Closed Pattern Mining from n-ary Relations

R V Nataraj
Department of Information Technology
PSG College of Technology
Coimbatore, India

S Selvan
Department of Computer Science
Francis Xavier Engineering College,
Tirunelveli, India.

ABSTRACT

In this paper, we address the problem of closed pattern mining from n-ary relations. We propose CnS-Miner algorithm which enumerates all the closed patterns of the given n-dimensional dataset in depth first manner satisfying the user specified minimum size constraints. From the given input, the CnS-Miner algorithm generates an n-ary tree and visits the tree in depth first manner. We have proposed a generalized duplicate pruning method which prunes the subtrees that generate duplicate patterns. The space complexity of our algorithm is *O(D+d)* where *D* is the n-ary dataset and *d* is the depth of the tree. We have experimentally compared the proposed algorithm with DataPeeler, a recently proposed algorithm for closed pattern mining from n-ary relations.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications – Data Mining

General Terms

Algorithms, Design, Experimentation.

Keywords

Data Mining, Closed Patterns, Algorithms

1. INTRODUCTION

Closed pattern mining algorithms including LCM [6], DCI_Close [9], FP-Close [8], AFOPT-Close [10] are limited to 2D datasets. These algorithms were proposed in the context of association rule mining and are optimized for transactional datasets. Closed pattern mining algorithms optimized for gene expression data in 2D context have also been proposed including D-Miner [2], B-Miner & C-Miner [5] and CARPENTER. Also, to efficiently mine closed patterns from 2D symmetric adjacency matrix, LCM-MBC [3] algorithm has been proposed. Closed pattern mining has also been studied in 3D context [4] and two algorithms (CubeMiner algorithm & Representative Slice Miner algorithm) have been proposed to mine 3-dimensional closed patterns. A generalized efficient algorithm for closed pattern mining from n-dimensional data (alternatively n-ary relations) appears to be a timely challenge. In the traditional market-basket scenario, it is very common to collect data on 4 dimensions including customer, location, time and item. Association analysis based on multiple dimensions can reveal interesting patterns about how one dimension influences the other dimensions. To enumerate closed patterns from n-dimensional datasets, DataPeeler algorithm [1] has been proposed which enumerates all the n-dimensional closed patterns in depth first manner using a binary tree enumeration strategy. Also, the DataPeeler algorithm is memory efficient since it does not store the previously computed patterns in main memory for duplicate detection and closure checking.

This paper proposes CnS-Miner (**C**losed **n**-**S**et pattern **M**iner) algorithm to enumerate closed patterns from n-dimensional Boolean datasets. Similar to DataPeeler algorithm, the CnS-Miner algorithm enumerates all the closed patterns in depth first manner and the algorithm does not store the previously computed closed patterns in main memory. Unlike DataPeeler algorithm, which generates a binary tree, the CnS-Miner algorithm generates an n-ary tree from the input dataset for mining closed patterns. Also, the CnS-Miner algorithm prunes all the nodes which generate duplicate patterns and the number of comparisons required to prune a node is *O(d)* where *d* is the depth of the node from the root node. The space complexity of CnS-Miner algorithm is *O(D+d)* where *D* is the n-dimensional dataset and *d* is the depth of the tree. Experiments involving several synthetic datasets show that CnS-Miner algorithm outperforms DataPeeler algorithm in certain cases where one of the dimensions contains large number of elements when compared to other dimensions.

The rest of the paper is organized as follows. Section 2 presents the preliminaries. Section 3 presents the duplicate pruning method, CnS-Miner algorithm and the description of the algorithm. Section 4 analyzes the experimental results while section 5 concludes the paper.

2. PRELIMINARIES

This section presents the basic definitions followed by the problem definition. Let D = { D_1, D_2, D_3,.. D_n } be the n-dimensional dataset where D_i denote the i^{th} dimension elements. An example 4-dimensional dataset with D_1={ a_1, a_2, a_3 }, D_2 ={ b_1, b_2, b_3 }, D_3={ c_1, c_2, c_3 } and D_4={ d_1, d_2 }is shown in Table 1. An n-set, U = { P_1, P_2..P_n}, where $P_i \subseteq D_i$, is said to be closed if and only if (i) all the elements of each set of P_i are in relation with all other elements of other dimensions and (ii) P_i cannot be enlarged without violating (i). For example, $d_1d_2 : c_1c_2c_3 : b_1b_2 : a_1$ is a closed 4-set pattern in Table 1.
Problem Definition: Given an n-dimensional dataset, the problem is to enumerate all the n-dimensional closed patterns satisfying the user specified minimum size constraints.

Table 1: An Example 4-Dimensional Dataset

		d_1			d_2		
		a_1	a_2	a_3	a_1	a_2	a_3
c_1	b_1	1	0	0	1	0	1
	b_2	1	0	1	1	0	1
	b_3	0	1	1	0	1	1
c2	b_1	1	0	0	1	0	0
	b_2	1	0	0	1	0	0
	b_3	1	1	1	1	1	1
c3	b_1	1	1	1	1	0	1
	b_2	1	0	1	1	0	1
	b_3	1	1	1	0	1	0

3. CLOSED n-Set PATTERN MINING

The proposed CnS-Miner algorithm is inspired by the D-Miner algorithm [2] which enumerates all the closed patterns satisfying the user specified minimal size constraints from two dimensional datasets and CubeMiner algorithm [4] which enumerates all the closed patterns from three dimensional datasets. The D-Miner algorithm starts with the entire 2-set (entire set of row and column elements) in the root node and generates a binary tree using cutters. A cutter in D-Miner algorithm is a 2-set containing one row element and a set of column elements such that none of the column elements are in relation with the row element. The proposed algorithm uses the same cutter concept but a cutter in CnS-Miner algorithm is an n-set containing one element from each dimension except the first dimension which contains a set of elements (in this paper, the column is assumed to be the first dimension). In closed 2-set pattern mining, the duplicate patterns will not be generated because all the nodes of D-Miner's binary tree are unique with respect to each other except the nodes which are in the same path from the root node i.e. only nodes in the same path from the root can be related by subset-superset relationship and no two nodes from different paths can be related by subset-superset relationship. This is due to, if a cutter with a particular row element is applied, then another cutter with the same row element will not occur. Hence, no duplicate patterns are generated in the closed 2-set pattern mining. However, the same will not hold true in an n-ary tree that is generated using cutter concept and duplicate patterns are likely to be generated. Hence, an efficient duplicate pruning method is needed to prune duplicate patterns.

3.1 Duplicate Pruning:

Let us recall why duplicate patterns are not generated in closed 2-set pattern mining of D-Miner algorithm. The reason is that all the nodes that are from different paths are unique with respect to each other. Hence, in the n-ary tree of closed n-set pattern mining, the uniqueness of all the nodes from different paths need to be ensured. The following examines when a particular node from a path becomes a subset to a node from a different path and then a generalized solution is proposed to prune such nodes.

Let us consider the root node, its cutter and its child nodes for the example dataset given in Table 1. The root node and its cutter are { d_1d_2 : $c_1c_2c_3$: $b_1b_2b_3$: $a_1a_2a_3$ } and { d_1 : c_1 : b_1 : a_2 a_3} respectively. The child nodes are { d_2 : $c_1c_2c_3$: $b_1b_2b_3$: $a_1a_2a_3$ } { d_1d_2 : c_2c_3 : $b_1b_2b_3$: $a_1a_2a_3$ } { d_1d_2 : $c_1c_2c_3$: b_2b_3 : $a_1a_2a_3$ } and { d_1d_2 : $c_1c_2c_3$: $b_1b_2b_3$: a_1 }. A child node is denoted as k^{th} child if k^{th} dimension cutter element is removed from that child. All the child nodes of the root node are unique with respect to each other. The n^{th} child of the root node becomes a superset to a node of other branches if and only if n^{th} dimension cutter element is removed in any of the nodes of the subtree of n^{th} child's sibling nodes. In no other circumstances, the n^{th} child of the root node becomes a superset to a node generated in the subtree of sibling nodes. Similarly, the $(n-1)^{th}$ child of the root node becomes a superset to a node of other branches if and only if $(n-1)^{th}$ dimension cutter element is removed in the subtree of $(n-1)^{th}$ child's sibling nodes. This holds true for all the nodes in the entire tree. Hence, to ensure the unicity of all the nodes in the entire tree, a k^{th} dimension element of a cutter should not be removed in the entire subtree of the k^{th} node's siblings. For example, let Q denote an n-set of a node and Z denote an n-set cutter of Q. Let P_n, P_{n-1}, ..P_2, P_1 be the child nodes of Q and the k^{th} dimension of cutter element is removed in the k^{th} child. Then, the k^{th} dimension cutter element should not be removed in the entire subtree of P_i where $1 \leq i < k$. In this way, the unicity can be ensured. Also, for a node P_k, this constraint need not be enforced in the subtree of ${p_{(k+1)}}^{th}$ node since $P_{(k+1)}$ is already unique to all the nodes of the ${p_k}^{th}$ subtree by the same constraint since $(k+1)^{th}$ cutter element of Z is never removed in the subtree of ${P_k}^{th}$ node. For example, considering the root node and its cutter, the removal of cutter element c_1 in the 4th son subtree need not be checked since all the nodes of 4th son subtree are already unique to the nodes of 3rd son subtree because of d_1, the 4th dimension cutter element. Based on this discussion the generalized unicity constraint lemma is stated as follows.

Lemma 1: *A k^{th} son of a node (generated by applying a cutter Z) can be pruned if there exists a cutter Z' of type q in its path from the root node where q<k and the k^{th}-dimension-element(Z') ∩ k^{th}-dimension-element(Z) ≠ null.*

In a path, if a cutter generates a k^{th} child, then the cutter is said to be of type k. For example, the cutter { d_1 : c_1 : b_1 : a_2 a_3 } of the root node is said to be of type 3 for a node from the 3rd son subtree of the root node. Similarly, the same cutter is said to be of type 2 for a node from the 2nd son subtree of the root node. A generalized notation for an n-dimension is given below. Let us consider an n-dimensional data D = { D_n, D_{n-1}, D_{n-2}. … D_2, D_1 }. Let the cutter for the root node be c_n, c_{n-1}, c_{n-2},.. c_2, C_1 (note that only C_1 contains more than one element and hence denoted using uppercase). Then, the root node and its n child nodes are shown in Figure 1.

The lemma says that the cutter element c_n is never removed in the entire subtree of q^{th} son nodes where $(n-1) \geq q \geq 1$. Also, the cutter element c_{n-1} is never removed in the entire subtree of m^{th}

son nodes where $(n-2) \geq m \geq 1$. Similarly, the c_2 cutter element is never removed in the entire subtree of 1st son nodes. In Figure 1, the root node cutter is of type n for all the nodes of the entire subtree of nth son. Also, the root node cutter is of type (n-1) for all the nodes of the entire subtree of (n-1)th son and so on. This lemma is applicable for all the nodes of the entire n-ary tree.

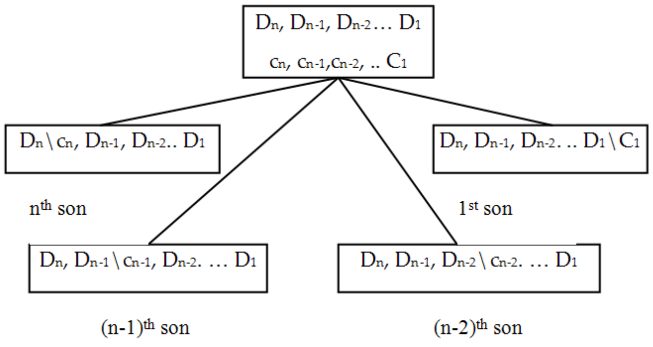
$D_n, D_{n-1}, D_{n-2} \dots D_1$
$C_n, C_{n-1}, C_{n-2}, \dots C_1$
$D_n \setminus C_n, D_{n-1}, D_{n-2} \dots D_1$
$D_n, D_{n-1}, D_{n-2} \dots D_1 \setminus C_1$
nth son
1st son
$D_n, D_{n-1} \setminus C_{n-1}, D_{n-2} \dots D_1$
$D_n, D_{n-1}, D_{n-2} \setminus C_{n-2} \dots D_1$
(n-1)th son
(n-2)th son

Figure 1. A Node in the n-ary Tree and its Child Nodes

3.2 CnS-Miner Pseudo Code

INPUT: An n-dimensional dataset, size constraints for each of the n dimensions (min_n, …. min_1).

OUTPUT: set of closed n-set patterns satisfying the size constraints.

```
1.  construct the dataset D in memory
2.  Z = null (cutter elements are initialized to null)
3.  ZL=null (cutter list is set to null)
4.  Initialize $Q_R$ (the root node)
5.  Call CnS-Miner ($Q_R$, ZL, n )

6.  CnS-Miner(Q, ZL, k)
7.  {
8.      while(k !=0)
9.          generate a cutter Z for Q
10.         if (cutter-exists)
11.             if $|Q_k \setminus Z_k| \geq$ min_k //check for size constraint
                    //check for duplicate
12.                 if $\exists Z' \in ZL$ and type(Z')<k and $Z'_k \cap Z_k \neq$ null
                        //recursively generate the child nodes
13.                     call CnS-Miner($Q \setminus Z_k$, $ZL \cup Z$, n)
14.                 endif
15.             endif
16.         else
17.             if (closed) //Perform Closure checking
18.                 write the closed n-set pattern to the disk
19.             endif
20.         endif
21.         k=k-1
22.     endwhile
23. }
```

The CnS-Miner algorithm starts with a root node containing the entire n-set. The notation Q indicates an n-set and Q_R indicates the entire n-set i.e. the root node. The notation Q_k indicates the k^{th} dimension elements of the corresponding n-set. The notation Z indicates the current cutter and Z_k indicates the k^{th} dimension elements of Z. For the root node, the algorithm generates a cutter. A cutter is also an n-set containing only one element in all the dimensions except the first dimension which may contain more than one element. If a cutter exists then left node (nth node) is generated provided both size constraint (line no 11) and duplicate constraint (line no 12) are satisfied. The algorithm uses a variable (in the pseudo code, the variable is denoted as k) to determine which child node is to be generated. Initially, when the CnS-miner algorithm is called, the variable k is initialized to the value of n. The value of k is decremented by 1 (line no 21) if a cutter does not exist or size constraint is not satisfied or a node is a duplicate or a recursive call is returned. When the value of k is decremented by 1 (line no 21) for the first time, k will have the value n-1 and the (n-1)th child will be generated. When k is again decremented, the (n-2)th child will be generated. This process gets repeated for all the nodes. When the value of k reaches zero and the stack is empty, the entire procedure returns. For duplicate checking, the algorithm maintains a variable named *ZL* which stores the set of cutters of a path from the root node to the current node. For a node, the closure checking (line no 17) is done only if no more child node can be generated i.e. a cutter does not exist for that node. If the n-set is found to be closed, then the corresponding n-set pattern is written into the disk as closed n-set pattern. Otherwise, the n-set is discarded. For the sake of simplicity, maintaining the type of a cutter is not shown in the pseudo code.

3.3 An Example

For the dataset given in Table 1, the root node of the 4-ary tree is $\{ d_1\ d_2 : c_1\ c_2\ c_3 : b_1\ b_2\ b_3 : a_1\ a_2\ a_3 \}$ and the cutter for the root node is $\{ d_1 : c_1 : b_1 : a_2a_3 \}$. Applying a cutter on the root node results in four child nodes namely $\{ d_2 : c_1\ c_2\ c_3 : b_1\ b_2\ b_3 : a_1\ a_2\ a_3 \}$, $\{ d_1\ d_2 : c_2\ c_3 : b_1\ b_2\ b_3 : a_1\ a_2\ a_3 \}$, $\{ d_1\ d_2 : c_1\ c_2\ c_3 : b_2\ b_3 : a_1\ a_2\ a_3 \}$ and $\{ d_1\ d_2 : c_1\ c_2\ c_3 : b_1\ b_2\ b_3 : a_1 \}$ respectively. Since the example dataset contains four dimensions, there are four types of cutters namely type 1, type 2, type 3 and type 4. The root node cutter is of type 4 for the 4th son and its entire subtree. Similarly, the root node cutter is of type 3, type2 and type 1 for the 3rd son, 2nd son and 1st son respectively and also for their entire subtree. For the 4th son of the root node, all the child nodes are generated. For the 4th son of the 3rd son of the root node, there exists a cutter of type 3 in its path from the root with same 4th dimension element and hence, according to lemma 1, the 4th son is pruned. Similarly, the 4th son of the 2nd son of the root node and 4th son of the 1st son of the root node are also pruned. This procedure is repeated for all the nodes of the entire 4-ary tree. The unicity constraint is applied for all the nodes of the entire n-ary tree.

3.4 Parallelization

Nodes of the n-ary tree generated by the CnS-Miner algorithm can be processed parallelly and concurrently on several machines. Each and every node along with a set of cutters in its path from the root node constitutes an independent subtask. The availability of the dataset in all the processors is the only requirement which can be easily done without much overhead. To achieve better load sharing in static context, an efficient approach would be to generate as much number of nodes in the master processor in breadth first manner and assign these nodes to the slave processors. When a slave processor completes its execution, another node from the master processor can be allocated to this slave processor for processing. In this way, the overall running time can be reduced and the load sharing among different processors can be improved. The work stealing concept can also be easily incorporated to achieve immediate load balancing. The only data to be transferred to the free processor is a node and a set of cutters in its path from the root node.

4. EXPERIMENTAL RESULTS

The CnS-Miner algorithm and DataPeeler algorithm are implemented using C language and the code is compiled using 32-bit Microsoft Visual C++ compiler. The data structures used in both the algorithms are same i.e. the data structure used to represent the n-set, the algorithm to access each elements of n-set, the data structure used for the representing the stack (note that both DataPeeler and CnS-Miner algorithm explores the tree in depth first manner and hence a stack is needed) and the methodology of stack manipulation are same in both CnS-Miner and DataPeeler algorithm. All the experiments were conducted on Pentium 4 machines with 1GB of main memory loaded with Windows XP operating system. Several synthetic datasets created using IBM synthetic dataset generator are used for experimental analysis and the description of the datasets is given in Table 2. The seven columns in Table 2 represent the total number of elements in each dimension i.e. for example, Dataset-1 in Table 2 has a total of 5 dimensions with 5th to 3rd dimension containing 2 elements, 2nd dimension containing 20 elements and the first dimension contains 5k elements. Similarly, the Dataset-2 contains a total of 7 dimensions as shown in Table 2.

Table 2 Datasets Used

Dataset	Number of Dimensions	Total Elements in each dimension						
		7	6	5	4	3	2	1
Dataset-1	5	-	-	2	2	2	15	5K
Dataset-2	7	2	2	2	2	2	15	5K

Table 3 Total Running Time in Seconds for CnS-Miner Algorithm and DataPeeler Algorithm

Dimension Size Constraints				Running Time	
Min-4	Min-3	Min-2	Min-1	CnS-Miner	Data Peeler
1	1	1	1	123.267	240.545
1	1	1	5	120.454	229.749
1	1	1	10	118.878	218.650
1	1	1	15	113.564	202.543
1	1	1	20	109.453	189.467
1	1	1	25	107.774	177.986

In Table 3 and Table 4, the Min-x represents the minimum size constraint for the xth dimension. For example, in Table 3, the values of the Min-3 column represent the minimum number of elements that are to be present in the 3rd dimension for each of the closed patterns.

Table 4 Total Running Time in Seconds for CnS-Miner Algorithm and DataPeeler Algorithm

Dimension Size Constraints				Running Time	
Min-4	Min-3	Min-2	Min-1	CnS-Miner	Data Peeler
1	1	1	1	749.004	1043.878
1	1	1	5	740.057	1036.620
1	1	1	10	730.476	1024.398
1	1	1	15	731.860	1009.844
1	1	1	20	729.546	996.654
1	1	1	25	724.764	984.873

Memory efficiency: Both CnS-Miner algorithm and DataPeeler algorithm are highly memory efficient. The space complexity of both algorithms is $O(D+d)$, where d is the depth of the tree (note that both algorithms explores the tree in depth first manner) and D is the input dataset.

Scalability: It is to be noted that both CnS-Miner and DataPeeler algorithm are highly scalable. However, as the size of the dataset increases, the running time of the algorithm also increases. For very large dense datasets, both algorithms will not give the results in reasonable amount of time. The parallelized execution is the only way to get the results in reasonable amount of time and both the algorithms can be easily parallelized to any number of processors without much overhead i.e. the nodes of the tree generated by the algorithms can be assigned to different processors for their parallelized execution.

5. CONCLUSION

We have proposed CnS-Miner algorithm for mining closed n-set patterns from n-ary relations. The recently proposed Data Peeler algorithm uses a binary tree enumeration strategy to generate closed n-set patterns whereas the CnS-Miner algorithm uses an n-ary tree enumeration strategy. Our comprehensive experimental analysis have shown that CnS-Miner algorithm outperforms DataPeeler algorithm when the cardinality of one of the dimensions is large in number whereas DataPeeler algorithm outperforms CnS-Miner algorithm when more than one dimension contain large number of elements.

ACKNOWLEDGMENT

We wish to thank the authors of DataPeeler algorithm, D-Miner algorithm and CubeMiner algorithm for responding to our queries.

6. REFERENCES

[1] Loïc Cerf, Jérémy Besson, Céline Robardet, and Jean-François Boulicaut, "Closed Patterns meet n-ary relations", ACM Transactions on Knowledge Discovery from Data, Vol 3, Issue 1, March 2009. http://doi.acm.org/10.1145/1497577.1497580

[2] J. Besson, C. Robardet, J.F. Boulicaut and S. Rome,"Constraint Based Concept Mining and its Application to Microarray Data Analysis", Journal of Intelligent Data Analysis, pp. 59-82, 2005.

[3] Jinyan Li, Guimei Liu, Haiquan Li, Limsoon Wong, "Maximal Biclique Subgraphs and Closed Pattern Pairs of the Adjacency Matrix: A One-to-One Correspondence and Mining Algorithms," IEEE Transactions on Knowledge and Data Engineering, vol. 19, no. 12, pp. 1625-1637, Dec. 2007

[4] Ji Liping, K.L. Tan and A.K.H. Tung, "Mining Frequent Closed Cubes in 3D datasets," Proc. 32nd int. conference on very large data-bases, 2006

[5] Ji Liping, Kian-Lee Tan,K H. Tung, "Compressed Hierarchical Mining of Frequent Closed Patterns from Dense Data Sets," IEEE Trans. on Knowledge and Data Engineering, Vol 19, No.9, Sept 2007.

[6] T. Uno, T. Asai, Y. Uchida, H. Arimura, "LCM: An Efficient Algorithm for Enumerating Frequent Closed Item Sets," In Proc. IEEE ICDM'03 Workshop FIMI'03, 2003.

[7] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering Frequent Closed Itemsets for Association Rules," Proc. 7th Int. Conf. Database Theory (ICDT'99), pages 398-416, Jan 1999.

[8] G. Grahne, J.Zhu, "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] C. Lucchese,S. Orlando and R. Perego, "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.

[10] G.Liu, "Supporting Efficient and Scalable Frequent Pattern Mining," PhD dissertation, Dept. of Computer Science., Hong Kong University., May 2005.