EFFICIENT MINING OF CLOSED SETS FROM HIGH DIMENSIONAL DATASETS

[1]R.V. Nataraj and [2]S.Vinoth Kumar

Department of Information Technology, PSG College of Technology, rvn@ieee.org
Computer Science Corporation, Chennai csevino@gmail.com, Coimbatore-641004, TamilNadu, India

ABSTRACT

In this paper, we address the problem of mining closed sets from four-dimensional datasets. Several algorithms have been proposed for mining frequent closed sets from transactional datasets. However, these algorithms are limited to two-dimensional datasets and three-dimensional datasets. The collected data in the context of transactional database contains much interesting useful information spanning several dimensions. These multiple dimensions used for analyzing and knowledge discovery in database system induced to present the QuadMiner algorithm, an efficient algorithm for mining closed four-sets from four-dimensional datasets. Experimental results involving several synthetic datasets show that the algorithm takes less running time than the recently proposed DataPeeler algorithm.

Keywords : Association Rule Mining Data Mining Knowledge Discovery Closed 4-set.

1. INTRODUCTION

The mining of frequently occurring sets plays an important role in many data mining tasks. Frequent set mining can produce a large number of frequent sets and many of them are redundant, which reduce the efficiency and effectiveness of the mining process. For many applications, mining all the frequent sets is not necessary, and mining the closed sets too will provide the same amount of information. Rather than frequent set mining, frequent closed set mining will generate less number of sets, improves the efficiency and effectiveness of the mining tasks. Different techniques and algorithms have been proposed to solve this problem in both 2D and 3D space. Several algorithms have been proposed including FP-Close [7], A-Close, AFOPT-Close [9], B-Miner & C-Miner [2], and DCI-Close [6]. These algorithms are all limited to 2D dataset analysis. The *gene-time dataset, gene-sample dataset in* biological micro array analysis, and the *transaction-itemset* datasets in market basket data analysis are example of 2D datasets. To enumerate all the closed 3-sets (also known as frequent closed cubes) from 3D dataset, several algorithms have been proposed including RSM (Representative Slice Mining) [1], Cube-Miner [1] and TRIAS [5].

The RSM framework generates 2D representative slices from the 3-dimensional dataset. After generating the representative slices, the framework uses 2D closed set algorithm to mine 3-dimensional closed sets. 3D closed sets are generated by combining each 2D closed set with heights contributing to its representative slices. Finally, post-pruning strategy removes all the unclosed 3-dimensional sets. This framework is efficient, only if one of the dimensions is small. Unlike RSM algorithm, the Cube-Miner algorithm generates frequent closed cubes on 3-dimensional space directly. Cube-Miner algorithm is efficient for large datasets. However, forth-mentioned algorithms are limited to 3D datasets. The *gene-sample-time* micro array data and *transaction-itemset-time in* market basket analysis are examples of 3D datasets. Nowadays, due to advancement in technology, multidimensional datasets are very common and hence efficient algorithms are required to mine closed sets from datasets greater than 3-dimensions. The Data Peeler [4] is a recently proposed algorithm to enumerate closed set from datasets greater than 3-dimensions.

In this paper, we propose an algorithm for mining frequent closed sets from 4-D datasets. The rest of this paper is organized as follows. Section 2 presents the basic notations and the problem definition associated with this paper. Section 3 presents QuadMiner algorithm, the pseudo code and its description while section 4 analyzes the experimental results comprehensively. The paper is concluded in section 5.

2. PRELIMINARIES

In this section, we present the basic notations and definitions that we will be using throughout this paper. A dataset D is a quadset, D= (D_1, D_2, D_3 and D_4) where $D_1 = \{c_1, c_2, \dots, c_k\}$ denotes a set of columns, $D_2 = \{r_1, r_2, \dots, r_l\}$ denotes a set of rows, $D_3 = \{h_1, h_2, \dots, h_m\}$ denotes a set of heights, $D_4 = \{q_1, q_2, \dots, q_n\}$ denotes a set of cubes. Then a four dimensional dataset can be represented by k×l×m×n binary multidimensional matrix O= $D_1 \times D_2 \times D_3 \times D_4$. Each cell $O_{q,h,r,c}$ corresponds to the relationship among cubes q_k, height h_l, row r_m and column c_n. The presence of '0' indicates that *q, h, r and c* are not connected, whereas '1' indicates that they are connected. A sample four dimensional dataset is shown in Table 1.

Table 1: An Example 4-Dimensional Dataset

		q_1			q_2		
h_1							
		c1	c2	c3	c1	c2	c3
	r1	1	0	0	1	1	0
	r2	0	1	1	0	1	0
h2							
		c1	c2	c3	c1	c2	c3
	r1	0	0	1	0	1	0
	r2	1	1	1	1	1	1

Definition 2.1 Closed 4-set: Given a set of dimension $D_1' \subseteq D_1, D_2' \subseteq D_2, D_3' \subseteq D_3$ and $D_4' \subseteq D_4$, a 4-set $S=(D_1' \times D_2' \times D_3' \times D_4')$ is defined as a closed 4-set when it satisfies the following, (a) $D_1 = D_1(D_2 \times D_3 \times D_4)$, (b) $D_2 = D_2(D_1 \times D_3 \times D_4)$, (c) $D_3 = D_3(D_1 \times D_2 \times D_4)$ and (d) $D_4 = D_4(D_1 \times D_2 \times D_3)$. (a), (b), (c) and (d) are referred to as "closed" in column set, row set, height set and quad set respectively.

Definition 2.2 Frequent Closed 4-set: A 4-set $S=(D_1 \times D_2 \times D_3 \times D_4) \in O$ is called as frequent closed 4-set if D_1 support $|D_1(D_2 \times D_3 \times D_4)|$, D_2 support $|D_2(D_1' \times D_3' \times D_4')|$, D_3 support $|D_3(D_1 \times D_2 \times D_4)|$ and D_4 support $|D_4(D_1 \times D_2 \times D_3)|$ are higher than the user defined minimum D_1 support (min_s_D_1), minimum D_2 support (min_s_D_2), minimum D_3 support (min_s_D_3) and minimum D_4 support (min_s_D_4) respectively.

Table 2: Closed 4-sets

Sl. No	Closed 4-sets
1	q2 : h1h2 : r1r2 : c2
2	q1q2 : h2 : r2 : c1c2c3
3	q1 : h2 : r1r2 : c3
4	q1 : h1h2 : r2 : c2c3
5	q1q2 : h1h2 : r2 : c2
6	q1q2 : h1 : r1: c1

Problem Definition: Given a 4-dimensional dataset, the problem is to enumerate all the closed 4-sets satisfying the user specified minimum size constraints.

3. QUADMINER ALGORITHM

In this section, we present the QuadMiner algorithm, which enumerates the closed 4-sets and operates directly on the 4-dimensional datasets. The quad-miner algorithm is inspired from the cube-miner [1] algorithm and generates a quad tree from the input dataset. The root node of the quad-miner algorithm tree consists of all the elements of all dimensions. The algorithm is based on '0' removing principle (note that a closed 4-set is a 4D set with all 1's i.e. all the elements are in relation with each other) and starting from the root node, the child nodes are generated by removing zeros. For removing zeros, the cutter concept [5] is used. A cutter is a 4D-set containing all 0's having an element from each dimension except the first dimension which may contain more than one element from the column set. Set A(D_1', D_2', D_3' and D_4') is called a "cutter" as a whole and each atom (D_1', D_2', D_3', D_4') are called as the extreme left atom, middle left atom, middle right atom and extreme right atoms respectively. Note that, an element of the cutter is referred to as an atom. Applying the cutter on a node results in four child nodes namely, the extreme left child, middle left child, middle right child and extreme right child. The extreme left child node is generated by removing the quad element of cutter and the middle left child node is generated by removing height element of cutter. Similarly, the middle right child node is generated by removing row element of cutter and the extreme right child is generated by removing the column elements of cutter.

For example, the root node consists of entire 4-set attributes (Column, Row, Height, quad) as shown in Fig 1. The cutter for the root node is q1: h1: r1: c2c3 and the resulting extreme right child, the middle right child, middle left child and the extreme left child of the root node are ER (*q1q2, h1h2, r1r2, c1*), MR (*q1q2, h1h2, r2, c1c2c3*), ML (*q1q2, h2, r1r2, c1c2c3*) and EL(*q2, h1h2, r1r2, c1c2c3*) respectively. While generating the child nodes, the algorithm does several checking to ensure the size, uniqueness and the closeness of the node. If these conditions are satisfied, then there is no further cutter applicable for a node, the algorithm concludes the node as a closed 4-set. For example, Table 2 shows all the closed 4-sets for the example dataset given in Table1. In the following section, we propose track checking to remove duplicates (unicity checking) and closeness checking to remove subsets.

3.1 Removing duplicates

While generating the quadtree using the cutter concept, a duplicate closed 4-set pattern may also be generated. While generating the child nodes (the extreme right, middle right, middle left and extreme left), we perform the following to remove duplicate 4-sets.

(1) Extreme Left track checking

This checking is done only for extreme left child node from middle left branch, middle right branch and extreme right branch. In extreme left track checking, the extreme left atom of the current cutter is compared with the extreme left atom of the set of cutters of the path from the root node. For example, the extreme left atom *q1* of cutters (*q1, h2, r1, c1c2*), (*q1, h1, r2, c1*) and (*q1, h1, r2, c1*) has already cut the path of middle left child node EL (*q2, h2, r1r2, c1c2c3*) (*a1* in level 2) , middle right child node EL (*q2, h1h2, r2, c1c2c3*) (a2 in level 5) and extreme EL (*q2, h1h2, r1r2, c1*) (a3 in level 2) respectively. Hence, the following are to be pruned-off as the subset of node EL (*q2, h1h2, r1r2, c1c2c3*): a1 from the middle left child node branch, *a2* from the middle right child branch and *a3* from the extreme right child branch.

(2) Middle left track checking

This checking is done only for middle left child node from middle right branch and extreme right branch. In middle left track checking, the middle left atom of the current cutter is compared with the middle left atom of the set of cutters of the path from the root node. For example, the middle left atom *h1* of cutter (*q1, h1, r2, c1*) has already cut the path of middle left child node ML (*q1q2, h2, r2, c1c2c3*) (*b1* in level 5), MR (*q1q2, h2, r1r2, c1*) (*b2* in level 1). Hence, the following are to be pruned-off as the subset of node ML (*q1q2, h2, r2, c1c2c3*): *b1*,*b2*, *b3*, *b4* and *b5*.

(3) Middle Right Track checking

□ This checking is done only for middle right child node from extreme right branch. In middle right track checking, the middle right atom of the current cutter is compared with the middle right atom of the set of cutters of the path from the root node. For example, the middle right atom *r1* of cutter (*q2, h2, r1, c3*) has already cut the path of extreme right child node ER (*q1q2, h2, r2, c3*) (*c1* in level3). Hence c1 should be pruned off as a subset which may

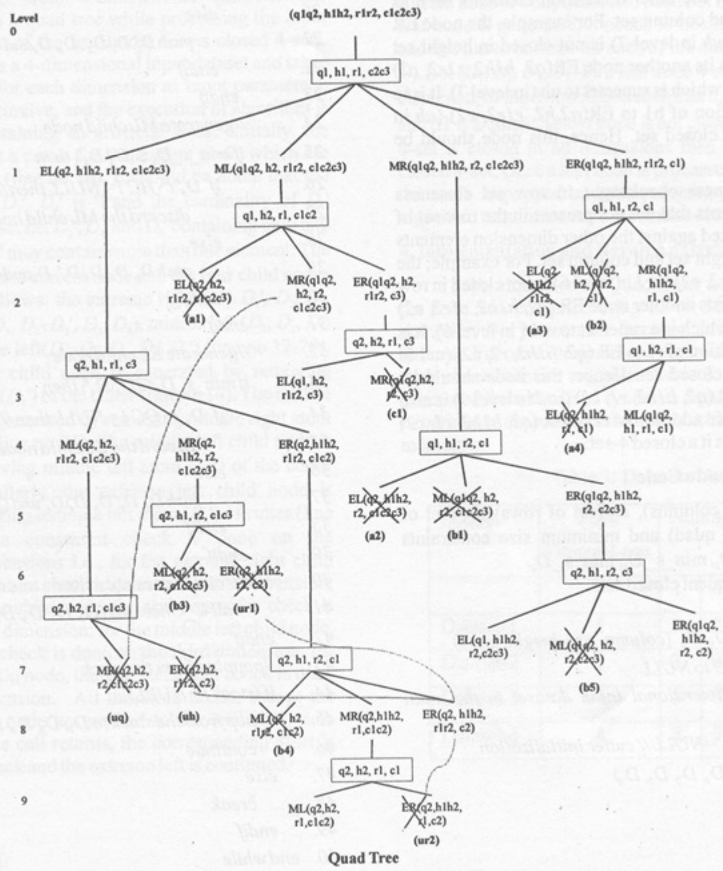

Figure 1: The Quad Tree Generated by the QuadMiner Algorithm

3.2 Removing subsets

Closeness checking is required to remove subsets. This closeness checking operations are done only in the leaf node as follows

(a) **Quad set closeness checking:** In quad set closeness checking, the elements that are not present in the quad set of leaf nodes are checked against the other dimension elements such as height set, row set and column set. For example, the node MR (*q2, h2, r2, c1c2c3*) (*uq* in level 7) is not closed in quad set because there exists another node MR (*q1q2, h2, r2, c1c2c3*) (2nd node in level 2) which is a superset to uq (in level 7). It is to be noted that, addition of q1 to MR (*q2, h2, r2, c1c2c3*) (*uq* in level 7) makes it a closed 4-set. Hence, this node should be pruned.

(b) **Height set closeness checking :**. In this checking, the elements that are not present in the height set of leaf nodes are checked against the other dimension elements such as quad set, row set and column set. For example, the node ER (*q2, h2, r1r2, c2*) (*uh* in level 7) is not closed in height set because there exists its another node ER(*q2, h1h2, r1r2, c2*) (3rd node in level 8) which is superset to uh (in level 7). It is to be noted that, addition of h1 to ER(*q2,h2, r1r2, c2*) (*uh* in level 7) makes it a closed set. Hence, this node should be pruned.

(c) **Row set closeness checking** : In row set closeness checking, the elements that are not present in the row set of leaf nodes are checked against the other dimension elements such as quad set, height set and column set. For example, the node ER (*q2, h1h2, r2, c2*) (*ur1* in level 6) is not closed in row set because there exists another node ER (*q2, h1h2, r1r2, c2*) (3rd node in level 8) which is a superset to ur1 (in level 6). It is to be noted that, addition of r1 to ER (*q2, h1h2, r2, c2*) (ur1 in level 6) makes it a closed set. Hence, this node should be pruned-off. Node ER (*q2, h1h2, r1, c2*) (*ur2* in level 9) is also to be pruned-off, since addition of r2 to ER (*q2, h1h2, r1, c2*) (*ur2* in level 9) makes it a closed 4-set.

3.3 QuadMiner Pseudo Code

INPUT : D_1 (set of columns), D_2 (set of rows), D_3 (set of heights), D_4 (set of quad) and minimum size constraints min_s_ D_1, min_s_ D_2, min_s_ D_3, min_s_ D_4.
OUTPUT: set of frequent closed sets

```
1. Initialize $D_1$, $D_2$, $D_3$, $D_4$ (column, row, height, quad)
2. set QC,HC and RC to NULL
3. construct the n-dimensional input dataset in the main
   memory
4. $D_1' = D_2' = D_3' = D_4'$=NULL// cutter initialization
5. Call QuadMiner($D_1$, $D_2$, $D_3$, $D_4$)
6.QuadMiner($D_1$, $D_2$, $D_3$, $D_4$)
7. {
8.   while(true)
9.       generate cutter for current dimensions
10.      if(cutter present) then
11.        update QC, HC, RC, $D_1'$, $D_2'$, $D_3'$ and $D_4'$
12.        //generate ER child node
13.        if min_s_$D_1 \leq |D_1 \backslash D_1'|$ then
14.            push $D_1 \backslash D_1'$, $D_2$, $D_3$, $D_4$ and QC, HC, RC to stack
15.        endif
16.        //generate MR child node
17.        if min_s_$D_2 \leq |D_2 \backslash D_2'|$ then
18.            if $D_2' \cap RC$ != NULL then// MR track check
19.                discard the MR child node as duplicate
20.            else
21.                push $D_1$, $D_2 \backslash D_2'$, $D_3$, $D_4$ and QC, HC, RC to stack
22.            endif
23.        endif
24.        //generate ML child node
25.        if min_s_$D_3 \leq |D_3 \backslash D_3'|$ then
26.            if $D_3' \cap HC$ != NULL then// ML track check
27.                discard the ML child node as duplicate
28.            else
29.                push $D_1$, $D_2$, $D_3 \backslash D_3'$, $D_4$ and QC, HC, RC to stack
30.            endif
31.        endif
32.        //generate EL child node
33.        if min_s_$D_4 \leq |D_4 \backslash D_4'|$ then
34.            if $D_4' \cap QC$ != NULL then// EL track check
35.                discard the EL child node as duplicates
36.            else
37.                push $D_1$, $D_2$, $D_3$, $D_4 \backslash D_4'$ and QC, HC, RC to stack
38.            endif
39.        endif
40.      else// cutter does not exits
41.          call C_Checking($D_1$, $D_2$, $D_3$, $D_4$)
42.      endif
43.      //popping from the stack
44.      if(stack != NULL)
45.         pop from the stack to $D_1$, $D_2$, $D_3$, $D_4$ and QC, HC, RC
46.         continue
47.      else
48.         break
49.      endif
50.  end while
51. }
```

3.4 Closeness Checking Pseudo Code

```
1.  C_Checking($D_1, D_2, D_3, D_4$)
2.  {
3.      if $\neg\exists d_4 \in (D_4 \backslash D_4')$ && $d_4 \times D_3 \times D_2 \times D_1$ is true
4.          if $\neg\exists d_3 \in (D_3 \backslash D_3')$ && $D_4 \times d_3 \times D_2 \times D_1$ is true
5.              if $\neg\exists d_2 \in (D_2 \backslash D_2')$ && $(D_4 \times D_3 \times d_2 \times D_1$ is true
6.                  output $D_4 : D_3 : D_2 : D_1$ as a closed 4-set
7.              endif
8.          endif
9.      endif
10. }
```

3.5 Description

The QuadMiner Algorithm is a depth-first method to mine closed 4-sets from 4-dimensional datasets. The algorithm generates a quad tree while processing the input dataset and a subset of the leaf nodes forms closed 4-sets. This algorithm takes a 4-dimensional input dataset and takes the size constraint for each dimension as input parameters. The algorithm is recursive, and the execution of algorithm is controlled by maintaining an artificial stack. Initially, the algorithm generates a cutter for the current node, which are D_1', D_2', D_3' and D_4' (line no 10). It should be noted that the cardinality of D_2', D_3', D_4' is '1'and the cardinality of D_1' depends on the dataset i.e. D_2', D_3' and D_4' containing only one element whereas D_1' may contain more than one element. The cutter is applied on the current node and the four child nodes are generated as follows: the extreme right ($D_1 \backslash D_1'$, D_2, D_3, D_4), middle right (D_1, $D_2 \backslash D_2'$, D_3, D_4), middle left (D_1, D_2, $D_3 \backslash D_3'$, D_4) and extreme left(D_1, D_2, D_3, $D_4 \backslash D_4'$) (line no 12-29). The extreme right child node is generated by removing extreme right atom (D_1') of the cutter (line no 14). The middle right child node is generated by removing middle right atom (D_2') of the cutter (line no 21). The middle left child node is generated by removing middle left atom (D_3') of the cutter (line no 29). Similarly, the extreme left child node is generated by removing extreme left atom (D_4') of cutter (line no 37). The size constraint check is done on the corresponding dimensions i.e., for the extreme right child node, the size constraint check is done on the first dimension; for the middle right child node, the size constraint check is done on the second dimension; for the middle left child node, the size constraint check is done on the third dimension; for the extreme left child node, the size constraint check is done on the fourth dimension. All the three nodes, except the extreme left child node are pushed to the artificial stack. When the recursive call returns, the corresponding 4-set is popped from the stack and the extreme left is continued.

The above-explained cutting phase goes on repeating until no more cutters are available or the node is found to be a duplicate. The duplicate checking is carried out as follows: For the extreme left nodes, a cutter list QC is maintained which contains all the extreme left cutters, which were used in the path of current node from the root node. Let (*qc, hc, rc, cc*) be the cutter generated for the current node. If qc $\cap$ QC is not empty, then the left extreme left child node for the current node is a duplicate and need not be generated. For the middle left nodes, a cutter list HC is maintained, which contains all the middle left cutters, which is used in the path of the current node from the root node. If hc $\cap$ HC is not empty then the middle left child node of the current node is a duplicate and need not be generated. For the middle right nodes, a cutter list RC is maintained which contains all the middle right cutters, which is used in the path of the current node from the root node. If rc $\cap$ RC is not empty then the middle left track for the current node is a duplicate and need not be generated. Once a leaf node is generated, it needs to be checked for closeness and this has to be done for each of the dimensions -D_1, D_2 and D_3. Let (D_1, D_2, D_3, D_4) be a leaf node 4-set. If there exists an element d_3 of third dimension such that $d_3 \times D_1 \times D_2 \times D_4$ is true and d3 $\notin D_3$ then the leaf node is not closed. If the leaf node's 4-set is closed in all dimensions then it is outputted as a closed 4-set. Once a leaf node is processed, the top element of the stack is popped and the processing is continued. The algorithm terminates once the stack becomes empty.

4. Implementation and Result Analysis

We implemented QuadMiner using C language and the code was compiled using 32-bit Microsoft Visual C++ compiler. We have used Boolean data type to store the datasets. For our experiments, we have used four datasets and their characteristics are given Table 3. All the datasets are generated from the IBM synthetic dataset generator. All the experiments are run on Pentium 4 machine with 1GB of main memory.

Table 3: Datasets Used

Dataset	Total Dimensi-ons	Number of instances			
		4	3	2	1
Dataset-1	4	3	3	15	100
Dataset-2	4	3	3	15	500
Dataset-3	4	3	3	15	1000
Dataset-4	4	3	3	15	5000

To get the accurate time to the extent possible, we have made sure that no other programs were running in the background while conducting the experiments. Table 4-7 show the running time in seconds for both QuadMiner and DataPeeler algorithm. For fair comparison, the datapeeler algorithm has also been implemented using C language and the data structures used in both algorithms are same. The QuadMiner is based on zero removing principle whereas the datapeeler algorithm is based on 1 growing principle. In almost all cases, the experimental results shows that QuadMiner takes less running time when compared to datapeeler algorithm and this is mainly due to the following two reasons. In QuadMiner, many elements are removed while generating the child nodes whereas in datapeeler algorithm one element is chosen to generate the two child nodes. 2) each and every node of QuadMiner consists of only one 4-set whereas two 4-sets are present in the nodes of DataPeeler enumeration tree.

Table 4: Total running time in Seconds for QuadMiner and Data Peeler algorithm for dataset-1

Minimum support				Total Running time in Seconds	
D4	D3	D2	D1		
				QuadMiner	Data Peeler
1	1	1	1	1.806	3.1605
1	1	2	2	1.606	2.8105
1	1	3	3	1.324	2.317
1	1	4	4	0.998	1.7465
1	1	5	5	0.670	1.1725

Table 5: Total running time in Seconds for QuadMiner and Data Peeler algorithm for dataset-2

Minimum support				Total Running time in Seconds	
D4	D3	D2	D1		
				QuadMiner	Data Peeler
1	1	1	1	18.172	31.801
1	1	2	2	17.532	30.681
1	1	3	3	16.616	29.078
1	1	4	4	15.229	26.65075
1	1	5	5	13.109	22.94075

Table 6: Total running time in Seconds for QuadMiner and Data Peeler algorithm for dataset-3

Minimum support				Total Running time in Seconds	
D4	D3	D2	D1		
				QuadMiner	Data Peeler
1	1	1	1	68.877	103.3155
1	1	2	2	68.829	103.2435
1	1	3	3	68.369	102.5535
1	1	4	4	66.432	99.648
1	1	5	5	61.874	92.811

Table 7: Total running time in Seconds for QuadMiner and Data Peeler algorithm for dataset-4

Minimum support				Total Running time in Seconds	
D4	D3	D2	D1		
				QuadMiner	Data Peeler
1	1	1	1	178.264	267.396
1	1	2	2	120.119	180.1785
1	1	3	3	119.008	178.512
1	1	4	4	116.079	174.1185
1	1	5	5	108.377	162.5655

5. CONCLUSION

We have proposed Quad-miner algorithm, which operates on four-dimensional datasets directly and uses a quad- tree enumeration strategy for generating closed 4-sets. Our comprehensive experimental results here show that QuadMiner algorithm outperforms DataPeeler in most cases. In our future work, we have planned to create real datasets to analyze the algorithm performance. In addition, we are further working on to improve the performance by optimizing the code and the data structures used in the algorithm.

Acknowledgments

We wish to thank Liping Ji, who has kindly provided executables of RSM and CubeMiner. We would like to thank authors Loïc Cerf, Jérémy Besson Céline Robardet, and Jean-François Boulicaut for providing us the executables of Datapeeler algorithm and responding to our numerous queries.

REFERENCES

1. Liping, Ji, Tan, K.L. and Tung, A.K.H. "Mining Frequent ClosedCubes in 3D datasets," Proc. 32nd int. conference on very large databases, 2006
2. Liping, Ji, Kian-Lee Tan,K H. Tung, "Compressed Hierarchical Mining of Frequent Closed Patterns from Dense Data Sets," IEEE Transaction on Knowledge and Data Engineering, Vol 19, No.9, Sept 2007.
3. Liping, Ji "Mining Localized Co-expressed Gene Patterns from Microarray Data," PhD dissertation, School of Computing., Na-tional University of Singapore, June 2006.
4. Loic Cerf, Jeremy Besson, Celine Robardet, and Jean-Francois Boulicaut, "Data Peeler: Contraint-Based Closed Pattern Mining in nary Relations", Proceedings of the 2008 SIAM International Conference on Data Mining, pp. 37-48, Atlanta, Apr-2008.
5. Robert Jaschke, Andreas Hotho, Christoph Schmitz, Bernhard Ganter, Gerd Stumme "TRIAS An Algorithm for Mining Iceberg Tri-Lattices" Proceedings of the Sixth International Conference o n Data Mining (ICDM'06).
6. Lucchese,C. Orlando, S. and Perego, R. "Fast and Memory Efficient Mining of Frequent Closed Itemsets", IEEE Transactions on Knowledge and Data Engineering, VOL 18, No 1, pages 21-36, January 2006.
7. Grahne,G., Zhu, J. "Fast Algorithms for Frequent Itemset Mining Using FP-Trees", IEEE Transactions on Knowledge and Data Engineering, Vol 17, No 10, pages 1347-1362, October 2005.
8. Grahne, G. Zhu, J. "Fast Algorithms for Frequent Itemset Mining Using FP-Trees", IEEE Transactions on Knowledge and Data Engineering, Vol 17, No 10, pages 1347-1362, October 2005.
9. Liu, G. "Supporting Efficient and Scalable Frequent Pattern Mining," PhD dissertation, Dept. o f Computer Science., Hong Kong University., May 2005
10. Pan, F., Cong,G., Tung,,A.K.H., Yang, J. and Zaki, M. J. "CARPENTER: Finding Closed Patterns in long Biological Datasets", SIGKDD-03, 2003.
11. Gao Cong, Kian-Lee Tan, Tung, A. K. H. and Feng Pan, Mining Frequent Closed Patterns in Microarray Data", ICDM'04, Vol 1, Issue 4, pp: 363-366, Nov 2004.
12. Besson, J., Robardet, C. and Boulicaut, J.F. "Constraint based mining of formal concepts in trasactional data, PAKDD'04 pp. 615-624, 2004.
13. Ben Yahia, S., Hamrouni,T. and Mephu Nguifo, E. "Frequent Closed itemset based Algorithms: A through structural and analytical survey," SIGKDD Explorations, Vol. 8, issue 1, pages 93-104 2006.
14. Besson, J., Robardet, C., Boulicaut J.F. and S. Rome,"Constraint Based Concept Mining and its Application to Microarray Data Analysis", Journal o f Intelligent Data Analysis, pp. 59-82, 2005
15. Wang,J., Han J., and pei, J. "CLOSET+: searching for the Best Strategies for Mining Frequent Closed Itemsets", Proc. 2003 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, pages 236-245, Aug 2003.