A Schema Feature Based Frequent Pattern Mining Algorithm for Semi-structured Data Stream

Weiqi Fu[1, a], Husheng Liao[1,b], Xueyun Jin[1,c]

[1]Faculty of Information Technology, Beijing University of Technology, Beijing, China

[a]fwq0609@163.com, [b]liaohs@bjut.edu.cn, [c]jinxueyun@bjut.edu.cn

Keywords: frequent pattern mining, semi-structured data stream, schema feature.

Abstract. Data mining is used to find useful information from massive data. Frequent pattern mining is one important task of data mining. Recently, the researches on frequent pattern mining for semi-structured data have made some progresses, and it also have a lot of focuses for data stream. However, only a few studies focus on both semi-structured data and data stream. This paper proposes an algorithm named *SPrefixTreeISpan*. We segment the semi-structured data stream first, and then uses the pattern-growth method to mine each segment. In the end, we maintain all the results on a structure called *patternTree*. At the same time, the mining algorithm is optimized by the inevitable parent-child relationship and the inevitable child-parent relationship extracted from XML schema. Experiment shows that *SPrefixTreeISpan* has better performance.

Introduction

With the development of information technology, massive data is constantly generated. The analysis of the data is no longer a task that can be completed by manpower. To solve this problem, people developed data mining technology to discover useful information from massive data. This technique is widely concerned because it is automatic and has good performance. Frequent pattern mining is an important task in data mining. Frequent pattern refers to a data fragment that are repeated in the data. And frequent pattern mining refers to finding these frequent patterns from massive data. Many data mining tasks, such as classification analysis, clustering analysis, association rule analysis and prediction, are based on frequent pattern mining. So how to efficiently and accurately mine the frequent patterns, has become the focus of the study.

Previous studies of frequent pattern mining focused on the data from static databases, which is limited and can be reused. However the static databases no longer meet people's requirement. For example, in the fields of network access, financial analysis, sensor networks and medical testing, data is generated in real time, and the data is infinite. Research for mining frequent patterns in data stream has become a new hot spot. The data stream is real-time, ordered, infinite and continuous. Especially because the stream is infinite and the physical memory is limited, the data can't be permanently stored. The data can only be scanned in limited time. Because the data in the stream can't be reused, the traditional mining algorithm is no longer suitable. How to efficiently mine frequent patterns from data stream in real time with limited memory has become the main focus in the research.

On the other hand, there are many types of data, such as transaction data, sequence data, structured data, semi-structured data, etc. Researches on mining these types of data are in progress. Among them, semi-structured data is widely used in many fields because of the special tree structure, such as XML data. XML as an extensible markup language is used for the data transmission and storage. XML has become the standard of data exchange. XML is widely used in the stock market, network monitoring, e-commerce, database storage and other fields. Therefore, the study of semi-structured data such as XML becomes a hot topic. And the tree structure is the main research problem.

At present, researches on frequent pattern mining for semi-structured data have made some progresses, and researches on frequent pattern mining for data stream also have a lot of focuses. However, only a few studies focus on both semi-structured data and stream data. In fact, XML data usually transmits in data stream. Therefore how to mine semi-structured data stream need a study.

This paper focuses on the study of frequent pattern mining algorithm for XML stream. Mining algorithms for semi-structured data are mainly Apriori-like algorithms, algorithms based on enumerated tree and algorithms based on pattern-growth. Apriori-like algorithms and algorithms based on enumerated tree need to generate and test candidate sets, which is the main cost of these two types of algorithms. Algorithms based on the pattern-growth don't need to generate candidate sets, the mining efficiency is better than other two types of algorithms. On the other hand, the structure of XML data is defined by the XML schema. Therefore, using the feature extracted from schema can reduce the mining steps. Based on the pattern-growth algorithm *PrefixISpan* proposed by L. Zou [1], this paper proposes a frequent pattern mining algorithm for mining semi-structured data stream and being optimized by schema feature.

The contribution is as follows:

1) Based on *PrefixISpan*, we propose a frequent pattern mining algorithm for semi-structured data stream named *SPrefixTreeISpan*;

2) Based on the feature extracted from schema, we optimize our algorithm;

3) We propose a segmentation strategy for XML stream.

The structure of this paper is as follows: the second section introduces the current researches on mining semi-structured data, mining data stream and schema feature; the third section introduces the design idea of *SPrefixTreeISpan*; the fourth section gives the implementation; the fifth section gives the experimental results; the sixth section gives a conclusion.

Related Works

Researches on Mining Semi-structured Data. The mining algorithms can be divided into three categories. The first type of algorithm is Apriori-like algorithm. The *Apriori* algorithm was proposed by R. Agrawal [2]. Any super itemsets of an infrequent itemset must also be infrequent. Based on this theory, the *Apriori* algorithm combines frequent k-length itemsets to generate (k+1)-length candidate itemsets, and tests these itemsets to find (k+1)-length frequent itemsets. The second type of algorithm is algorithm based on enumerated tree. Although the Apriori-like algorithm is widely used in mining tasks, the efficiency will significantly reduce as the length of the frequent itemsets increasing. R.J. Bayardo proposed the enumerated tree [3] which can enumerate all candidate sets without repetition and proposed an algorithm called *Max-Miner* which can efficiently mine long itemsets. The third type of algorithm is pattern-growth algorithm. The pattern-growth method was proposed by J. Han [4]. The method finds the 1-length frequent patterns first, then finds the 2-length frequent patterns, and then finds all frequent patterns recursively. The pattern-growth algorithm doesn't need to generate and test candidate sets, the efficiency of mining is improved.

In the studies of mining semi-structured Data. C. Wang proposed *Chopper and XSpanner* algorithm [5] which uses pattern-growth method to mine semi-structured data. Based on this algorithm, L. Zou proposed *PrefixTreeESpan* algorithm [6] and *PrefixTreeISpan* algorithm. The *PrefixTreeESpan* algorithm can find all the frequent embedded subtree patterns which focuses on the ancestor-descendant relationship in XML data. And the *PrefixTreeISpan* algorithm can find all the frequent induced subtree patterns which focuses on the parent-child relationship in XML data.

Since the pattern-growth algorithm doesn't need to generate and test candidate sets, it is more efficient than the other two types of algorithms. However, the proposed algorithms can’t be used to mine semi-structured data stream.

Researches on Mining Data Stream. In the studies of mining data stream, C. Giannella proposed *FP-stream* algorithm [7], which uses a tilting time window to mine transaction data. S.B. Naik proposed *QMINE* algorithm [8], which can incremental mine closure frequent subsets from data stream. In the case of sequential data stream, C. Ho proposed *IncSPAM* algorithm [9] which can mine sequence data stream over a sliding window. In the case of structured data stream, Leung proposed a structure called *DS-matrix* [10] which can mine frequent subgraph patterns from data stream. For semi-structured data stream, X. Lei proposed a paging data stream mining model *Tmlist* [11], which can mine frequent embedded subtree patterns from XML stream.

At present, most of the mining algorithms can't mine semi-structured data stream. *Tmlist* model can mine the frequent embedded subtree patterns from the XML stream. But *Tmlist* needs to build a coding table, then records and maintains the relationship between nodes. In this paper, we use pattern-growth method to mine XML stream, and our algorithm will be compared with *Tmlist*.

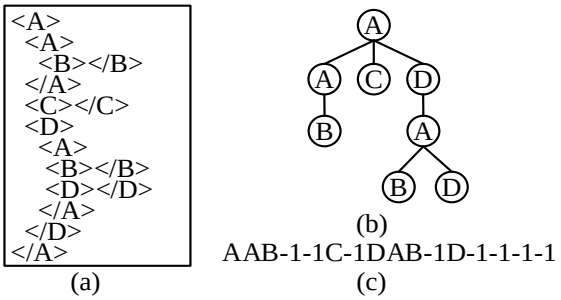
Researches on XML Schema. XML schema is used to describe the XML data structure. In the studies of XML, schema features are extracted to optimize XML query. S. Paparizos proposed a schema information structure [12] to store the features contained in the schema. K. Liu proposed a XML Schema feature extraction method based on model checking [13], which can be used to find the inevitable parent-child and the inevitable ancestor-descendant relationship. Subsequently, K. Liu introduced the extended CTL formula [14] for the same purpose.

At present, in the studies of mining XML, there are no algorithms using XML schema information to optimize the mining. By using the inevitable child-parent relationship and the inevitable parent-child relationship extracted from the XML schema, mining steps can be reduced and mining efficiency can be improved. In this paper, our algorithm is optimized by using schema feature.

Design Idea

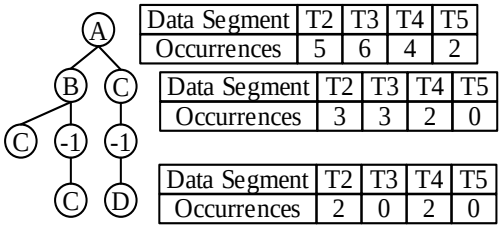
Our algorithm focuses on mining frequent induced subtree patterns from XML stream, which is the parent-child relationship in XML data. The algorithm uses the sliding time window model to maintain the XML stream. The time window contains k segments of data. The length of each segment is equal and each segment is tagged by time. Whenever a new data segment flows into the time window, the oldest data segment is dropped. Each segment of XML data will be converted to sequence data, and then be mined by pattern-growth method. During the mining, we use the inevitable parent-child relationship and the inevitable child-parent relationship extracted from XML schema to optimize the process, improving the efficiency of mining. In the end, all the frequent patterns of k data segments are merged into a structure called *patternTree*. By updating and maintaining the *patternTree*, we can get the frequent patterns of current time window.

Data Stream Segmentation. In order to avoid remining entire data when new data coming, the data stream will be divided into multiple segment data. And then each segment will be processed alone. When segmenting data, it is possible to divide a complete XML tree into two parts. On the one hand, the structure of the XML data is broken, so it needs to be fixed in the process. On the other hand, since the first part of the data is processed, the second part of the data has not yet arrived. So the mining result of current data segment is not accurate. To solve this problem, we assume that the second part of the data could be any possible data, and get some inaccurate frequent patterns. These inaccurate patterns are also stored in the *patternTree*. And when the second part of the data arrives, these inaccurate patterns will be checked to ensure the accuracy of the mining results.



(a) (b) (c)

Fig. 1 XML document, XML tree and XML sequence



Data Segment	T2	T3	T4	T5
Occurrences	5	6	4	2

Data Segment	T2	T3	T4	T5
Occurrences	3	3	2	0

Data Segment	T2	T3	T4	T5
Occurrences	2	0	2	0

Fig. 2 patternTree

Data Conversion. During the frequent pattern mining, the XML data will be converted into sequence data, which can effectively reduce the use of memory. Also when updating the *patternTree*, the sequence mining results will be needed. At first, the XML data will be read by the order of the XML stream. When getting a start element, we will keep the element label. And if getting the end element, we will replace it with label "-1". Fig. 1 (a) is an element in an XML data stream. Fig. 1 (b) is the corresponding tree structure, and the converted result is shown in Fig. 1 (c). The converted result is "AAB-1-1C-1DAB-1D-1-1 -1-1 ". The sequence preserves the structure of the tree and can

be restored to the original XML. Since the "-1" at the end of the sequence does not affect the structure, the sequence can be simplified as "AAB-1-1C-1DAB-1D".

Mining by Pattern-growth Method. Our algorithm *SPrefixTreeISpan* is used to mine frequent induced subtree patterns. For a tree set T with a node set V and a edge set E, if a tree T' and a node set V' and a edge set E' satisfy 1) $V' \subseteq V$; (2) for each edge $(v1',v2') \in E'$ in T', there is a edge $(v1,v2) \in E$ corresponding to T' in T, then we can say T' is a induced subtree of T.

SPrefixTreeISpan is based on the *PrefixTreeISpan* algorithm. At first, we count the number of elements and find out all the 1-length frequent subtree patterns. Then we treat each subtree pattern as a prefix tree and construct the subdatabase by suffix subtrees which have the same prefix tree. Then we mine the subdatabase. For example, as shown in Fig. 1, the suffix subtrees of pattern "A" are "AB", "C", "DAB-1D", "B" and "D". The suffix subtrees of pattern "AA" are "B", "C" and "DAB-1D". The suffix subtree of the pattern "AD" is "AB-1D". After constructing the suffix subtree database, the subdatabase will be processed in the same way recursively until all the frequent patterns are found.

Optimized by Schema Feature. XML Schema is used to describe XML data structure. By analyzing the XML schema, we can get features of XML data. The features we needed in this paper are the inevitable parent-child relationship (IPC) and the inevitable child-parent relationship (ICP). IPC means that each A element must has a B element as child. ICP means that each B element must has an A element as parent. For IPC, during the mining, if we find pattern "A" is frequent, then pattern "AB" must be frequent. And pattern "AB" and pattern "A" have the same number of occurrences. For ICP, if pattern "A" and pattern "B" are frequent, since the B element must have A element as parent, mining result of the sub-database of pattern "A" must include all the frequent patterns with a prefix pattern "B". Therefore, there is no need to constructor subdatabase of pattern "B". By using these features, the mining steps can be reduced and the mining efficiency can be improved.

Constructing the patternTree. The *patternTree* contains all converted frequent subtree patterns. Each *patternTree* node refers to a converted frequent pattern which represented by the nodes on the path from the root node to this node. Each node on the tree has a table that stores the number of occurrences of the pattern referred. When new data segment arrives, we have to update and maintain the *patternTree*. Fig. 2 shows a *patternTree*. This *patternTree* consists of frequent patterns "ABC", "AB-1C" and "AC-1D". Take "AC-1D" as an example, every node has a table contains the frequency of the pattern "A", "AC", and "AC-1D" in four data segments in the time window. T2, T3, T4 and T5 represent the id of four data segments. If a new frequent pattern is found, adding a new node to the *patternTree*. If an old pattern is no longer frequent, deleting the corresponding node and the descendant nodes on the *patternTree*. By traversing the *patternTree*, we can get the final result.

SPrefixTreeISpan Algorithm

SPrefixTreeISpan needs to read the corresponding XML schema at first, extract the necessary features and then mine the XML data.

Extracting Schema Features Algorithm. We only need to extract the relationship IPC and ICP. When extracting, we scan the schema once. At the same time, we find the elements with same label and record the parent elements and the child elements.

For IPC, in the schema, if the A element has only one child B element or the type of children elements is sequence which is defined by schema, then the A element and the B element or these child elements are IPC. For ICP, in the schema, if the A element has only one parent B element, then the A element and the B element are ICP.

```
Algorithm: GetSchemaFeature
Input: XML schema
Output: ICP childParentMap, IPC parentChildrenMap
1  construct elementStack
2  for(read schema data)
3     if get start element
4        find parent in elementStack, record PC releationship in PCMap, push elementStack
5     if get end element
```

```
6        pop elementStack
7  traverses PCMap
8     if IPC, insert into parentChildrenMap
9     if ICP, insert into childParentMap
```

The algorithm *GetSchemaFeature* can extract IPC and ICP from the schema. By reading the schema once, *GetSchemaFeature* records all the parent-child relationship in *PCMap*. Then *GetSchemaFeature* traverses *PCMap*. If an A element has only one child B element or a sequence type of child elements, then the relationship between A element and B element will be stored in *parentChildrenMap*. If an A element has only one parent B element, then the relationship between A element and B element will be stored in *childParentMap.*

Pattern-growth Mining Algorithm. After extracting the schema features, we need to mine each data segment. Our algorithm is based on the *PrefixTreeISpan* algorithm, and can be used to mine XML stream. Moreover, our algorithm is optimized by the schema features.

```
Algorithm: SPrefixTreeISpan
Input: sequenceList(XML data), minSupport, patternTree, parentChildrenMap, childParentMap, checkStack
Output: updated patternTree
1  traverse sequenceList
2     if occurrences of element > minSupport, insert element into freqElems
3     if checkElement, push checkStack
4  find childParentMap, remove elements have ICP from freqElems
5  for(each e in freqElems)
6     update e and the count of e on patternTree
7     push e into currentPattern
8     traverse sequenceList, find the positions of e
9     Fre(sequenceList, minSupport, positions, currentPattern, count, 1, parentChildrenMap)
Function: Fre
Input：XML data seqList, minSup, pos, currentPattern, patternCount, patternDepth, parentChildrenMap
Output: current result
1  find parentChildrenMap
2  for(each p to be grown in currentPattern)
3     if p has IPC
4        insert the IPC children into freqElems
5     else
6        traverse seqList by pos,  insert frequent elements into freqElems
7        if checkElment, push checkStack
8     if  freqElems is not null
9       for(each e in freqElems)
10         update e and the count of e on patternTree
11         push e into currentPattern
12         traverse seqList, find the positions pos of e
13         Fre(seqList, minSup, pos, pattern, count, patternDepth+1, parentChildrenMap)
14 push " -1" into currentPattern
15 --patternDepth
```

The algorithm *SPrefixTreeISpan* uses the method of pattern-growth. It finds out all 1-length frequent patterns, and then calls function *Fre* to mine each pattern. When calling *Fre,* it is necessary to process each extensible element of the current pattern. *Fre* will find all the frequent child elements, and then call function *Fre* recursively. When mining, if finding current element is incompleted, which means there is no "-1" correspond to this element, this element has to be recorded in *checkStack.*

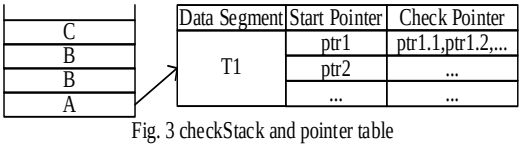


Fig. 3 checkStack and pointer table

Segmentation Algorithm. According to the characteristics of XML data, each element has a start element and the corresponding end element. When segmenting, it may broke the structure and cause one or more start elements to have no corresponding end elements. These elements are called check

elements. All the check elements will be pushed into a structure called *checkStack*. Each check element may produce multiple inaccurate patterns during the mining. These inaccurate patterns will be stored on the *patternTree*. And each check element has a pointer table pointing to these inaccurate patterns. As shown in Fig. 3, the check element A has a pointer table. T1 means that this check element is from data segment T1. The start pointer points to the positions of the check element A on the pattern tree. And the check pointer points to the elements needed to be checked on the pattern tree.

During mining, the check elements and positions of inaccurate patterns will be found and stored in the *checkStack*. When new data segment arrives, the algorithm will check these patterns to make sure the result is correctly.

```
Algorithm: CheckCount
Input: new segment sequenceList, checkStack, patternTree
Output: updated patternTree
1   traverse checkStack
2      find patternTree node n by start pointer
3      get children of n
4      for(each child of n)
5         match child and elements of sequenceList
6         if  ismatch
7            if check pointer points to child
8            ++count of child
9            check pointer points to the children of child
10       check the new children
```

The algorithm *CheckCount* checks the nodes on the *patternTree* by the information recorded in the *checkStack*. The check element node on the *patternTree* is found by the start pointer. The descendant nodes of the check element node need to match the new data, and only the node pointed by the check pointer need to be checked. Then the check pointer will point to the children of the node.

Experiment

This paper implements our algorithm and modified *Tmlist* [11], which can mine induced subtree patterns now, by using C++ standard library and STL. The experimental environment is 3.40GHz CPU, 8GB memory, Win7 operating system. The schema used by the experiment is converted from the DTD set in Niagara's experimental data [15]. And the data used is automatically generated by this schema.

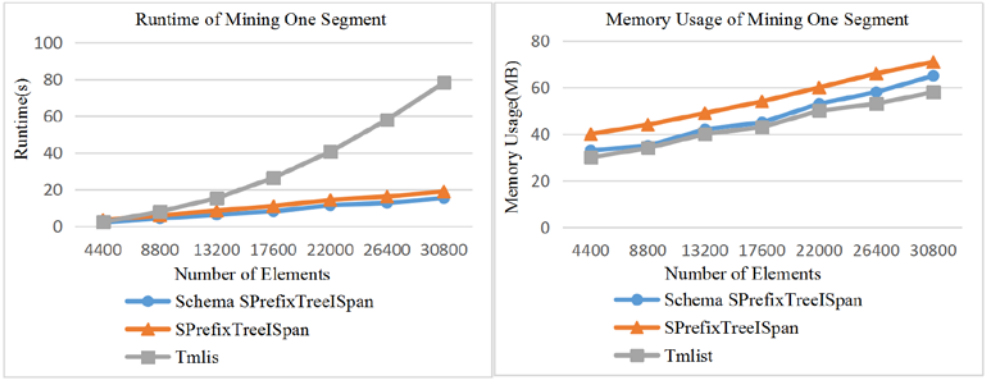


Fig. 4 Runtime and memory usage of mining different segment

Experiment compares *SPrefixTreeISpan* algorithm without schema, *SPrefixTreeISpan* algorithm with schema and *Tmlist*. Mining seven different sizes of data segments by these algorithms. As the sizes of the data segments are different, the minsupport is 3% of the number of elements. The experimental results are shown in Fig. 4. For the runtime, our algorithm is better than *Tmlist* whether it is optimized by schema feature or not. And after optimized by schema feature, the runtime is reduced. For the memory usage, *Tmlist* takes less memory than our algorithm. But after optimized by schema feature, the algorithm's memory usage is significantly reduced.

Conclusion

This paper proposes a frequent pattern mining algorithm for semi-structured data stream based on schema feature called *SPrefixTreeISpan*, which segments the XML stream, mines each data segment by pattern-growth method, and optimizes the mining algorithm by the relationship ICP and IPC extracted from schema. And *SPrefixTreeISpan* uses the *patternTree* to maintain the mining results, and gets the frequent induced subtree patterns by traversing the *patternTree*.

In the future work, we will study how to make more efficient use of schema. We not only can extract the inevitable child-parent and parent-child relationship from schema, but also can extract more features to be used.

Acknowledgements

This work was financially supported by the Beijing Natural Science Foundation (4122011) and National Natural Science Foundation of China (61202074).

References

[1] L. Zou, Y. Lu, et al., "Mining Frequent Induced Subtrees by Prefix-Tree-Projected Pattern Growth", in: Web-age Information Management Workshop (2006), pages 18-18.

[2] R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association Rules", in: Proceeding of 20th Very Large Data Bases Conference(1994), pages 487-499.

[3] R.J. Bayardo, "Efficiently Mining Long Patterns from Databases", in International Conference on Management of Data (1998), pages 85-93.

[4] J. Han, J. J. Pei, Y. Yin, "Mining Frequent Patterns without Candidate Generation", in: International Conference on Management of Data (2000), pages 1-12.

[5] C. Wang, M. Hong, et al., "Efficient Pattern-Growth Methods for Frequent Tree Pattern Mining" in: Pacific Asia Conference on Knowledge Discovery and Data Mining (2004), pages 441-451.

[6] L. Zou, Y. Lu, et al., "PrefixTreeESpan: A Pattern Growth Algorithm for Mining Embedded Subtrees", in: Web Information Systems Engineering (2006), pages 499-505.

[7] C. Giannella, J. Han, et al., "Mining Frequent Patterns in Data Streams at Multiple Time Granularities", in: Data Mining: Next Generation Challenges and Future Directions (2004), chapter 6, AAAI/MIT Press.

[8] S.B. Naik, J.D. Pawar, "A Quick Algorithm for Incremental Mining Closed Frequent Itemsets over Data Streams" in: the Second ACM IKDD Conference (2015), pages 126-127.

[9] C. Ho, H. Li, et al., "Incremental Mining of Sequential Patterns over a Stream Sliding Window", in: International Conference on Data Mining (2006), pages 677-681.

[10] C.K. Leung, A. Cuzzocrea, "Frequent Subgraph Mining from Streams of Uncertain Data", in: Computer Science and Software Engineering (2015), pages 18-27.

[11] X. Lei, Z. Yang, et al., "Mining Frequent Subtree on Paging XML Data Stream" in Chinese, in: Journal of Computer Research and Development (2012), pages 1926-1936.

[12] S. Paparizos, J.M. Patel, H.V. Jagadish, "SIGOPT: Using Schema to Optimize XML Query Processing", in: International Conference on Data Engineering (2007), pages 1456-1460.

[13] K. Liu, H. Yang, et al., "XML Schema Constraints Extraction Based on Model Checking" in Chinese, in: The China Computer Federation (2012), pages 160-164.

[14] K. Liu, H. Yang, et al., "XML Schema Features Extraction Algorithm" in Chinese, in: Computer Science (2015), pages 438-443.

[15] Information on http://research.cs.wisc.edu/niagara/ .