applied sciences

MDPI

Article

Incrementally Mining Column Constant Biclusters with FVSFP Tree

Jiaxuan Zhang, Xueyong Wang * and Jie Liu

School of Management Science, Qufu Normal University, Rizhao 276800, China; ZhangJX0207@163.com (J.Z.); liujieqfnu@163.com (J.L.)

* Correspondence: wangxy2015@qfnu.edu.cn

Abstract: Bicluster mining has been frequently studied in the data mining field. Because column constant biclusters (CCB) can be transformed to be discriminative rules, they have been widely applied in various fields. However, no research on incrementally mining CCB has been reported in the literature. In real situations, due to the limitation of computation resources (such as memory), it is impossible to mine biclusters from very large datasets. Therefore, in this study, we propose an incremental mining CCB method. CCB can be deemed as a special case of frequent pattern (FP). Currently the most frequently used method for incrementally mining frequent patterns is FP tree based method. In this study, we innovatively propose an incremental mining CCB method with modified FP tree data structure. The technical contributions lie in two aspects. The first aspect is that we propose a modified FP tree data structure, namely Feature Value Sorting Frequent Pattern (FVSFP) tree that can be easily maintained. The second aspect is that we innovatively design a method for mining CCB from FVSFP tree. To verify the performance of the proposed method, it is tested on several datasets. Experimental results demonstrated that the proposed method has good performance for incrementally handling a newly added dataset.

Keywords: feature value sorting frequent pattern tree; incremental mining; column constant bicluster

Citation: Zhang, J.; Wang, X.; Liu, J. Incrementally Mining Column Constant Biclusters with FVSFP Tree. *Appl. Sci.* **2023**, *13*, 6458. https://doi.org/10.3390/app13116458

Academic Editor: Antonio Fernández-Caballero

Received: 13 March 2023
Revised: 19 May 2023
Accepted: 22 May 2023
Published: 25 May 2023

1. Introduction

Clustering is an important part of data mining [1]. Clustering can group samples into different clusters. Traditional one-way clustering methods such as K-means [2] take all features into consideration when calculating the similarity between samples. In many cases samples are similar only under partial features. Biclustering can cluster from both the row (sample) dimension and column (feature) dimension, extracting local coherent patterns.

As shown in Figure 1, a bicluster can be categorized into constant bicluster, row constant bicluster, column constant bicluster, additive bicluster, multiplicative bicluster and coherent evolution bicluster [3]. Biclustering originates from gene expression data analysis [4], it has also been applied to other fields. In bicluster application, column constant biclusters (CCB) are most frequently used. In CCB, the values of each column are identical. As CCB can be transformed to be a horizontal discriminative vector through column averaging, to some extent, the horizontal discriminative vector has discriminating ability. CCB is expressed as follows: $CCB = \{(f_1, f_2, \ldots, f_i, \ldots, f_n) : m\}$ where f_i denotes the i-th column value of the CCB containing m rows and n columns.

In the literature, many studies based on CCB [5–11] have been proposed. In [5], detailed theoretical analysis of CCB based classification is given. In [6], CCB is used to mine breast cancer diagnosis rules, finally, with adaboost, the diagnosis rules can be combined to construct a strong diagnosis rule. In [7,9], CCB is applied to mine stock price fluctuating rules, finally, with KNN or fuzzy influence, an optimal solution can be obtained. In [8], CCB is used to extract motor imagery classification rules that can construct a fuzzy rule base, subsequently, with fuzzy inference, an excellent classifier can be built. In [10], a new

method for customer segmentation based on CCB is proposed. In [11], CCB is used to select features.

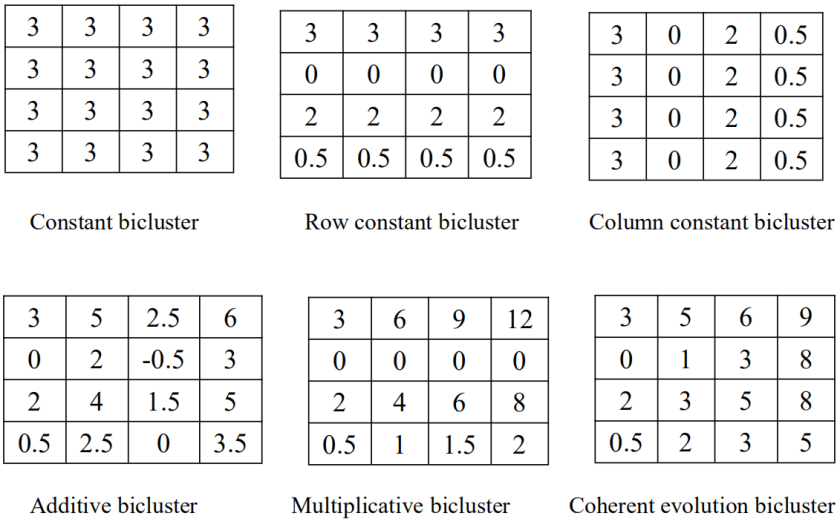


Figure 1. An example illustrating the six types of bicluster.

In a review of the studies in the literature, existing CCB-based studies process data in a batch way. How to mine CCB in an incremental way has not been investigated yet. In real situations, since the computation sources are limited and fixed, if the dataset is very big, running will fail because the required computation resource is bigger than the maximal available computation resource. If the dataset is divided into several small parts, the required computation resource for incrementally processing each part is smaller than processing the whole dataset, therefore, processing the whole dataset can be successful. Therefore, designing a novel method for incrementally mining CCB [12–16] is critically necessary. Incrementally mining CCB from an original dataset and an incremental dataset can be divided into the following two steps: (1) Mine original constant column biclusters CCB_o from original dataset D_o and save the intermediates I_o that are used to generate CCB_o. (2) Mine the whole column constant biclusters CCB_a from the new added dataset D_a and previously generated intermediates I_o, CCB_a is the same as the CCB directly mined from the whole datasets (the combination of $D_o \bigcup D_a$). Incrementally mining CCB should have the following properties: (1) With the coming of a newly added dataset, the intermediates that are used to generate the previous mined biclusters can be updated to improve the quality of previously mined CCB and new CCB can be mined. (2) The original datasets that have been processed before cannot be scanned/processed again in the incremental mining stage.

Nowadays, many biclustering methods have been proposed. Typical biclustering methods include the greedy search-based method [4], the evolutionary computation-based method [17], the exhaustive enumeration-based method [18], the statistical model-based method [19], etc. However, these methods are designed for mining all kinds of biclusters instead of only mining CCB and no incremental mining methods have been reported in the literature.

Frequent pattern [20] mining is a frequently used pattern [21–24]. In our opinion, CCB mining can be deemed as a special frequent pattern. Frequent patterns can be defined as the substructures that appear in a dataset with counts no less than predefined minimal number specified by the user [20]. Existing incremental frequent pattern mining methods can be categorized into two kinds. First is APRIORI-based incremental mining [25,26]. Such methods require appropriate parameter settings, consuming much time and space. Second is FP (frequent pattern) tree-based incremental mining [27]. The FP tree is a highly compressed data structure. Compared with APRIORI, FP tree is more widely used. The FP tree mining algorithm contains two phases, namely, constructing FP tree from dataset and deriving frequent patterns from FP tree. Constructing FP tree involves three steps.

The first step is scanning the whole dataset to find all items and their counts. The items whose counts are no less than the predefined minimal support are deemed to be frequent and will not be deleted. Then, the frequent items are sorted in descending order. Finally, the dataset is scanned again to construct the FP tree according to the descending order of large items. At the same time, a header table is built to record the frequent items. A header table is made of each large item's name, counts and a pointer to its first appearance node in the FP tree. When the FP tree has been constructed, the FP growth procedure is employed to mine all frequent patterns. Analyzing the whole frequent patterns mined by FP growth, it can be easily found that many frequent patterns are the subsets of others. If all the elements of frequent pattern A are included in frequent pattern B, then A is the subset of B. Each frequent pattern can be seen as a CCB. In the context of CCB based applications, the information contained in subset CCB is redundant, they should be deleted, only the CCB with the biggest volume is preserved. For example, from the left matrix in Table 1, 11 frequent patterns can be mined. Frequent patterns 2–11 are the subsets of the first frequent pattern. From the perspective of CCB, we just want to find the CCB having biggest volume, namely, the first frequent pattern: {(3, 0, 2, 0.5):4}.

Table 1. Illustration of subset frequent pattern mining.

				No.	FP	No.	FP
3	0	2	0.5	1	(3, 0, 2, 0.5):4	6	(3, 0):4
3	0	2	0.5	2	(0, 2, 0.5):4	7	(3, 2):4
3	0	2	0.5	3	(3, 2, 0.5):4	8	(3, 0.5):4
3	0	2	0.5	4	(3, 0, 0.5):4	9	(0, 2):4
		Matrix		5	(3, 0, 2):4	10	(0, 0.5):4
					11	(2, 0.5):4	

FP mining is a computationally expensive task. With the coming of a new dataset, if beginning from the scratch, finding FP from the whole dataset may waste a lot of time. In existing FP tree-based incremental mining methods, the key issue in updating the FP tree is the node adjustment of the FP tree [28–30]. Due to the addition of new samples, some frequent nodes in original FP tree may become unfrequent nodes, these nodes should be deleted in the new updated FP tree. Some unfrequent nodes in original FP tree may become frequent nodes, these nodes are supposed to be added to the new updated FP tree. Additionally, because the nodes in original FP tree are positioned in descending order, the positions of the nodes that are frequent in both original and new updated FP tree need to be adjusted. The nodes that are infrequent in both the original and the new updated FP tree are ignored [29].

Updating the FP tree consists of recalculating the counts of each item, recalculating the minimal supports threshold, renewing the head table and adjusting the node positions in the FP tree. The disadvantage of such updation is that it will take much time to finish the updation. Heavy computation is mainly caused by many updations. To avoid these updations, we proposed a novel tree structure, namely, the Feature Value Sorting Frequent Pattern (FVSFP) tree. The difference between FP tree and FVSFP tree lies in that infrequent nodes are preserved in FVSFP tree, the minimal supports are not taken into consideration when constructing the FVSFP tree. The infrequent nodes are deleted in the CCB mining stage instead of the tree construction stage. The elements in the head table and FVSFP tree are sorted according to feature values instead of feature values' counts/frequency. In this way, the tree structure can be more easily maintained when inserting a newly added dataset to the tree. The updation of the FVSFP tree is nearly the same as its initial construction process, reducing the consumed time and space. Since the proposed method aims to incrementally mine a bicluster, it is named as IMB. The technical contribution of the proposed method lie in the following:

1. We propose a modified FP tree data structure, namely, a Feature Value Sorting Frequent Pattern (FVSFP) tree, which can be easily maintained;
2. We innovatively design a method for mining CCB from FVSFP tree. The mining method is greatly different from the standard frequent pattern mining method [31].

The following parts are organized as follows. Section 2 presents a detailed description of the proposed method. Section 3 presents the experiment. The conclusion and discussion are presented in Section 4.

2. Method

In this section, each step of the proposed method is described in detail. Firstly, feature transformation used to ensure that the feature value in each sample is unique is presented. Secondly, mining CCB from the initial dataset is presented. Finally, incremental mining of CCB is introduced in detail. To illustrate vividly, an example for illustrating each step is given.

2.1. Preprocessing

To construct FP tree, the feature values in the same sample must be unique. In many cases, sample feature usually contain identical values, therefore, feature transformation should be performed to ensure feature uniqueness. As each sample feature value constitutes one node, each sample constructs one branch. Identical feature value in sample will lead to two nodes in one branch having identical names, causing confusion and mistakes. The feature transformation schema is as follows:

1. Calculate the number of each feature's possible values, denote $n_i (1 \leq i \leq k)$ as the ith feature value's size. Finally, a vector $[n_1, n_2, \ldots, n_k]$ containing the number of each feature's possible values can be obtained;
2. Transform feature values column-by-column in two steps. The first is to uniquify and sort the n_i feature values of the ith column in ascending order to obtain original feature value vector $[f_1, f_2, \ldots, f_{n_i}]$. The second step is to transform the ordered unique feature to new feature f_p=$\sum_{j=1}^{i-1} n_j + p - 1$ where $1 \leq p \leq n_i$. After transformation, the ith feature values fall in the range of $[\sum_{j=1}^{i-1} n_j - 1, \sum_{j=1}^{i} n_j - 1]$

An example for illustrating feature transformation is shown in Table 2. In Table 2, the top subtable is the original data matrix, in the second and fourth sample, there are identical feature values. The middle subtable provides each feature value's size and feature value range after transformation. The below subtable is the data matrix after transformation, each sample (row) has no duplicated values. After transformation, the sample feature value's uniqueness is ensured.

2.2. Initial CCB Mining

2.2.1. Construction of a Header Table

A head table should be constructed before constructing the FVSFP tree. The construction process is as follows: (1) Construct a empty head table. (2) Scan the transformed original dataset sample by sample. If a feature value exists in the head table, ignore it, otherwise, insert the feature value into the head table. Subsequently, sort the feature values in header table in feature value's ascending order instead of feature value counts' descending order that is commonly used in the literature. In the header table, all the feature values are stored, no feature value is deleted.

To vividly illustrate the construction of head table, an example is given in Figure 2. In the original dataset there are 5 samples. Scan the 5 samples sample by sample. After scanning the first sample and sorting the feature values in the first sample, a head table containing 4 feature values "0, 1, 3, 5" is obtained. After scanning the second sample, because feature values "2, 4, 6" are not contained in the head table, "2, 4, 6" should be inserted into the head table, subsequently, a head table containing 7 feature values "0, 1, 3, 5, 2, 4, 6" can be obtained. After the third sample, no new feature value is inserted to

the head table. Finally, a head table containing 7 feature values "0, 1, 3, 5, 2, 4, 6" can be obtained.

Table 2. An example illustrating the feature transformation process.

Sample No.	**Original Feature**			
1	0	0	0	0
2	0	1	1	1
3	0	0	0	0
4	0	0	0	1
Original feature value vector	[0]	[0 1]	[0 1]	[0 1]
Feature value size	1	2	2	2
New feature value range	[0]	[1 2]	[3 4]	[5 6]
Transfromation (mapping)	0->0	0->1 1->2	0->3 1->4	0->5 1->6
Sample No.	**Transformed Feature**			
1	0	1	3	5
2	0	2	4	6
3	0	1	3	5
4	0	1	3	6

2.2.2. Initial FVSFP Tree Construction

Constructing FVSFP tree is similar to constructing FP tree. The key is iteratively inserting sample to the tree. The tree is made of nodes. The node is defined as follows:

```
Struct FVSFPNode{
Name;
Count;
Struct FVSFPNode* Parent;
Struct FVSFPNode* Children[];
}
```

Name: the name of the node.

Count: the appearance number of the node.

Parent: a pointer to the parent node.

Children: one or several pointers to the whole children nodes.

In the FVSFP tree, there are three kinds of nodes, namely, Root node, leaf node and middle node. The Root node is in the topmost level, all domains except for Children are NULL. The leaf node is in the lowest level, has no children. Middle node is the node between the leaf node and the Root node. One branch of the tree is defined as the whole nodes from Root node to leaf node. One sample can form one branch. The construction of an original FVSFP tree can be summarized in the following two steps:

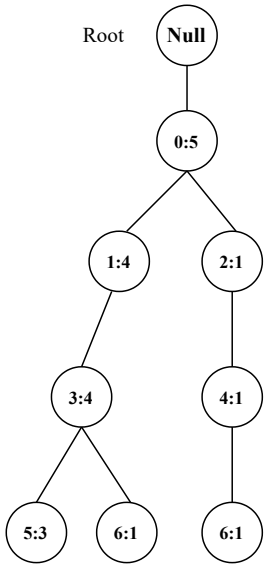
* Construct the Root node;
* Insert the samples of the original dataset sample-by-sample with Algorithm 1.

Sample no.	Feature values
1	0 1 3 5
2	0 2 4 6
3	0 1 3 5
4	0 1 3 6
5	0 1 3 5

Original datasets

No.	Feature value
1	0
2	1
3	2
4	3
5	4
6	5
7	6

Original header table



Original tree

Branch no.	Node and counts
1	(0:5, 1:4, 3:4, 5:3)
2	(0:5, 1:4, 3:4, 6:1)
3	(0:5, 2:1, 4:1, 6:1)

1	(0,1,3):4
2	(0,1,3):4
3	Null

Original biclusters: (0,1,3):4

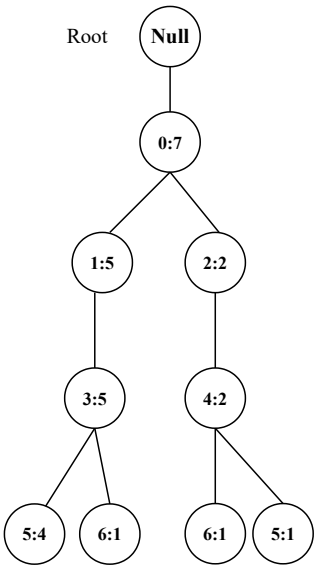
Mine biclusters from original tree

Sample no.	Feature values
6	0 1 3 5
7	0 2 4 5

New added datasets

No.	Feature value
1	0
2	1
3	2
4	3
5	4
6	5
7	6

Updated header table



Updated tree

Branch no.	Node and counts
1	(0:7, 1:5, 3:5, 5:4)
2	(0:7, 1:5, 3:5, 6:1)
3	(0:7, 2:2, 4:2, 6:1)
4	(0:7, 2:2, 4:2, 5:1)
1	(0, 1, 3, 5):4
2	(0, 1, 3):5
3	Null
4	Null

Updated biclusters: (0 1 3 5):4
(0 1 3):5

Mine biclusters from updated tree

Figure 2. Illustration of incrementally mining CCB.

Algorithm 1 Insert sample to FVSFP tree.

Require: Initial FVSFP tree, T_o; Sample, S; Feature number in S, L.
Ensure: Updated FVSFP tree, T_o

```
CurrentNode←Root of T_o.
CNodes←CurrentNode's children nodes.
n←CNodes' size.
for i=1:L do
    if CNodes[j]'s name (1 ≤ j ≤ n) is the same as S(i) then
        CurrentNode ← CNodes[j]
        CNodes←CurrentNode's children
        n←CNodes' size.
        CurrentNode's Count←CurrentNode's Count+ 1.
    else
        Add a new node NodeN to T_o.
        Set NodeN's Name as S(i), Count as 1.
        Set NodeN's Parent as CurrentNode.
        Set NodeN's Children as NULL.
        CurrentNode← NodeN
        CNodes←CurrentNode's children
        n←CNodes' size.
    end if
end for
return T_o
```

One example illustrating adding a sample to the FVSFP tree is shown in Figure 3. Figure 3 shows the process of inserting the second sample of the original datasets in Figure 2 to the FVSFP tree with Algorithm 1. The adding process is finding common nodes, increasing common nodes' count with 1 if the sample's feature value is identical and adding new nodes otherwise.

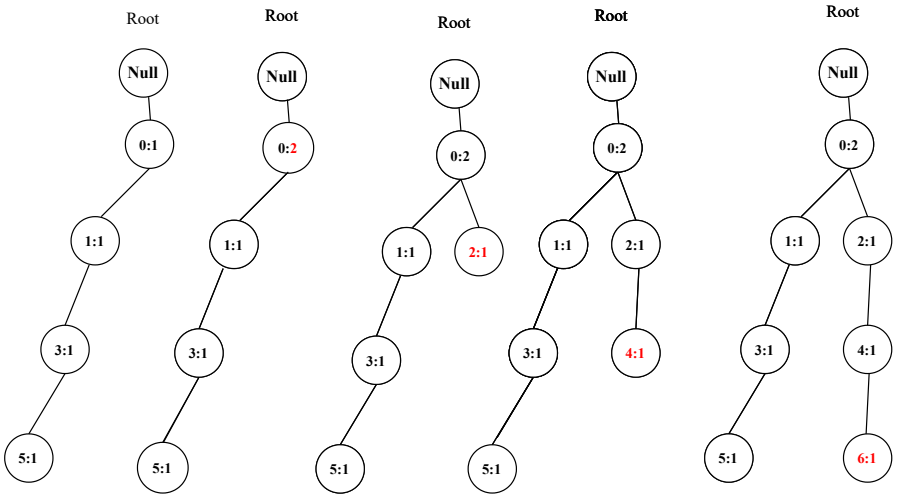


Figure 3. Illustration of adding a sample to the FVSFP tree. From left to right, the first tree is the tree after the first sample (0 1 3 5) is inserted to the tree. The second tree is the updated tree after the first feature value '0' of the second sample (0 2 4 6) is inserted to the tree, because feature value '0' is the same as the name of node "0", the number of node "0" increases from 1 to 2. The third tree corresponds to the updated tree after feature value '2' is inserted to the tree, because in the second level no node's name is the same as second feature value '2', thus, a new node "2" is created in the second level. The fourth and fifth trees are created in the same way as the third tree.

2.2.3. Initial CCB Mining

Having built the initial FVSFP tree, the following step is mining CCB from the FVSFP tree. As mentioned above, one of the differences between the FVSFP tree and the FP tree is that the unfrequent nodes are deleted in the mining process instead of the FVSFP tree construction process. In the context of CCB-based applications, the main goal is to find the CCBs whose volume is as big as possible. Therefore, a novel bicluster mining method is proposed by mining one bicluster with the largest volume from each branch of the FVSFP tree. Sometimes, two CCBs may have identical volume; compared with feature number, row number is more important [5], therefore, the CCB with larger supports (row number) is outputted. Any CCBs containing less than NT rows or 3 columns are deemed to be meaningless [7] and should be deleted.

$$NT = |D| * msr \tag{1}$$

where $|D|$denotes the number of samples in the datasets D, msr is the minimal support rate.

The CCB mining strategy is as follows:

* Find all leaf nodes;
* Find all branches by iteratively combining the whole nodes from the leaf node to the Root node;
* For each branch, delete the infrequent nodes whose count is less than threshold NT. Then, find the whole candidate CCBs from each branch with the following steps. (1): Unique the counts of the nodes in the branch to obtain the unique counts array $[mcc_1, \ldots, mcc_i, \ldots, mcc_n]$ $(1 \leq i \leq n)$ where n is the length of the array, mcc means the maximal common counts among the whole nodes in the CCB. For each mcc, calculate the whole nodes whose count is greater than or equal to the mcc, then the volume of the CCB can be obtained by Equation (2). Finally, the bicluster with the biggest volume is selected.
  Take the first branch (0:5 1:4 3:4 5:3) of the FVSFP tree generated by the original dataset in Figure 2, for instance, to illustrate mining CCB. Suppose the minimal support rate is 0.4, therefore, the minimal supports is 0.4 × 5 = 2, the nodes whose count is less than 2 are unfrequent nodes, no node is unfrequent node, all the four nodes should be preserved. Then, the remaining effective branch becomes (0:5 1:4 3:4 5:3). Unique the counts [3 4 5], because the number of the nodes whose count is no less than 5 is less than 3, therefore remaining valid counts are [3 4]. Two candidate CCBs (shown in Table 3) can be generated, both CCBs have identical volume, but counts are more important than the node number, finally, the optimal CCB (0,1,3):4 is outputted.
* Delete the identical and subset CCBs. Because the optimal CCBs generated from different branches may be identical, identical CCBs should be deleted. Additionally, some CCBs may be the subsets of other CCBs, the subset CCBs also should be deleted. The maximal common count deleting subset pattern can guarantee that the obtained CCBs are inclusion-maximal.

$$V_i = N_i * mcc_i \tag{2}$$

where mcc denotes the least common counts of the nodes, N_i denotes the number of nodes whose count is no less than mcc_i.

Table 3. Selecting the optimal CCB from the whole candidate patterns.

CCB No.	Nodes	Least Common Counts (*lcc*)	Volume
1	(0, 1, 3, 5)	3	3 × 4 = 12
2	(0, 1, 3)	4	4 × 3 = 12

As shown in the top part of Figure 2, D_0 is composed of 5 samples, the feature values in sample are non-repetitive. The feature values in the original header table are sorted according to feature values' ascending order. With Algorithm 1, the original FVSFP tree can be constructed. The final step is to mine CCB from the branches of the tree. There are 3 branches, each of which is listed in the top subtable of the rightmost table. The middle subtable shows the optimal CCB mined from each branch. The subtable below is the final outputted CCB after deleting the identical and subset CCB.

2.3. Incremental CCB Mining

2.3.1. Updation of Initial FVSFP Tree

With the arrival of the newly added dataset, update the initial FVSFP tree through scanning only the updated dataset instead of scanning the entire datasets that is composed of the original dataset and the newly added dataset. The updation process involves inserting each sample to the initial FVSFP tree with Algorithm 1, the same method as constructing the initial FVSFP tree.

2.3.2. Remining of CCB

The next step is to mine biclusters from the updated FVSFP tree. The mining process is the same as the initial CCB mining. As shown in the lower part of Figure 2, the incremental (newly added) dataset consisting of 2 samples is inputted to the update initial FVSFP tree and remine CCB. There is no reordering in the header table. The tree is updated by inserting each sample in the incremental dataset to the initial FVSFP tree. Finally, two CCBs are outputted. One CCB is the refinement of previously generated CCB, the other is newly generated.

3. Experiments

3.1. Experimental Settings

To verify the performance of the proposed incrementally mining CCB method, it is tested on 5 datasets. The experiment scheme is similar to the scheme in [32]. Firstly, the whole dataset is equally divided into disjoint two subsets, namely, the original dataset and incremental dataset.

Subsequently, the original dataset is inputted to the algorithm to produce the initial FVSFP tree and initial CCBs. Then, the following experiment is divided into two parts. The first part is inputting the incremental dataset and initial FVSFP tree to the algorithm to obtain the updated FVSFP tree and CCBs. The second part is inputting the whole dataset that consists of original and incremental datasets to the algorithm without the initial FVSFP tree. The 5 testing datasets can be divided into three parts. The first is the FIM dataset http://fimi.uantwerpen.be/data/ (accessed on 11 October 2022), FIM are frequently used in frequent pattern mining experiments. The second is the UCI dataset https://archive.ics.uci.edu/ml/index.php (accessed on 11 October 2022), UCI is commonly applied in classification and clustering experiments in the literature. The third is the BIRADS dataset [6] which is collected by our team. Detailed information on the 5 datasets is displayed in Table 4.

The methods are implemented in the C++ programming language in the Microsoft Visual Studio 2019 platform. The proposed method was run on a laptop. The configuration of the laptop is shown in Figure 4. To our knowledge, no previous studies about incrementally mining CCB have been reported, this study is the first study about incrementally mining CCB, thus, comparison experiments cannot be conducted.

Table 4. Description of 5 datasets.

Dataset	Division	Sample Counts	Size (KB)	Unique Feature?
chess	Original	1598	179	Yes
	Incremental	1598	180	
	Whole	3196	359	
mushroom	Original	4062	308	Yes
	Incremental	4062	312	
	Whole	8124	620	
semeion	Original	797	793	No
	Incremental	796	793	
	Whole	1593	1586	
Spambase	Original	2301	343	No
	Incremental	2300	343	
	Whole	4601	686	
BIRADS	Original	531	30	No
	Incremental	531	31	
	Whole	1062	61	

CPU	Intel(R) i5-8250U, 1.6 GHz, (4CPUs)
Memory	8GB
Disk	1TB
Operating System	Windows 10

Figure 4. Laptop configuration.

Experiment runtime and maximal memory utility under different minimal support rates are two important indicators [33] for evaluating performance. In this study, minimal support rates are set as 0.002, 0.004, 0.006, 0.008, and 0.01, respectively. Because the memory utility when running the algorithm is dynamic, only the maximal memory utility is reported. Comparatively speaking, the memory indicator is more important than the runtime indicator. When running IMB on each dataset, the mean and standard deviation of maximal memory utility and runtime of five times in incremental way and batch way are reported.

3.2. Experimental Results

3.2.1. Ablation Study

The proposed method contains two parts, namely, FVSFP tree and CCB mining. Therefore, to investigate the effectiveness of the two parts, an ablation study is conducted. As shown in Table 5, four combinations can be obtained. For each combination, "×" means that the corresponding part is not included. If the FVSFP tree is not contained, a standard FP tree [31] is used. If CCB mining is not contained, standard FP pattern mining [31] is used.

Ablation study result on chess dataset is shown in Table 5. We can see that the proposed FVSFP tree outperforms standard FP tree, the proposed CCB mining is better than standard FP pattern mining, demonstrating the effectiveness of the proposed two parts.

Table 5. Ablation study about the components. (bold means better result)

Components	Different Combinations of Components			
FVSFP tree	×	×	√	√
CCB mining	×	√	×	√
Runtime (s) (mean ± std)	39.39 ± 0.81	25.11 ± 0.93	28.72 ± 1.07	**15.76 ± 0.88**
Memory (MB) (mean ± std)	120.11 ± 1.25	79.25 ± 0.87	96.72 ± 0.04	**56.62 ± 0.68**

3.2.2. Testing on Chess Dataset

The experiment results on the chess dataset are displayed in Table 6. The first column denotes the minimal support rate. The second column represents two ways, "Incremental" means running in an incremental way, "Batch" means running in a batch way when processing the incremental dataset. Black bold indicates a better result. For the two performance indicators, the "Incremental" method achieves better performance than the "Batch" method in all cases.

Furthermore, with the decrease in minimal support rate, the runtime increases and the maximal memory utility also increases. This can be explained by the fact that in small minimal support rate cases, fewer nodes are deleted, more nodes are preserved, it will take a longer time and bigger memory to run. This can be explained by two advantages of the proposed method. The first is that the nodes in FVSFP tree are arranged with regard to feature value instead of feature value's counts. The second is that the infrequent nodes are preserved in the FVSFP tree construction process, they are deleted in the later mining process.

Table 6. Experimental results on the chess dataset. (bold means better result)

Minimal Support Rate	Methods	Runtime (s) (Mean ± std)	Memory (MB) (Mean ± std)
0.01	Incremental	**15.76 ± 0.88**	**56.62 ± 0.68**
	Batch	28.72 ± 1.07	96.72 ± 0.04
0.008	Incremental	**15.96 ± 0.22**	**57.22 ± 0.13**
	Batch	28.60 ± 0.83	97.06 ± 0.11
0.006	Incremental	**16.41 ± 1.50**	**57.54 ± 0.08**
	Batch	28.94 ± 1.05	97.26 ± 0.11
0.004	Incremental	**16.98 ± 0.32**	**58.4 ± 0.14**
	Batch	29.71 ± 0.35	98.04 ± 0.11
0.002	Incremental	**23.15 ± 1.17**	**59.92 ± 0.24**
	Batch	35.06 ± 0.22	99.38 ± 0.99

3.2.3. Testing on the Mushroom Dataset

Table 7 shows the experimental results on the mushroom dataset. Mining CCB in an incremental way costs less in terms of runtime and memory than mining in a batch way on all minimal support rates.

Table 7. Experimental results on the mushroom dataset. (bold means better result)

Minimal Support Rate	Methods	Runtime (s) (Mean ± std)	Memory (MB) (Mean ± std)
0.01	Incremental	**117.56 ± 4.99**	**89.22 ± 0.08**
	Batch	150.6 ± 1.51	144.06 ± 1.36
0.008	Incremental	**114.26 ± 5.16**	**97.6 ± 6.17**
	Batch	159 ± 2.34	145.44 ± 2.80
0.006	Incremental	**119.43 ± 8.52**	**79.4 ± 0.73**
	Batch	176.2 ± 10.80	147.76 ± 3.15
0.004	Incremental	**122.09 ± 9.43**	**80.18 ± 0.38**
	Batch	171.8 ± 9.49	147.28 ± 2.43
0.002	Incremental	**115.65 ± 6.90**	**88.68 ± 4.74**
	Batch	171.4 ± 5.77	149.44 ± 4.80

3.2.4. Testing on the Spambase Dataset

Test results on the Spambase dataset are reported in Table 8. The incremental method greatly outperforms the batch method on all minimal support rates.

Table 8. Experiment results on the Spambase dataset. (bold means better result)

Minimal Support Rate	Methods	Runtime (s) (Mean ± std)	Memory (MB) (Mean ± std)
0.01	Incremental	**26.16 ± 0.98**	**83.12 ± 0.78**
	Batch	48.22 ± 109	177.17 ± 0.52
0.008	Incremental	**26.99 ± 0.42**	**83.67 ± 0.23**
	Batch	48.90 ± 0.89	177.86 ± 0.41
0.006	Incremental	**27.39 ± 1.60**	**84.14 ± 0.11**
	Batch	49.24 ± 1.45	178.66 ± 0.41
0.004	Incremental	**27.84 ± 0.12**	**84.64 ± 0.34**
	Batch	49.91 ± 0.25	178.95 ± 0.21
0.002	Incremental	**38.05 ± 1.02**	**87.12 ± 0.26**
	Batch	60.86 ± 0.32	180.48 ± 0.59

3.2.5. Testing on the Semeion Dataset

Experiment results for the incremental method and batch method on the semeion dataset are shown in Table 9. The incremental method costs much less in terms of runtime and memory than the batch method on all minimal support rates.

Table 9. Experiment results on semeion dataset. (bold means better result)

Minimal Support Rate	Methods	Runtime (s) (Mean ± std)	Memory (MB) (Mean ± std)
0.01	Incremental	**37.76 ± 0.48**	**236.62 ± 3.75**
	Batch	67.8 ± 1.48	367.48 ± 0.25
0.008	Incremental	**37.77 ± 0.48**	**239.36 ± 0.32**
	Batch	67.2 ± 0.44	367.62 ± 0.08
0.006	Incremental	**39.16 ± 0.46**	**238.86 ± 1.19**
	Batch	68.6 ± 2.07	369.26 ± 2.54
0.004	Incremental	**40.91 ± 1.46**	**240.8 ± 4.40**
	Batch	70.4 ± 2.07	368.84 ± 0.11
0.002	Incremental	**60.92 ± 3.52**	**244.08 ± 3.30**
	Batch	90 ± 7.84	371.38 ± 0.04

3.2.6. Testing on the BIRADS Dataset

Comparison results of two methods on the BIRADS dataset are reported in Table 10. Incrementally mining CCB costs less runtime and less memory than mining using the batch method.

Table 10. Experimental results on the BIRADS dataset. (bold means better result)

Minimal Support Rate	Methods	Runtime (s) (Mean ± std)	Memory (MB) (Mean ± std)
0.01	Incremental	**2.03 ± 0.12**	**20.54 ± 0.13**
	Batch	3.19 ± 0.46	26.88 ± 0.14
0.008	Incremental	**2.04 ± 0.02**	**21.52 ± 0.14**
	Batch	3.27 ± 0.18	26.98 ± 0.08
0.006	Incremental	**2.31 ± 0.04**	**19.64 ± 0.23**
	Batch	3.65 ± 0.262	27.14 ± 0.11
0.004	Incremental	**3.20 ± 0.12**	**21.08 ± 0.13**
	Batch	4.20 ± 0.25	27.58 ± 0.04
0.002	Incremental	**5.40 ± 0.20**	**20.46 ± 0.13**
	Batch	7.62 ± 0.23	28.62 ± 0.08

3.2.7. Testing on the WebDocs Dataset

In addition to the five small datasets, one big dataset WebDocs http://fimi.uantwerpen.be/data/ (accessed on 11 October 2022) is tested. The size of T10I4D100K is 1.4 GB. It contains 1,692,082 samples. Running in a batch way fails because the required memory for the batch method is bigger than the maximal available memory. For the incremental method, WebDocs is divided into 20 equal subsets (71 MB), the required maximal memory is about 1.2 GB. The incremental method succeeds in running.

The final FVSFP tree and CCBs produced in both "Incremental" and "Batch" ways are identical. The average saved runtime and memory on five datasets is shown in Table 11. It can be found that the bigger the dataset size is, the longer time and more memory can be saved. If the dataset size is TB-level, the advantage of the proposed method can be more apparent.

Table 11. Saved time and memory on different datasets.

Dataset	Size (KB)	Average Saved Runtime (s)	Average Saved Memory (MB)
BIRADS	61	1.39	5.79
chess	359	12.55	44.75
mushroom	620	48.8	59.78
Spambase	686	22.3	54.2
semeion	1586	30.5	130.97

3.2.8. Comparison with State-of-the-Art Methods

To comprehensively investigate the performance of the proposed IMB method, it is compared with three state-of-the-art methods, FPMSIM [16], FIUFP [34] and TSTP [35]. FPMSIM and FIUFP improved the standard FP-growth [31] to update more effectively. TSTP is an evolutionary algorithm-based biclustering method that is designed to mine all kinds of biclusters. Since this study aims to only mine column constant biclusters, the fitness function of TSTP is modified as the variance sum of each column in a biclsuter. The comparison result of the four methods on five datasets when the minimal support rate is 0.01 is shown in Figure 5. We can see that the proposed IMB method costs much less runtime and memory than the three comparison methods, demonstrating its superiority.

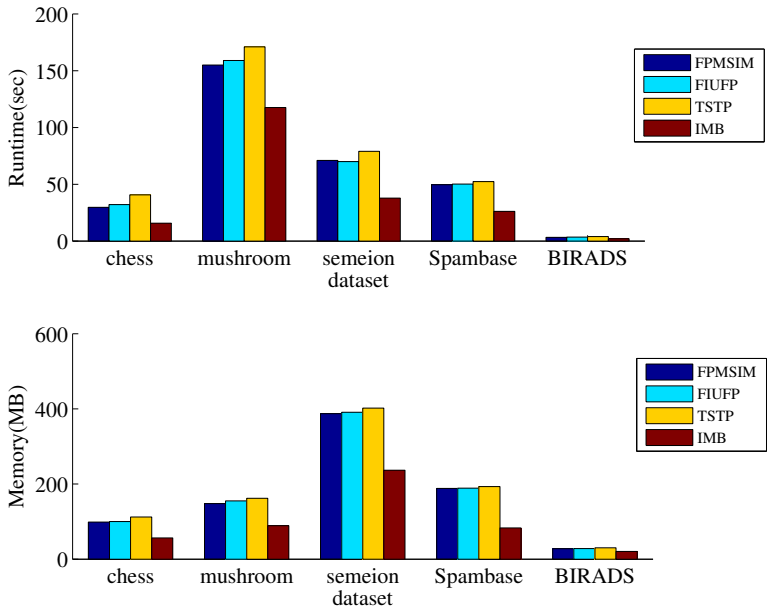


Figure 5. Comparison with state-of-the-art methods.

Analyzing the found CCBs, it can be found that for the mined CCBs of all methods (IMB, batch and the three state-of-the-art comparison methods), the number of mined CCBs of different methods are equal. Each CCB has identical elements. The difference of different methods lies only in the time and memory used to find the column constant bicluster.

4. Conclusions and Future Work

In this paper, we notice that nowadays the issue of incrementally mining column constant bicluster has not been investigated yet. We propose a novel incrementally mining column constant bicluster method based a modified FP tree named FVSFP. The technical contribution of FVSFP tree lies in two parts. The first is that the nodes in the FVSFP tree are arranged according to feature value instead of feature value's counts. The second is

that the infrequent nodes are preserved in the FVSFP tree construction process, they are deleted in the later mining process. Therefore, the FVSFP tree structure can be very easily maintained. Experiment results illustrate that the proposed model is capable of efficiently incrementally mining column constant biclusters, saving runtime and memory compared with mining in a batch way.

In the future, the following directions will be investigated: (1) In this study, only constant column bicluster is investigated. In real cases, due to the widespread noise, column nearly constant bicluster may be more frequently seen. In the future, the proposed FVSFP should be modified to mine column nearly constant bicluster. (2) Biclusters can be categorized as a constant bicluster, row constant bicluster, column constant bicluster, additive bicluster and multiplicative bicluster. In this study, only the column constant bicluster is studied. Incrementally mining other kinds of bicluster can be investigated in the future.

Author Contributions: Methodology, J.Z.; Writing and original draft preparation, J.Z. and X.W.; Writing, review and editing, J.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the NSFC under Grant No. 12071250.

Data Availability Statement: Data sharing not applicable No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: The author is very grateful to the referees for their careful reading and valuable suggestions, and to the teachers from the School of Management Science of Qufu Normal University for their guidance and help.

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

References

1. Samir, R.; El-Hennawy, H.; Elbadawy, H. Cluster-Based Multi-User Multi-Server Caching Mechanism in Beyond 5G/6G MEC. *Sensors* **2023**, *23*, 996. [CrossRef] [PubMed]
2. Li, M.; Wang, H.; Long, H.; Xiang, J.; Wang, B.; Xu, J.; Yang, J. Community Detection and Visualization in Complex Network by the Density-Canopy-Kmeans Algorithm and MDS Embedding. *IEEE Access* **2019**, *7*, 120616–120625. [CrossRef]
3. Huang, Q.; Tao, D.; Li, X.; Liew, A. Parallelized Evolutionary Learning for Detection of Biclusters in Gene Expression Data. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **2012**, *9*, 560–570. [CrossRef] [PubMed]
4. Cheng, Y.; Church, G.M. Biclustering of expression data. In Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, San Diego, CA, USA, 19–23 August 2000; pp. 799–808.
5. Cheng, H. *Towards Accurate and Efficient Classification: A Discriminative and Frequent Pattern-Based Approach*; Technical Report; University of Illinois: Urbana, IL, USA, 2008.
6. Huang, Q.; Chen, Y.; Liu, L.; Tao, D.; Li, X. On Combining Biclustering Mining and AdaBoost for Breast Tumor Classification. *IEEE Trans. Knowl. Data Eng.* **2020**, *32*, 728–738. [CrossRef]
7. Huang, Q.; Yang, J.; Feng, X.; Liew, A.W.; Li, X. Automated Trading Point Forecasting Based on Bicluster Mining and Fuzzy Inference. *IEEE Trans. Fuzzy Syst.* **2020**, *28*, 259–272. [CrossRef]
8. Sun, J. Motor Imagery EEG Classification with Biclustering Based Fuzzy Inference. *J. Med. Imaging Health Inform.* **2020**, *10*, 1486–1493. [CrossRef]
9. Huang, Q.; Wang, T.; Tao, D.; Li, X. Biclustering Learning of Trading Rules. *IEEE Trans. Cybern.* **2015**, *45*, 2287–2298. [CrossRef]
10. Xue, Y.; Li, T.; Chen, J.; Zhao, H.; Zhang, H. A New Customer Segmentation Framework Based on Biclustering Analysis. *J. Softw.* **2014**, *9*, 1359–1366.
11. Huang, Q.; Jin, L.; Tao, D. An unsupervised feature ranking scheme by discovering biclusters. In Proceedings of the 2009 IEEE International Conference on Systems, Man and Cybernetics, San Antonio, TX, USA, 11–14 October 2009; pp. 4970–4975. [CrossRef]
12. Saini, R.; Mussbacher, G.; Guo, J.L.; Kienzle, J. Machine learning-based incremental learning in interactive domain modelling. In Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems, Montreal, QC, Canada, 16–21 October 2022; pp. 176–186.
13. Ditzler, G.; Polikar, R. Incremental Learning of Concept Drift from Streaming Imbalanced Data. *IEEE Trans. Knowl. Data Eng.* **2013**, *25*, 2283–2301. [CrossRef]
14. Lange, S.; Zilles, S. Formal models of incremental learning and their analysis. In Proceedings of the International Joint Conference on Neural Networks, Portland, OR, USA, 20–24 July 2003; Volume 4, pp. 2691–2696. . [CrossRef]

15. Liu, X.; Zheng, L.; Zhang, W.; Zhou, J.; Cao, S.; Yu, S. An evolutive frequent pattern tree-based incremental knowledge discovery algorithm. *ACM Trans. Manag. Inf. Syst. (TMIS)* **2022**, *13*, 1–20. [CrossRef]
16. Xun, Y.; Cui, X.; Zhang, J.; Yin, Q. Incremental frequent itemsets mining based on frequent pattern tree and multi-scale. *Expert Syst. Appl.* **2021**, *163*, 113805. [CrossRef]
17. Huang, Q.; Huang, X.; Kong, Z.; Li, X.; Tao, D. Bi-Phase Evolutionary Searching for Biclusters in Gene Expression Data. *IEEE Trans. Evol. Comput.* **2019**, *23*, 803–814. [CrossRef]
18. Amos, T.; Roded, S.; Ron, S. Discovering statistically significant biclusters in gene expression data. *Bioinformatics* **2002**, *18*, S136–S144.
19. Gu, J.; Liu, J.S. Bayesian biclustering of gene expression data. *BMC Genom.* **2008**, *9*, S4. [CrossRef]
20. Han, J.; Cheng, H.; Xin, D.; Yan, X. Frequent pattern mining: current status and future directions. *Data Min. Knowl. Discov.* **2007**, *15*, 55–86. [CrossRef]
21. Djenouri, Y.; Belhadi, A.; Djenouri, D.; Lin, J.C.W. Cluster-based information retrieval using pattern mining. *Appl. Intell.* **2021**, *51*, 1888–1903. [CrossRef]
22. Belhadi, A.; Djenouri, Y.; Lin, J.C.W.; Cano, A. A general-purpose distributed pattern mining system. *Appl. Intell.* **2020**, *50*, 2647–2662. [CrossRef]
23. Wu, J.M.T.; Srivastava, G.; Wei, M.; Yun, U.; Lin, J.C.W. Fuzzy high-utility pattern mining in parallel and distributed Hadoop framework. *Inf. Sci.* **2021**, *553*, 31–48. [CrossRef]
24. Azzam, B.; Harzendorf, F.; Schelenz, R.; Holweger, W.; Jacobs, G. Pattern discovery in white etching crack experimental data using machine learning techniques. *Appl. Sci.* **2019**, *9*, 5502. [CrossRef]
25. Cheung, D.W.; Han, J.; Ng, V.T.; Wong, C.Y. Maintenance of discovered association rules in large databases: an incremental updating technique. In Proceedings of the Twelfth International Conference on Data Engineering, New Orleans, LA, USA, 26 February–1 March 1996, pp. 106–114. [CrossRef]
26. Li, Y.; Zhang, Z.H.; Chen, W.B.; Min, F. TDUP: an approach to incremental mining of frequent itemsets with three-way-decision pattern updating. *Int. J. Mach. Learn. Cybern.* **2017**, *8*, 441–453. [CrossRef]
27. Lin, C.; Hong, T.; Lu, W. The Pre-FUFP algorithm for incremental mining. *Expert Syst. Appl.* **2009**, *36*, 9498–9505. [CrossRef]
28. Nath, B.; Bhattacharyya, D.K.; Ghosh, A. Incremental association rule mining: a survey. In*Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*; Wiley: Hoboken, NJ, USA, 2013; Volume 3.
29. Koh, J.L.; Shieh, S.F. An Efficient Approach for Maintaining Association Rules Based on Adjusting FP-Tree Structures. *Lect. Notes Comput. Sci.* **2004**, *2973*, 417–424.
30. Sun, J.; Xun, Y.; Zhang, J.; Li, J. Incremental Frequent Itemsets Mining with FCFP Tree. *IEEE Access* **2019**, *7*, 136511–136524. [CrossRef]
31. Han, J.; Pei, J.; Yin, Y.; Mao, R. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.* **2004**, *8*, 53–87. [CrossRef]
32. Giang, N.; Son, L.; Ngan, T.; Tuan, T.; Phuong, H.; Abdel-Basset, M.; de Macêdo, A.R.L.; de Albuquerque, V.H.C. Novel Incremental Algorithms for Attribute Reduction From Dynamic Decision Tables Using Hybrid Filter-Wrapper With Fuzzy Partition Distance. *IEEE Trans. Fuzzy Syst.* **2020**, *28*, 858–873. [CrossRef]
33. Goethals, B.; Zaki, M. Advances in frequent itemset mining implementations: introduction to FIMI'03. In Proceedings of the Workshop on FIMI, Melbourne, FL, USA, 19 September 2003.
34. Thurachon, W.; Kreesuradej, W. Incremental association rule mining with a fast incremental updating frequent pattern growth algorithm. *IEEE Access* **2021**, *9*, 55726–55741. [CrossRef]
35. Sun, J.; Huang, Q. Two stages biclustering with three populations. *Biomed. Signal Process. Control.* **2023**, *79*, 104182. [CrossRef]