTRACT A growing amount of data is being generated, communicated and processed at the edge nodes of cloud systems; this has the potential to improve response times and thus reduce communication bandwidth. We found that traditional static association rule mining cannot solve certain real-world problems with dynamically changing data. Incremental association rule mining algorithms have been studied. This paper combines the fast update pruning (FUP) algorithm with a compressed Boolean matrix and proposes a new incremental association rule mining algorithm, named the FUP algorithm based on a compression matrix (FBCM). This algorithm requires only a single scan of both the database and incremental databases, establishes two compressible Boolean matrices, and applies association rule mining to those matrices. The FBCM algorithm effectively improves the computational efficiency of incremental association rule mining and hence is suitable for knowledge discovery in the edge nodes of cloud systems.

INDEX TERMS Edge computing, association rule, Boolean matrix, fast update pruning algorithm, matrix compression.

I. INTRODUCTION

Since Nature published its special issue on big data in 2008 [1], big data research has rapidly developed and become a hot research topic that has attracted substantial attention [2]–[4]. Currently, as an integral part of every industry and business function, data represent an important production factor [5]–[7]. The proliferation of the Internet of Things (IoT) and the success of cloud services have attracted research into new computing paradigms. Edge computing is being developed to achieve better response times with satisfactory safety and security [8]–[10]. However, data analysis solutions at edge nodes must use limited storage, limited computing power and limited communication bandwidth.

As an important field in data mining, the concept of association rule mining was first proposed by Agrawal *et al.* [11]. The purpose of association rule mining is to identify the connections among the items in a database and the hidden dependency relationships. This relevance of the data can bring great value to businesses [12]–[15]. The algorithmic complexity is important for edge computing [16]–[18]. Research on traditional association rule mining has mainly considered static databases [19]–[21]. However, most real-world transaction databases change continuously. Changes in transaction data mainly take on one of two forms: an increase or a decrease. Association rule mining in the first case is called incremental association rule mining.

The incremental association rule mining that is considered in this paper mainly relates to the case in which the data volume increases. When the size of the transaction database increases, if the minimum support threshold is calculated according to the percentage, then an initially frequent item may become an infrequent item, and an initially infrequent item may become a frequent item. The famous fast update pruning (FUP) algorithm has been used to solve this type of problem [22]. The FUP algorithm is a substantial improvement compared with traditional static association rule mining algorithms. First, the FUP algorithm prunes existing frequent items. Second, it receives new frequent itemsets from incremental databases. These advantages reduce

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

unnecessary calculations. However, two problems associated with the FUP algorithm need to be considered in the big data environment. On the one hand, the FUP algorithm needs to scan a database many times. The overhead that is needed for a large number of scans becomes a performance bottleneck whereby the results can be improved when the amount of transaction data is large. On the other hand, each iteration requires the entire transaction database to be scanned; thus, the search space of the solution is very large.

To solve these two problems, this paper combines the FUP algorithm with the idea of a compressed Boolean matrix and proposes the algorithm FUP based on a compression matrix (FBCM), which is a new, improved algorithm. The main advantages of the algorithm are as follows. First, the FBCM algorithm scans the transaction database only once, which reduces the time overhead. Second, the FBCM algorithm accelerates the computation using a binary AND operation to replace the string operation. Third, the FBCM algorithm compresses and prunes the transaction matrix in each iteration to further reduce the storage space and the number of computations.

This paper uses the contrast method to compare the performance of the FBCM algorithm. The experimental datasets are obtained from a frequent item mining warehouse. The experimental environment includes laboratory computers. A detailed configuration of the experimental environment and the experimental process are provided in the latter part of the paper. The comparative analysis is carried out from multiple dimensions, including a comparison between an incremental algorithm and a non-incremental algorithm and a comparison between a matrix algorithm and a non-matrix algorithm. The experimental results reflect the advantages of the incremental algorithm compared with the non-incremental algorithm. The results also show that the above-mentioned three advantages improve the performance of the FBCM algorithm. In addition, the greater the number of iterations is, the more obvious the advantage of the FBCM algorithm.

This paper is divided into five sections. The first section introduces the research background, the limitations of the FUP algorithm and the advantages of the FBCM algorithm. The second section presents relevant work. The third section introduces the formalisation of the concepts, including the concept of the incremental association rules, a review of the FUP algorithm and the general procedure of the matrix-based association rule mining algorithm. The fourth section describes the FBCM algorithm in detail, introduces the algorithm's main steps and explains the algorithm through a specific case study. The fifth section presents experimental details for comparison of the FBCM algorithm with the classical FUP algorithm to demonstrate the advantages of the FBCM algorithm.

II. RELATION WORK

We review the main studies on incremental association rule mining methods. References [23]–[25] summarized the research methods for incremental association rule mining. The current main research methods can be divided into two main types.

The first type involves mining based on tree structures. Representative mining methods are the compressed and arranged transaction sequences (CATS) tree algorithm [26] and the canonical-order tree (CanTree) algorithm [27]. The CATS Tree algorithm is based on the frequent pattern tree (FP-Tree) [28] and the prefix tree, and it was proposed by Cheung et al. in 2003. The algorithm scans the data only one time and establishes a CATS tree. The algorithm then mines the tree scanning data only once, which makes it easy to add and delete data. The algorithm has an obvious problem, i.e., the greater the influence of the transaction sequence is, the greater the number of tree branches and the more complex the structure. Based on the CATS tree, Leung and others proposed an improved algorithm named CanTree in 2005 [27]. Unlike the CATS tree, CanTree requires the nodes to be sorted in a certain order, such as dictionary order, before construction such that adding and deleting operations will not cause changes in the tree structure. Although the incremental association rule solution based on trees does not require the frequent scanning of the transaction database, a disadvantage is often observed. In particular, under large amounts of data, the irregular tree structure itself will be very complex, and the implementation costs will be relatively high. This condition is not promising for solving problems in contemporary big data environments.

The other type includes mining algorithms based on iteration. The main performance bottleneck of the FUP algorithm, which was proposed by Cheung *et al.* [22], is the overhead that is caused by conducting multiple scans of a transaction database. The FUP algorithm is used to mine the association rules when the data volume is increasing. For the case with a decreasing amount of data, Cheung proposed the FUP2 algorithm, which is based on the FUP algorithm [29]. The algorithm is similar to the FUP algorithm when the amount of data is increasing. While the FUP2 algorithm can address reductions in the amount of data, a performance bottleneck occurs similar to that of the FUP algorithm.

To address the problem resulting from the frequent scanning of the transaction database, most studies are currently conducted in a mining environment based on classical non-incremental association rules. Reference [30] provided a detailed summary of classical non-incremental algorithms, of which the most popular algorithm is the Apriori algorithm [31]. References [32]–[34] proposed an association rule mining algorithm based on the Boolean matrix. The main idea to generate a Boolean matrix with transaction data as the rows and all individual items as the columns after scanning the transaction database once instead of scanning the database based on the matrix operation. Essentially, the iterative process of this algorithm is the Apriori algorithm; however, it reduces the costs of the frequent scanning of the operation. The limitation is that the iterative process, i.e., the Apriori algorithm, produces a number of infrequent candidates and wastes valuable computing resources. To address the

bottleneck of the algorithm, Liu and Wang [35] improved the Apriori algorithm based on a matrix. The main process was to introduce the concept of matrix compression, adopt the corresponding compression strategy based on the Boolean matrix affair after each iteration, and delete the rows and columns that conform to the strategy. Compared with the work in [32]–[34], reference [35] reduced the number of infrequent candidate items and the computational complexity via matrix compression. Based on the idea of matrix compression, the studies in [36]–[38] proposed various improvement schemes, which improved matrix compression in many aspects, including matrix storage, itemset sorting, the compression matrix, and support computations, thus reducing the time and spatial costs of the algorithm.

The studies in [32]–[35] proposed a matrix association rule mining algorithm and a variety of improvement strategies for static databases. The algorithm complexity is also a problem that must be considered; the studies in [39]–[41] improved the fault tolerance and robustness of the algorithm. However, incremental matrix mining algorithms are currently relatively scarce. Therefore, this paper proposes the FBCM algorithm, which is an incremental association rule mining improvement algorithm based on matrices. The technique first introduces the matrix method to the FUP algorithm to solve the bottleneck problem caused by the frequent scanning of the transaction database and then compresses the matrix in each iteration to further improve the mining efficiency of the algorithm by reducing the search space of the solution. The FBCM algorithm is designed to solve the problem of incremental association rule mining in the case of a growing database.

III. FORMALISATION OF THE CONCEPTS

A. INCREMENTAL ASSOCIATION RULE MINING

The essence of association rule mining is to discover the frequent itemsets in transaction data. The concept of association rule mining can be described as follows [42]:

Definition 1: Let the dataset $DB = \{T_1, T_2, T_3, \ldots, T_n\}$. The size of the dataset is $N = |DB|$, and the transactions $T_i \in DB$, where $I = \{I_1, I_2, \ldots, I_m\}$, are an itemset of transactions, and $T_i \subset I$. For Itemset $X, Y (X \subset I, Y \subset I)$ and $X \bigcap Y = \emptyset$, $X \Rightarrow Y$ are the association rules. The number of times that X appears in the dataset is $count(X)$, and its support is $support(X) = count(X)/N$. For the minimum support minsup, if $support(X) \geq minsup$, it is regarded as a frequent item.

In the association rule mining research, the main task is to identify all frequent itemsets in a fast and efficient manner. The fundamental properties of association rules are often used in this process. Specifically, a subset of frequent itemsets is also a frequent itemset, and a superset of the infrequent itemsets is an infrequent itemset. This property can help in pruning the candidates during the mining process to sidestep a large number of calculations.

As previously mentioned, we generally call the above description static association rule mining. However, the size of a database is not static in most real situations since the amount of data will increase or decrease. For example, a shopping list tends to increase in commodity mining. The requirements of the mining can also change, such as by strengthening the connections between items that require mining, which corresponds to increasing the support threshold. Incremental association rule mining involves the mining of changes in the data quantity or changes in the support degree. In this paper, we mainly study association rule mining in the case of an increasing amount of data. The incremental database is represented by db, and the size of the dataset is $n = |db|$.

TID	Itemsets
T_1	a, c, e
T_2	b
T_3	b, c
T_4	a, c, d, e

→

	a	b	c	d	e
T_1	1	0	1	0	1
T_2	0	1	0	0	0
T_3	0	1	1	0	0
T_4	1	0	1	1	1

FIGURE 1. General transformation rule based on a matrix.

B. ASSOCIATION RULE MINING BASED ON A MATRIX

According to [32]–[34], we know that mining association rules based on a Boolean matrix are generally divided into two steps. The first step is to build a Boolean matrix and set up the transaction database $DB = \{T_1, T_2, T_3, \ldots, T_n\}$ and itemset $I = \{I_1, I_2, \ldots, I_m\}$. For a transaction item that contains n transactions and m entries, a Boolean matrix M is established, with transactions as the rows and entries as the columns. The established rule is as follows. Every transaction is scanned. When an item appears in a transaction, the coordinate value is 1; otherwise, it is 0. The first step is shown in Figure 1.

In Figure 1, f represents the mapping of the transaction matrix D to the Boolean matrix M. The value in the first row and first column is "1", representing the existence of item a in transaction T_1, and the value in the second row and first column indicates that item a does not exist in the transaction T_2. Similarly, if the value in column j and line i is "1", then the term that is represented by column j in the matrix exists in the transaction T_i and vice versa. The second step is frequent item mining based on the Boolean matrix via various mining strategies. The mining process mainly considers the nature of a vector. The association rule mining algorithm based on a matrix depends on the following four properties:

Property 2: The number of "1"s in the column vector represents the support count of the column in the matrix [32].

As an example of column a in the above graph, the number of "1"s in column a is $count(a) = 2$; thus, the support of term a is $support(a) = 2$.

Property 3: In a matrix, if the result of the AND operation between column i and column j is the column vector k, and the number of "1"s in the column vector k represents the support count of two frequent itemsets $< i, j >$ [32].

Consider column a and column d in Figure 1. The result of the AND operation between column a and column d is 0001; thus, the support count of ad is $support(ad) = 1$.

If two columns have a "1" in the same row, then the representative items of the two columns appear together in the transaction; thus, the support count should be increased by one. All other cases indicate that the representative items of the two columns do not appear together. When the two operands are "1", the result of the AND operation is "1". If one of the two operands is "0", then the result of the AND operation is "0". This phenomenon is a feature of the AND operation.

Property 4: For the Boolean matrix, if the number of column vectors with a value of "1" is less than the minimum support threshold s in the kth iteration, then the column can be deleted [35].

Property 5: For the Boolean matrix, if the number of row vectors with a value of "1" is less than k in the kth iteration, then the row can be deleted [35].

Property 2 and Property 3 are the basic properties of association rule mining based on matrices, while Property 4 and Property 5 are the matrix compression strategies that were proposed in [35]. The compression strategy in the static association rules is effective. Using these two properties, this paper will incorporate the properties that are suitable for incremental association rule mining. The details of the new properties and their validity will be given in the next section.

C. FUP ALGORITHM

The FUP algorithm is a classical incremental association rule mining algorithm. The algorithm mainly solves the association rule mining problem under the condition of an increase in the amount of data [22]. The algorithm can be divided into two steps. The first step uses existing frequent itemsets $L(DB)$ to determine and delete the items of infrequent itemsets that are called losers, which the original frequent itemsets become. The remaining items in $L(DB)$ are called winners. Winners are added to the frequent itemsets $L(DB + db)$, and then, a new winner is found in the incremental database db (the original infrequent itemsets become frequent itemsets), which is then added to the total number of frequent itemsets $L(DB+db)$. The new frequent itemsets can be generated only in the original DB frequent itemset $L(DB)$ and the new data db frequent itemset $L(db)$. The process is shown in detail in the following pseudocode:

Compared with the Apriori algorithm, the FUP algorithm does not need to calculate all candidates from the beginning. The FUP algorithm uses several existing frequent itemsets to perform many iterations of pruning to somewhat reduce the number of unnecessary scanned operations. The basis of this pruning mainly involves the following two properties (for details regarding the specific correctness proofs, please read the original text regarding the FUP algorithm.)

Property 6: The necessary and sufficient condition of itemset X becoming a frequent itemset is $SUP_{DB+db}(X) = (|DB| + |db|) * s$ [22].

Property 7: If itemset X does not belong to a primary frequent itemset, then the necessary condition for itemset X to become a frequent itemset is $SUP_{db}(X) \geq |db| * s$ [22].

Algorithm 1 FUP Algorithm

Input: Original dataset, DB
Frequent itemsets in DB: $L(DB)$
Incremental dataset: db
Support threshold: s
Output: Frequent itemsets on DB + db: $L(DB + db)$

```
L(db) = U_k L_k(db)
for all item in L(db) do
    if item ∈ L(DB) then
        add(item) to L(DB + db)
        delete(item) from L(db)
// Frequent itemset item ∈ L(db)-L(DB)
for all T in DB do
    if T contain(item) then
        Item.count + +
L(DB + db)+ = Item(item.count ≥ s)
// Frequent itemset item ∈ L(DB) -L(db)
for all item in L(DB) do
    for All T in db do
        if T contain(item) then
            Item.count + +
L(DB + db)+ = Item(item.count ≥ s)
return L(DB + db)
```

The experimental results obtained by Cheung *et al.* [22] indicate that the FUP algorithm is faster than the Apriori algorithm and prove that FUP is more suitable for solving the problems of incremental association rule mining when the amount of data increases. Based on property 6, FBCM performs the matrix operation of incremental association rule mining. Property 7 provides a pruning method for the FBCM algorithm and reduces the computational complexity. The details of the FBCM algorithm are introduced in the next section.

IV. FBCM ALGORITHM

A. ALGORITHM DESCRIPTION

The FBCM algorithm is an incremental association rule mining algorithm that is used to solve the problem of increasing the amount of data. Based on the FUP algorithm, the algorithm transforms the FUP scan operation into the calculation of a matrix. The first step transforms the original transaction database DB and incremental database db into two transaction matrices. This step transforms the database DB into a matrix DB and transforms database db into a matrix db. This step is one of the main aspects of this paper. To scan the database only once to generate a Boolean matrix, we need to transform the matrix once. The second step performs association rule mining based on the transaction matrix. In the mining process, each iteration requires that two matrices be separately compressed to reduce the search space of the solution. The detailed pseudo-code of the algorithm is as follows.

Algorithm 2 FBCM Algorithm

Input: Original dataset, DB
Frequent itemsets on DB: $L(DB)$
Incremental dataset: db
Support threshold: s
Output: Frequent itemsets on DB + db: $L(DB + db)$

1: $MatrixDB = convert(DB), Matrixdb = convert(db)$
2: $L_1(db) = getL_1(Matrixdb)$
3: **for** all items in $L_1(db) \cap L_1(DB)$ **do**
4: add(item) to $L_1(DB + db)$
5: delete(item) from $L_1(db)$
6: **for** all items in $L_1(db) - L_1(DB)$ **do**
7: $item.count+ = MatrixDB.getCount(item)$
8: $L_1(DB + db)+ = item(item.count \geq s)$
9: **for** all items in $L_1(DB) - L_1(db)$ **do**
10: $item.count+ = Matrixdb.getCount(item)$
11: $L_1(DB + db)+ = item(item.count \geq s)$
//Compressing matrix based on frequent "1" itemsets
12: $MatrixDB = compress1(MatrixDB, L_1(DB + db))$
13: $Matrixdb = compress1(Matrixdb, L_1(DB + db))$ // Frequent multiple set calculation, first compression and recalculation
14: $MatrixDB = compress2(MatrixDB, s, k)$
15: //s: support threshold. k: frequent K itemsets
16: $Matrixdb = compress2(Matrixdb, s, k)$
17: $L(db) = U_k L_k(db)\ /k \geq 2$
18: **for** all item in $L(db) \cap L(DB)$ **do**
19: add(item) to $L_1(DB + db)$
20: delete(item) from $L(db)$
21: **for** all item in $L(db) - L(DB)$ **do**
22: $item.count+ = MatrixDB.getCount(item)$
23: $L(DB + db)+ = item(item.count \geq s)$
24: **for** all item in $L(DB) - L(db)$ **do**
25: $item.count+ = Matrixdb.getCount(item)$
26: $L(DB + db)+ = item(item.count \geq s)$
27: **return** $L(DB + db)$

Research on the FBCM algorithm emphasizes two main points. The first point is the transformation of the matrix. Step one of the pseudocodes expresses that the transaction database is transformed into a matrix via one database scan. The detailed transformation process is shown in Figure 2, where the numerical record is the transaction number. This is an ordered process. For example, I_2 is in T_2 and T_3; thus, the corresponding numerical vector of I_2 is $< 2, 3 >$. Then, the tensile transformation of the numerical matrix is carried out. Four transactions exist in the database, and each vector should be stretched to 4. In the same way, taking I_2 as an example, the numbers 2 and 3 exist in the vector I_2. The corresponding positions of numbers 2 and 3 should be replaced with a 1, while the other positions should be replaced with a 0; thus, $< 2, 3 >$ is stretched to $< 0, 1, 1, 0 >$.

The second research emphasis is matrix compression. The previous step constructs a complete transaction matrix.

	I_1	I_2	I_3	I_4	I_5	r.count
T_1	1	0	0	1	0	2
T_2	0	1	1	0	1	3
T_3	1	1	1	0	1	4
T_4	0	1	0	0	1	2
T_5	1	1	1	0	0	3
c.count	3	4	3	1	3	

FIGURE 2. Boolean matrix matrix *DB*.

This step will compress the entire matrix with the following method, which is of one of two types. The first type involves deleting the column in which the number of itemsets is not "1" in the Boolean matrix after confirming that the number of itemsets is 1. The compression process is shown in steps 6 and 7 of the pseudocode. From the basic nature of the frequent itemsets, a subset of frequent itemsets is necessarily a frequent itemset, and a superset of infrequent itemsets is an infrequent itemset. The second compression is for those frequent itemsets whose number is 2, and compression occurs before the subsequent iterations, which is represented by steps 8 and 9 of the pseudocode.

When the number of iterations is k, if the count of a row vector whose value is "1" is less than k, then the row can be deleted. In addition, if the number of column vectors whose number is "1" in the two matrices is less than the support threshold, then the column can be deleted. In the FBCM algorithm, the matrix compression strategy is mainly dependent on the following two properties:

Property 8: For the original Boolean matrix and the incremental Boolean matrix, if the number of row vectors whose number of "1" is less than k, the row is deleted.

Proof of Property 9: According to Property 5, we know that for any row X, the necessary and sufficient condition for X not being deleted is $count(X) \geq k$, where $count(X)$ is the number of row vectors X, whose number is "1" in the matrix. Assuming that the original Boolean matrix is M_1, the incremental Boolean matrix is M_2, and k represents the number of iterations. For any row Y, this can be divided into three situations. (1) Y exists in M_1 but does not exist in M_2; if $count(Y) < k$, row Y in M_1 can be deleted since property 5 is satisfied. (2) Y exists in M_2 but does not exist in M_1; if $count(Y) < k$, it can be deleted similarly. (3) Y exists in both M_1 and M_2. The original Boolean matrix M_1 and the incremental Boolean matrix M_2 are independent. Therefore, the same rules apply here.

Property 10: For a Boolean matrix, for k iterations, if the sum of the number of matrix DB column vectors whose number is "1" and the number of matrix db column vectors whose number is "1" is less than the minimum support threshold $s * (|DB| + |db|)$, the column is deleted.

Proof of Property 11: If item $X \subset I$, then $support_{DB+db}(X) = support_{DB}(X) + support_{db}(X)$. From Property 2 of the vector, $support_{DB}(X)$ is the number of column vectors X whose number is "1" in the matrix DB.

TABLE 1. Original database *DB*.

TID	Item Set
T_1	I_1, I_4
T_2	I_2, I_3, I_5
T_3	I_1, I_2, I_3, I_5
T_4	I_2, I_5
T_5	I_1, I_2, I_3

TABLE 2. Incremental database *db*.

TID	Item Set
T_1'	I_2
T_2'	I_2, I_3, I_5
T_3'	I_1, I_2, I_4, I_5

	I_1	I_2	I_3	I_4	I_5	r.count
T_1'	0	1	0	0	0	1
T_2'	0	1	1	0	1	3
T_3'	1	1	0	1	1	4
c.count	1	3	1	1	2	

FIGURE 3. Boolean matrix matrix *db*.

	I_2	I_5	r.count
T_1	0	0	0
T_2	1	1	2
T_3	1	1	2
T_4	1	1	2
T_5	1	0	1
c.count	4	3	

FIGURE 4. Matrix *DB* after the first compression.

	I_2	I_5	r.count
T_1'	1	0	1
T_2'	1	1	2
T_3'	1	1	2
c.count	3	2	

FIGURE 5. Matrix *db* after the first compression.

	I_2	I_5	r.count
T_2	1	1	2
T_3	1	1	2
T_4	1	1	2
c.count	3	3	

FIGURE 6. Matrix *DB* after the second compression.

Similarly, $support_{db}(X)$ is the number of column vectors X whose number is "1" in the matrix *db*. Therefore, $support_{DB+db}(X)$ = the number of column vectors X whose number is "1" in the matrix DB + the number of column vectors X whose number is "1" in the matrix *db*. If the right side of the equation is less than the minimum support threshold, then $support_{DB+db}(X)$ is less than the minimum support threshold; thus, item X is unlikely to be in the frequent "1" itemset. According to the frequent items, itemsets including X cannot become frequent itemsets and thus can be deleted.

Based on properties 8 and 10, the FBCM algorithm compresses the original data and incremental data via matrix compression. Most of the invalid data are pruned by matrix compression, which greatly reduces the cost of the subsequent kth iteration. For sparse data sets, the compression effect is more obvious. The efficiency of the algorithm after matrix compression will be verified experimentally in the fifth section of this paper.

B. EXPLANATION USING AN EXAMPLE

This section explains the key steps of the FBCM algorithm via a simple example. The known conditions are as follows. The support threshold is set as $s = 0.4$. The original transaction database *DB* is shown in Table 1. The number of transactions is $N = 5$, and $M = 5$. The incremental database *db* is shown in Table 2. The number of transactions is $n = 3$, and $m = 5$. The frequent itemsets in the *DB* include $L_1(DB) = \{I_1 : 3, I_2 : 4, I_3 : 3, I_5 : 3\}$ and $L_2(DB) = \{I_1I_2 : 2, I_1I_3 : 2, I_2I_3 : 3, I_2I_5 : 3, I_3I_5 : 2\}$. The first step of the algorithm is to transform *DB* and *db* into Boolean matrices, as shown in Figures 2 and 3.

We now present the calculation of the frequent "1" itemsets. We know that $L_1(db) = \{I_2 : 3, I_5 : 2\}$ based on c.count in the matrix *db* and that $L_1(DB) = \{I_1 : 3, I_2 : 4, I_3 : 3, I_5 : 3\}$ based on the minimum support count of the frequent itemset $(|DB| + |db|) * s = 3.2$. Finally, we obtain $L_1(DB + db) = \{I_2 : 7, I_5 : 6\}$. Since the superset of infrequent itemsets is an infrequent itemset, no items except for I_2 and I_5 need to be calculated in the subsequent iterations. The other set can be deleted using the compressed matrix. The matrices after the first round of compression are depicted in Figures 4 and 5.

Before mining the frequent 2 itemsets, we need to compress the matrix. Similarly, in subsequent iterations, we need to compress the matrix before each iteration. We take the mining process of the frequent 2 itemsets as an example to introduce the compression operation. First, we delete all rows of $r.count < 2$, i.e., we delete the rows of T_1, T_5 and T_1' in this case. Then, we delete the column of $matrixDB(c.count) + matrixdb(c.count) < (|DB| + |db|) * s = 3.2$. Nothing needs to be deleted in this example. The results are shown in Figures 6 and 7.

	I_2	I_5	r.count
T_2'	1	1	2
T_3'	1	1	2
c.count	2	2	

FIGURE 7. Matrix *db* after the second compression.

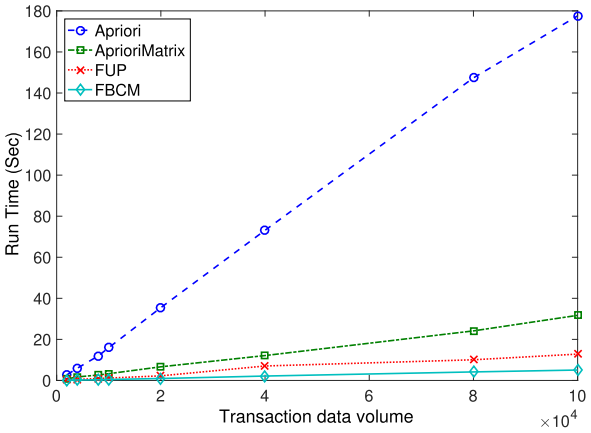


FIGURE 8. Experimental contrasting diagram under different data volumes.

Now, we extract the frequent 2 itemsets based on the matrix. We obtain $L_1(DB + db) = \{I_2 : 7, I_5 : 6\}$. The db frequent 2 itemset candidates are generated via the result of $L_1(DB + db)$; thus, the candidates are only $< I_2I_5 >$. The support degree of $< I_2I_5 >$ in *db* is 2 based on the AND operation between column I_2 and column I_5 in the matrix *db*. Given $L_2(DB) = \{I_1I_2 : 2, I_1I_3 : 2, I_2I_3 : 3, I_2I_5 : 3, I_3I_5 : 2\}$, we can determine that $Support_U(I_2I_5) = 5$ and that the count of the matrix *db* is always 0 after the compression of the other items in $L_2(DB)$. Finally, we obtain $L_2(DB+db) = \{I_2I_5 : 5\}$. There are only two columns in the matrix, and the frequent 3 itemsets are empty; thus, the mining ends.

V. EXPERIMENT AND RESULT ANALYSIS

The test environment utilizes the Windows 10 64-bit operating system and 8 GB of memory. The CPU is an Intel Core i7-3770 operating at 3.40 GHz.

Experiment 12: The experiment uses the T10I4D100K dataset. The incremental database *db* is acquired by sampling on the T10I4D100K dataset, and |db| = 100. The given minimum support threshold is $s = 0.02$. The incremental database *db* corresponds to the original transaction database *DB* obtained from sampling on T10I4D100K. Using the run time, this experiment compares the execution efficiency of the Apriori algorithm, the Apriori matrix algorithm, the FUP algorithm and the FBCM algorithm.

Since this study is based on the idea of the classical non-incremental algorithm in many ways, the comparison in experiment 12 includes the classical non-incremental Apriori algorithm. Experiment 12 compares the matrix algorithm

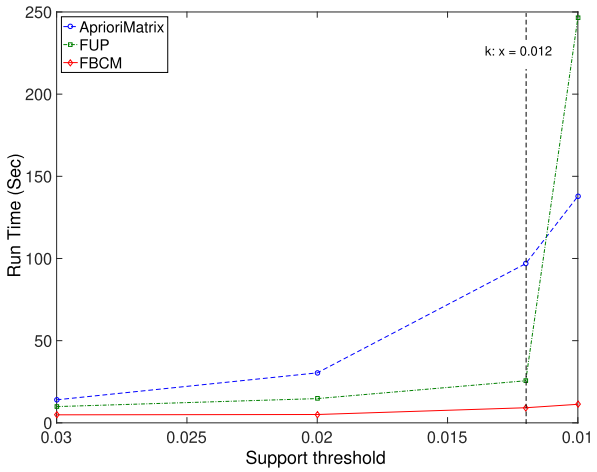


FIGURE 9. Diagram of the experimental comparison under different support thresholds.

based on an Apriori method called the Apriori Matrix algorithm, which was proposed in [32]; the FUP incremental algorithm; and the FBCM algorithm proposed in this paper. The experimental results are shown in Figure 8. The abscissa indicates that the transaction data volume increases from $0.2 * 10^4$ to $10 * 10^4$. The ordinate indicates the runtimes of the algorithms. When the transaction data volume is $0.2*10^4$, the times required by the Apriori algorithm, Apriori Matrix algorithm, FUP algorithm and FBCM algorithm are 2.79, 0.77, 0.29, and 0.15 s, respectively. When the transaction data volume increases to 10×10^4, the times needed by the four algorithms are 177.58, 31.71, 12.83, and 5.07 s, respectively.

The experimental results show that under the experimental conditions, the Apriori algorithm is the slowest. The Apriori Matrix algorithm is the second slowest; however, it has been greatly improved compared with the Apriori algorithm. These first two algorithms are non-incremental algorithms. The FUP algorithm is superior to the two Apriori algorithms on the incremental problem. The efficiency of the FBCM algorithm is the highest. In addition, when the transaction data volume increases, the gap between these algorithms becomes increasingly obvious. The insufficient speed of the Apriori algorithm affects the results of the experiment; hence, the comparison with this algorithm will be omitted in the follow-up experiment.

Experiment 13: This experiment considers the T10I4D100K dataset. The incremental database *db* is acquired by sampling on the T10I4D100K dataset, and |db| = 100. The given original database is T10I4D100K, and the incremental database is *db*. With the minimum support threshold *s* as the abscissa, this experiment compares the execution efficiency of the Apriori Matrix algorithm, the FUP algorithm and the FBCM algorithm in terms of time.

Figure 9 presents the runtime of the Apriori Matrix, FUP and FBCM algorithms when the support threshold changes from 0.01 to 0.03. In this experimental environment, when the threshold is $s = 0.03$, the runtimes of the Apriori Matrix

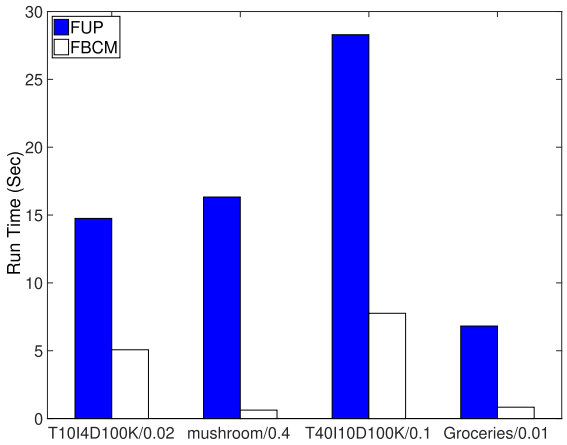


FIGURE 10. Diagram of the experimental comparison under different datasets.

algorithm, FUP algorithm and FBCM algorithm are 14.05, 9.89 and 4.08 s, respectively. The computation speed of the FBCM algorithm is higher than that of the FUP algorithm. At this point, there are few frequent itemsets in the incremental datasets. The FUP algorithm does not need to frequently scan the *DB* transaction datasets. However, the first step of the FBCM algorithm scans the transaction data to construct a matrix, which requires a non-negligible amount of time. Hence, the advantage is not obvious.

When the support threshold is reduced to 0.012, the three curves all have an obvious inflection point. The three intersecting points between the vertical line K and the curves are shown in Figure 9. When $s < 0.012$, the runtime of the FUP algorithm rapidly increases.

To achieve point P in the figure, the runtime of the FUP algorithm had to exceed that of the Apriori Matrix algorithm. Since the dataset produces a large number of frequent multiple set calculations, FUP requires frequent scanning of the database, which increases the overhead, even more so than the Apriori Matrix algorithm. However, the FBCM algorithm needs to operate on and scan the database only one time; thus, it is faster. When the threshold is $s = 0.01$, the computing speed of the FBCM algorithm is much higher than that of the FUP algorithm. This experiment shows that the FBCM algorithm is more suitable for frequent itemset mining with smaller thresholds, that is, for frequent itemset mining with many computations.

Experiment 14: We know that the incremental database *db* and the minimum support threshold *s* correspond to different original transaction databases *DB*. Considering the time that is required, this experiment compares the FUP algorithm and FBCM algorithm in terms of their execution efficiency.

Figure 10 shows the performances on different datasets by the algorithms. The abscissa depicts the datasets and the threshold values. The incremental dataset include 100 samples from the original dataset. The detailed experimental dataset and the corresponding results of the experiment are presented in Table 3.

TABLE 3. Detailed experimental data and results.

Datasets	Data size	Support threshold	FUP run time (s)	FBCM run time (s)	Rate ratio
T10I4D100K	100000	0.02	14.75	5.0	2.95
mushroom	8124	0.4	16.33	0.75	21.77
T40I10D100K	100000	0.1	28.29	7.42	3.81
Groceries	9835	0.01	6.82	0.84	8.12

From the efficiency ratios shown in Table 3, we know that the running efficiency of the FBCM algorithm is higher than that of the FUP algorithm on our experimental datasets. On the T10I4D100K and T40I10D100K datasets, the mining speed is increased approximately 1.95 to 2.81 times, respectively. The improvement is more significant when considering the Groceries and Mushroom datasets.

Based on the analysis of the datasets and the results of the algorithms, we know that the Mushroom datasets have a stronger relationship between items. The datasets can produce high-frequency itemsets under a higher threshold setting; however, more iterations are needed, which means numerous scans of the transaction database by FUP, which increases the costs. Therefore, the performance of the FBCM algorithm is better.

From the above three experiments, we can draw the following conclusions. (1) The computational efficiency of the FBCM algorithm is significantly higher than that of the traditional algorithm. (2) The FBCM algorithm is suitable for large dataset cases involving a large number of computations. The greater the number of calculations is, the greater the efficiency of FBCM. (3) As the matrix algorithm, the recalculation efficiency of the FBCM algorithm based on the incremental dataset is much higher than that of the Apriori Matrix algorithm. Conclusion "1" and Conclusion 3 can be directly drawn from Figures 8, 9 and 10. However, Conclusion 2 requires the analysis as well as the results of algorithm operations. The smaller the threshold is, the greater the number of frequent items; additionally, the greater the number of calculations is, the higher the efficiency of the FBCM algorithm. The larger the datasets are, the greater the number of frequent items; additionally, the greater the number of calculations of the FBCM algorithm is, the higher the FBCM algorithm's efficiency.

VI. CONCLUSION

Considering the computing capacity and network bandwidth constraints of edge nodes, this paper proposes an incremental association rule mining algorithm based on matrix compression. This algorithm constructs a Boolean matrix by scanning a transaction database once, carries out incremental association rule mining using the matrix, and compresses the matrix during the mining process. Given these limitations, our solution uses two techniques to improve the efficiency on large databases. First, the operation of the Boolean matrix is used to replace the original transaction record scan. Second,

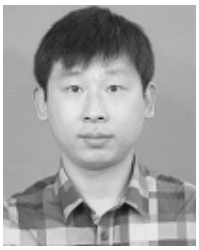
the compression of the matrix can decrease the search space of the solution. The experimental results show that the FBCM algorithm improves the efficiency of incremental association rule mining in edge computing.

REFERENCES

[1] M. Waldrop, "Big data: Wikiomics," *Nature*, vol. 455, no. 7209, pp. 22–25, 2008.
[2] J. Y. Liang, C. J. Feng, and P. Song, "A survey on correlation analysis of big data," *Chin. J. Comput.*, vol. 39, no. 1, pp. 1–18, 2016.
[3] Y.-S. Jeong, H. Hassan, and A. K. Sangaiah, "Machine learning on big data for future computing," *J. Supercomput.*, vol. 75, no. 6, pp. 2925–2929, 2019.
[4] H. Gao, W. Huang, and X. Yang, "Applying probabilistic model checking to path planning in an intelligent transportation system using mobility trajectories and their statistical data," *Intell. Automat. Soft Comput.*, vol. 25, no. 3, pp. 547–559, Jan. 2019.
[5] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, *Big Data: The Next Frontier for Innovation, Competition, and Productivity*. 2011.
[6] Y. Yin, J. Xia, Y. Li, Y. Xu, W. Xu, and L. Yu, "Group-wise itinerary planning in temporary mobile social network," *IEEE Access*, vol. 7, pp. 83682–83693, Jun. 2019.
[7] Y. Yin, L. Chen, Y. Xu, J. Wan, H. Zhang, and Z. Mai, "QoS prediction for service recommendation with deep feature learning in edge computing environment," *Mobile Netw. Appl.*, to be published.
[8] G. Xu, B. Guo, C. Su, X. Zheng, K. Liang, D. S. Wong, and H. Wang, "Am I eclipsed? A smart detector of eclipse attacks for Ethereum," *Comput. Secur.*, vol. 88, Jan. 2020, Art. no. 101604.
[9] L. Li, G. Xu, L. Jiao, X. Li, H. Wang, J. Hu, H. Xian, and W. Lian, "A secure random key distribution scheme against node replication attacks in industrial wireless sensor systems," *IEEE Trans Ind. Informat.*, to be published.
[10] X. Zhang, K.-K. R. Choo, and N. L. Beebe, "How do I share my IoT forensic experience with the broader community? An automated knowledge sharing IoT forensic platform," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6850–6861, Apr. 2019.
[11] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proc. ACM SIGMOD Rec.*, vol. 22, no. 2, 1993, pp. 207–216.
[12] W. Altaf, M. Shahbaz, and A. Guergachi, "Applications of association rule mining in health informatics: A survey," *Artif. Intell. Rev.*, vol. 47, no. 3, pp. 313–340, 2017.
[13] S. Gole and B. Tidke, "Frequent itemset mining for big data in social media using clustbigfim algorithm," in *Proc. Int. Conf. Pervasive Comput.*, 2015, pp. 1–6.
[14] K. Bartos, "Association rules in the study of consumer behaviour," in *Europejska przestrzeń komunikacji elektronicznej. T. 2* (Ekonomiczne Problemy Usług), no. 105. Szczecin, Poland: Zeszyty Naukowe Uniwersytetu Szczecińskiego, 2013, pp. 279–286.
[15] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques* (Series in Data Management Systems), 3rd ed. San Mateo, CA, USA: Morgan Kaufmann, 2011, pp. 83–124.
[16] M. Cheng, Y. Ling, and W. B. Wu, "Time series analysis for jamming attack detection in wireless networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2017, pp. 1–7.
[17] W. Yang, J. Yuan, W. Wu, J. Ma, and D. Z. Du, "Maximizing activity profit in social networks," *IEEE Trans. Comput. Social Syst.*, vol. 6, no. 1, pp. 117–126, Jan. 2019.
[18] Y. Shi, Z. Zhang, Y. Mo, and D.-Z. Du, "Approximation algorithm for minimum weight fault-tolerant virtual backbone in unit disk graphs," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 925–933, Sep. 2017.
[19] C. Y. Jia, "Association rule mining: A survey," *Comput. Sci.*, vol. 30, no. 4, pp. 147–151, 2003.
[20] S. Ziauddin, K. Kammal, and M. I. Z. Khan, "Research on association rule mining," *J. Acoust. Soc. Amer.*, vol. 117, no. 4, p. 2392, 2012.
[21] J. Kaur and N. Madan, "Association rule mining: A survey," *Int. J. Hybrid Inf. Technol.*, vol. 8, no. 7, pp. 239–242, 2015.
[22] D. W. Cheung, C. Y. Wong, J. Han, and V. T. Ng, "Maintenance of discovered association rules in large databases: An incremental updating technique," in *Proc. 12th Int. Conf. Data Eng.*, Feb. 1996, pp. 106–114.
[23] S. Bhanderi, S. Shah, and N. C. Chauhan, "Incremental mining of association rules: A survey," *Int. J. Comput. Sci. Inf. Technol.*, vol. 3, no. 3, pp. 4071–4074, 2013.
[24] B. Nath, D. K. Bhattacharyya, and A. Ghosh, "Incremental association rule mining: A survey," *Wires Data Mining Knowl. Discovery*, vol. 3, no. 3, pp. 157–169, 2013.
[25] B. Z. Zhang, K. Q. Jiang, and Y. Z. Zhang, "Survey on incremental association rule mining research," *J. Chin. Comput. Syst.*, vol. 37, no. 1, pp. 18–23, 2016.
[26] W. Cheung and O. R. Zaiane, "Incremental mining of frequent patterns without candidate generation or support constraint," in *Proc. 7th Int. Database Eng. Appl. Symp.*, 2003, pp. 111–116.
[27] C. K.-S. Leung, Q. I. Khan, and T. Hoque, "CanTree: A tree structure for efficient incremental mining of frequent patterns," in *Proc. 5th IEEE Int. Conf. Data Mining*, Nov. 2005, pp. 274–281.
[28] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2000, pp. 1–12.
[29] W. L. Cheung, S. D. Lee, and B. Kao, "A general incremental technique for maintaining discovered association rules," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 1997, pp. 185–194.
[30] J. Hipp, U. Güntzer, and G. Nakhaeizadeh, "Algorithms for association rule mining—A general survey and comparison," *SIGKDD Explor.*, vol. 2, no. 1, pp. 58–64, 2000.
[31] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases*, 1998, pp. 487–499.
[32] Y. Yuan and T. Huang, "A matrix algorithm for mining association rules," in *Advances in Intelligent Computing*, vol. 3644. Berlin, Germany: Springer, 2005, pp. 370–379.
[33] N. Khare, N. Adlakha, and K. R. Pardasani, "An algorithm for mining multidimensional association rules using Boolean matrix," in *Proc. Int. Conf. Recent Trends Inf., Telecommun. Comput.*, Mar. 2010, pp. 95–99.
[34] X. W. Luo and W. Wang, "Improved algorithms research for association rule based on matrix," in *Proc. Int. Conf. Intell. Comput. Cogn. Inform.*, Jun. 2010, pp. 415–419.
[35] H. Liu and B. Wang, "An association rule mining algorithm based on a Boolean matrix," *Data Sci. J.*, vol. 6, pp. 559–565, Sep. 2007.
[36] Y. Zhong and D. Liu, "An efficient association mining method via matrix compression," in *Proc. Int. Symp. Comput. Intell. Design*, Dec. 2017, pp. 188–192.
[37] T. Li and D. Luo, "A new improved apriori algorithm based on compression matrix," in *Proc. Int. Conf. Adv. Data Mining Appl.*, vol. 8933. Berlin, Germany: Springer-Verlag, 2014, pp. 1–15.
[38] S. Shu, "A new association rule mining algorithm based on compression matrix," in *Computer Engineering and Networking* (Lecture Notes in Electrical Engineering), vol. 277. Shanghai, China: Springer, 2014, pp. 281–289, doi: 10.1007/978-3-319-01766-2_33.
[39] X. Zhan, F. F.-H. Nah, and M. X. Cheng, "An assessment of users' cyber security risk tolerance in reward-based exchange," in *Proc. Int. Conf. HCI Bus., Government, Org.* Cham, Switzerland: Springer, 2018, pp. 431–441.
[40] J. Zhou, Z. Zhang, S. Tang, X. Huang, and D.-Z. Du, "Breaking the $O(\ln n)$ barrier: An enhanced approximation algorithm for fault-tolerant minimum weight connected dominating set," *INFORMS J. Comput.*, vol. 30, no. 2, pp. 225–235, 2018, doi: 10.1287/ijoc.2017.0775.
[41] A. Goli, E. B. Tirkolaee, B. Malmir, G.-B. Bian, and A. K. Sangaiah, "A multi-objective invasive weed optimization algorithm for robust aggregate production planning under uncertain seasonal demand," *Computing*, vol. 101, no. 6, pp. 499–529, Jun. 2019. [Online]. Available: https://doi.org/10.1007/s00607-018-00692-2
[42] C. Zhang and S. Zhang, *Association Rule Mining: Models and Algorithms*. Berlin, Germany: Springer-Verlag, 2002.

DONGDAI ZHOU received the B.Eng. and M.Sc. degrees from the Changchun University of Science and Technology, China, in 1992 and 1997, respectively, and the Ph.D. degree from Jilin University, China, in 2001. He is currently a Professor with the School of Information Science and Technology, Northeast Normal University, China. His research interests include software architecture and software code auto-generation, education data mining, and deep learning.

MENG OUYANG received the B.E. degree from the School of Software Engineering, Northeast Normal University, China, in 2016, and the M.S. degree from the School of Information Science and Technology, Northeast Normal University, China, in 2019. His research interests include big data and data mining.



ZHEJUN KUANG received the B.E. degree in computer science and information system from Massey University, New Zealand, in 2008, and the M.S. and Ph.D. degrees from the School of Computer Science and Technology, Jilin University, China, in 2011 and 2014, respectively. He is currently an Assistant Professor with the School of Computer Science and Technology, Changchun University, China. His research interests are in cyber-physical systems, the Internet of Things, and data mining.

ZHEN LI received the B.E. and M.S. degrees from the School of Software Engineering, Northeast Normal University, in 2012 and 2015, respectively, where he is currently pursuing the Ph.D. degree with the School of Information Science and Technology. His research interests include educational data mining and learning analytics.

JIN PENG ZHOU is currently pursuing the bachelor's degree in engineering science program machine intelligence option with the University of Toronto. He is currently a Machine Learning Researcher at Layer 6 AI, working on recommender system. Before this, he has worked on eye tracking on embedded devices at MIT and text generation using deep reinforcement learning at the Vector Institute. His research areas include big data mining, recommender systems, and computer vision.

XIAOCHUN CHENG (SM'04) received the B.Eng. degree in computer software engineering and the Ph.D. degree in computer science from Jilin University, in 1992 and 1996, respectively. He has been a Coordinator of the Computer Science EU Project, Middlesex University, since 2012. He is a member of the IEEE SMC Technical Committee on Enterprise Information Systems, the IEEE SMC Technical Committee on Computational Intelligence, the IEEE SMC Technical Committee on Cognitive Computing, the IEEE SMC Technical Committee on Intelligent Internet Systems, the IEEE Communications Society Communications and Information Security Technical Committee, BCS Information Security Specialist Group, BCS Cybercrime Forensics Specialist Group, and BCS Artificial Intelligence Specialist Group.