The Pennsylvania State University

The Graduate School

Capital College

MINING ASSOCIATION RULES FROM XML DATA USING A VERTICAL FORMAT

A Master's Paper in

Computer Science

by

Holly L. Trego

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Computer Science

December 2004

Abstract

XML documents are increasingly being used to store information for a variety of reasons. When XML documents are used to represent a market-basket database, often the description of the elements are embedded as attributes of the items being purchased. This makes it unique from traditional market-basket analysis because characteristics about the items being purchased can be stored and accessed easily. The majority of the existing association rule algorithms that deal with XML data do not take into account the attributes of the items. We present here a new vertical association rule mining algorithm for XML data called XMLVMINE. XMLVMINE is an algorithm that processes market-basket style XML documents to find association rules pertaining to both the elements and the attributes of the elements. A tree structure is used to store the candidates for the rules. The candidates are shown to the user so that some of the candidates can be removed at the beginning of the algorithm. This approach fits well with an interface that uses a document type definition (DTD), because DTDs representing market basket data can easily be converted into a tree. This algorithm also uses vertical data mining techniques that perform better than the traditional horizontal approach. Our experimental results show that items along with the attributes of items in an XML file can be efficiently mined in a manner similar to the existing Apriori style algorithms.

Table of Contents

Acknowledgements

I am very grateful for the support and advice of my project advisor, Dr. Qin Ding, for her continuous support and for keeping me focused on the research. Her insights into the topic were the key to the success of this paper. The comments given by the rest of the team, Dr. Thang N. Bui, Dr. Sukmoon Chang, Dr. Pavel Naumov, and Dr. Linda Null were greatly appreciated. The support of my husband, Terrance Trego, during the past several years was also important in the success of my studies at the Pennsylvania State University.

List of Figures

List of Tables

1. Introduction

There have been many algorithms [1,2,5,6,10,13,14,17] that have shown the need for efficient mining of market-basket databases. An example of market-basket style data is a set of supermarket transactions. Each transaction represents a sale that was made and within the transaction there are any number of items that were purchased. It is now very easy to store a wide variety of information in large quantities. The challenge has shifted from gathering the information to being able to use the information in a meaningful manner. The results of mining market-basket data can be used when making many marketing decisions. These decisions range from what items can be sold near each other to what items can be marketed to certain individuals. The transactions can be mined for either a particular individual to try to predict future behavior, or for an entire population, thus predicting customers' behaviors. The most notable algorithm for mining association rules is called VIPER. This algorithm is described in more detail in section 2.3 (Vertical Mining). Many of the topics discussed in the VIPER algorithm were used in our algorithm, specifically the idea of using a vertical format. The VIPER algorithm was not written for XML documents and it did not examine attributes of items being purchased.

There have also been a number of documents [7,8,9,11,18] that discuss mining association rules from XML databases using association rules. There has been little research on mining market-basket style data that includes attributes of the items that are being mined, for example we may want to determine what types of items are purchased together. There has also been little research into the area of connecting the mining process to the DTD. Therefore we have decided to focus on these aspects of

mining XML data. We have decided to use a vertical approach, as this has been shown to have better performance than the classic horizontal approach. The vertical approach to mining association rules was discussed in detail in [3], but was limited to sets of items and did not discuss attributes of those items. This approach was used as a starting point for the algorithm and implementation discussed in this paper. As was mentioned in [3], the vertical approach makes a logical choice for mining association rules because the goal is to discover correlated items. This approach also allows for vertical partitioning of the data, reducing effective database size, and compact storage of the database. We have also decided to use a tree structure to represent the 1-item candidates because this allows us to tie the candidates to the document type definition, which in turn allows us to give the user more detailed options as to what candidates the user wants to include in the rules.

The rest of the paper is organized as follows: The background section gives a brief introduction on XML Data, association rule mining, vertical mining, and the current attempts at mining XML data. The next section gives details on the proposed algorithm (XMLVMINE) along with a detailed example. This section also contains an evaluation on the performance of the proposed algorithm. The final section is a conclusion that includes the successes of the algorithm along with those items that should be included in future research.

2. Background

2.1 XML Data

XML is the Extensible Markup Language, a subset of the Standard Generalized Markup Language (SGML). In XML, documents and the elements that they contain can have whatever structure and meaning their creator wants them to have, because these definitions are up to the user. In effect, users can define their own standard for the structure of a document. This may not seem to be a good idea, but the situation is redeemed by the Document Type Definition (DTD). The DTD defines the grammar for the XML application and describes the structure that the elements must adhere to in order for the document to be a valid instance of that application. The

XML Document:

```
<transactions>
  <transaction id="1">
     <name >Holly L. Trego</name>
     <dateofpurchase>10/12/04</dateofpurchase>
     <items>
        <item color="Blue" size = "6" thickness = "thin"> Plates</item>
        <item color="Blue" size = "9" thickness = "thick"> Cups</item>
        <item> Streamers</item>
     </items>
  </transaction>
</transactions>
```

DTD:

```
<!ELEMENT transactions (transaction) >
<!ELEMENT transaction(name, dateofpurchase, items)>
<!ELEMENT items(item)>
<!ATTLIST item color (Blue|Red|Green)>
<!ATTLIST item size(6|9|12)>
<!ATTLIST item thickness(thin|medium|thick)>
```

Figure 1: A simple XML document with a document type definition

main advantage of DTDs is that they can be shared, and so if a DTD is available for a certain type of data, all valid XML files using that DTD can be understood by everyone [16]. This allows various data sources to be shared easily between entities if they agree on the structure of the data. An example of an XML file along with a DTD describing that document is shown in Figure 1.

One of the main goals of the XML development workgroup at the World Wide Web Consortium (W3C) was to standardize the syntax of XML using a strict set of rules. More information on the W3C workgroup and XML standards can currently be found at www.w3c.org. In XML it is vital that every element and attribute used in XML, along with the other instructions that are required to build a valid XML document, follows the syntax rules precisely. In most situations, this even extends to case sensitivity. This helps when mining the information because it is likely that the information is in a standard format.

The building blocks of XML documents are: Elements, Tags, Attributes, Entities, PCDATA, and CDATA [16]. Elements are the main building blocks of XML documents. Tags are used to markup elements. A starting tag like <element_name> mark up the beginning of an element, and an ending tag like </element_name> mark up the end of an element. Attributes provide extra information about elements. Attributes are placed inside the start tag of an element. Attributes come in name/value parts. An example of an element with an attribute is <item color="Red">Plate. Entities are variables used to define common text. Entity references are references to entities. The following entities are predefined in XML documents: &lt (<), &gt (>), &amp (&), &quot ("), and &apos ('). PCDATA means

parsed character data. Character data is the data found between the start and end tag of an element. Tags inside the text are treated as markup and entities are expanded. CDATA also means character data. CDATA is text that is not parsed by a parser. Tags inside the text are not treated as markup and entities are not expanded.

The Document Object Model (DOM) is an in-memory representation of an XML document. The DOM allows you to programmatically read, manipulate, and modify an XML document. In our implementation, the DOM is used to process the file because it is similar to the tree structure that we create for the candidates. The DOM contains the entire structure of the document, whereas our candidate tree removes all duplicate entries and creates a link from each node in the candidate tree to an object which contains more details on the object. As XML is read into memory, nodes in the DOM are created. These nodes are then accessed when processing the file and determining what should be added to the candidate tree. The DOM is most useful for reading XML data into memory to change its structure, to add or remove nodes, or to modify the data in the XML file. The W3C has also written specifications for DOM that can be found at the W3C website (www.w3c.org/DOM).

The W3C has also created a standard for accessing information in an XML document. The language for this is called XQuery. In the XQuery language an XPath expression is used to find a subset of data within the file. This is useful when a file has additional information other than the market-basket data. There are two major documents that give in detail the specifications for both XQuery (“XQuery 1.0: An XML Query Language”) and XPath (“XML Path Language (XPath) 2.0)”. Both of these documents can be found on the W3C website (www.w3c.org).

For the XML document described in Figure 1, the information of interest (the items being purchased) can be accessed using the XPath expression of "//transactions/transaction/items". The results of using this expression to access the document is the same as mining information for the document shown in Figure 2 with an XPath expression of "//". Using the XPath expressions allows us to access a subset of information in a document which is often very useful. This is useful in that it allows for less pre-processing of the document that is being examined.

```
<items>
  <item color="Blue" size = "6" thickness = "thin"> Plates</item>
  <item color="Blue" size = "9" thickness = "thick"> Cups</item>
  <item> Streamers</item>
</items>
```

Figure 2: A subset of an XML document

2.2 Association Rule Mining

The idea of generating association rules was first presented in [1]. Association technique is often applied to market-basket analysis, where it uses point-of-sales transaction data to identify product affinities [12]. There are a number of papers that describe association rule mining each with a slightly different approach. A broad way of looking at association rule mining is by breaking it down into four main steps; context identification, context selection, rule selection and result construction [4]. This shows the problem of mining association rules, beginning with the task of identifying the data that is going to be mined (context identification), and the subset of the data that is going to be mined that is of interest (context selection). The rule selection portion of the process defines the constraints that are used when determining the rules, and finally result construction focuses on how the generated rules are

represented. The portion of this process that is of the most interest is the rule selection process. During this step all of the possible candidates are created. These candidates are combinations of items in the database that are present in a given number of transactions. Rules are then generated from these candidates. The rules take each candidate with more than one item and creates all subsets that are less than or equal to k-1 (where k is the number of items in the candidate) to create the antecedent to the rule. The consequent of the rule becomes those items not in the antecedent for the given candidate [1]. The goal of association rule mining is to find objective measures of interesting patterns. The two measures of interesting patterns that are used most often in association rule mining are support and confidence. The support of a rule represents the percentage of transactions that a given rule satisfies. The support of a rule is the probability that both X and Y, where X is the antecedent and Y is the consequent of the rule, are contained in a transaction in the data that is being mined. The confidence of a rule assesses the certainty of a rule. This is taken to be the probability that a transaction containing X also contains Y [9]. This is normally expressed as the following: $\text{support}(X \Rightarrow Y) = P(X \cup Y)$, $\text{confidence}(X \Rightarrow Y) = P(Y|X)$. A support of 2% means that 2% of all the transactions under analysis contain the item of interest. A confidence of 60% means that 60% of the customers that bought the first item of interest, bought the second item of interest. Typically, association rules are considered interesting if they satisfy both a minimum support threshold and a minimum confidence threshold.

There have been a number of approaches to mining association rules from databases. The most notable being the Apriori algorithm. This algorithm was first

introduced in 1993 by Agrawal, Imielinski, and Swami [1]. There have since been a number of algorithms that have attempted to improve the performance of the original algorithm. The main contribution of the algorithm was that it used passes in generating candidates for the rules, where each pass was determined by the previous pass. This algorithm noted that all subsets of a candidate must also be a candidate, so the candidates could be created by combining candidates at a lower level. For example, if both (Plates, Streamers) and (Plates, Cups) have enough support then (Plates, Streamers, Cups) should be examined to determine if it also has enough support.

2.3 Vertical Mining

Vertical mining is a technique that is used to mine information from a database. The traditional approach, horizontal mining, stores each transaction with an array of bits where each bit represents an item in the database. In the vertical approach, an array of bits is stored for each item.

Horizontal Mining

	Items				
TID	1	2	3	4	5
1	0	1	1	0	1
2	0	0	1	1	1
3	1	1	0	0	1
4	0	0	1	1	1
5	1	1	1	0	0

Vertical Mining

	Items				
TID	1	2	3	4	5
1	0	1	1	0	1
2	0	0	1	1	1
3	1	1	0	0	1
4	0	0	1	1	1
5	1	1	1	0	0

Figure 3: Vertical and horizontal mining

An algorithm (VIPER) for Vertical Mining of Large Databases was the first algorithm to discuss using a vertical approach to mining association rules [3]. This

was used as a starting point for the algorithm that is proposed in this paper. The VIPER algorithm details some of the advantages of using the vertical format over the horizontal format. The advantages of mining databases with a vertical representation include the ability to easily compute the support of itemsets because it involves only the intersection of the lists being evaluated for support. In horizontal mining, very complex structures, such as hash trees and functions, must be used, which not only take more space, but also time to construct and evaluate. Another advantage to vertical mining is that the database size can be reduced with each scan. Irrelevant information is not accessed as it is in the horizontal format. Another reason that vertical mining is more efficient is because of the ability to compress a vertical database. The number of items in a transaction is normally much smaller than the number of transactions. By representing an item as a list there is a much greater chance that the list can be compressed. The list below summarizes the advantages of the vertical mining approach [3].

- More natural choice for discovering correlated items
- Fast and simple support counting
- Asynchrony in the counting process
- Reduce the effective database size
- Compact storage of the database
- Better support of dynamic databases

An example of how the vertical mining approach works is illustrated in Figure 4. It is apparent that the length of the vertical bit arrays is equal to the maximum transaction id where the item was found. To determine if two items can be combined

into a single candidate, the vertical bit arrays are intersected and the count of 1's in the resulting bit array determine the support of the combined items. This approach is successful when an item cannot be found more than once in a transaction with different attributes. Figure 4 shows the difficulty that results with this situation, by simply having one position for a transaction; we may not know that Blue 9" plates were never purchased. The solution to this scenario is detailed in the proposed algorithm.

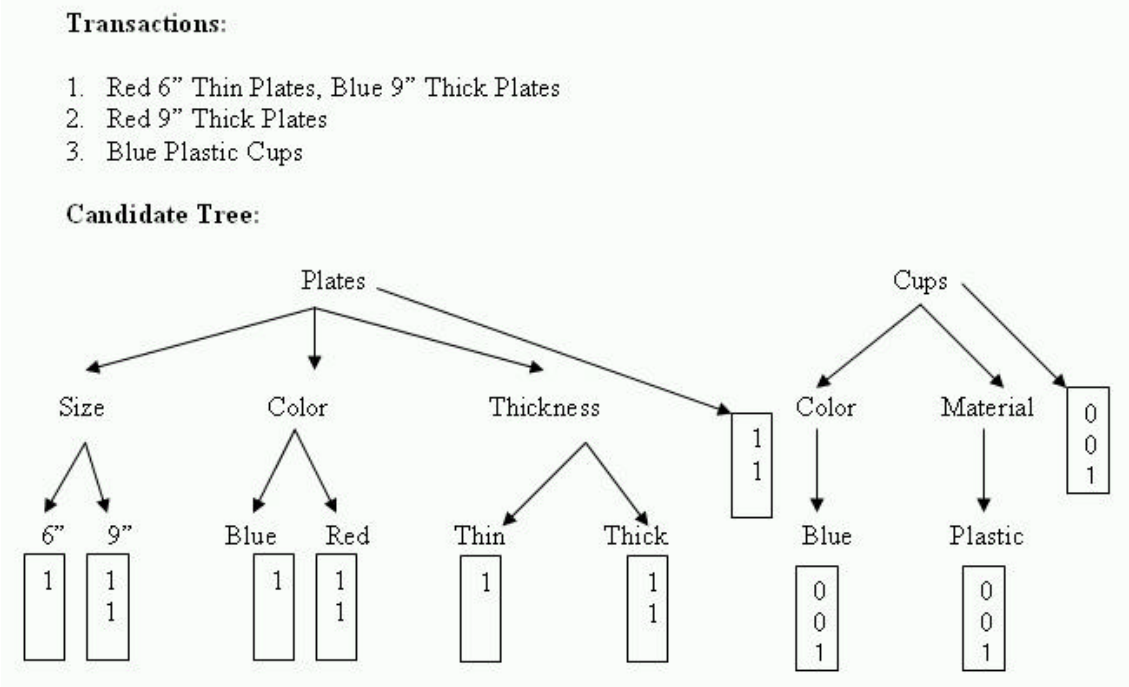


Figure 4: The vertical approach to mining multi-level data.

2.4 Current XML Mining Algorithms

There have been a number of algorithms that have focused on mining XML data. One of the first algorithms dealing with XML introduced an extension of XQuery for mining association rules [8]. In this application the hierarchical XML data was able to be mined without first transforming it to a "flat" relational model.

The goal of the research was to extract association rules that describe interesting relationships among fragments, of the source XML document. This algorithm was based on a previous non-XML algorithm called MINE RULE [5]. The new algorithm had similar syntax in that there was a source, root, body, head along with a where clause and a definition of the criteria. This algorithm exposed some of the benefits of XML, but still did not take into account the idea of items having attributes. This approach was later referred to as the XMINE operator. This type of approach of mining XML documents using XQuery was also researched with the goal in mind to eliminate any pre- and post-processing of the XML documents [11]. One approach was to use XQuery expressions. Figure 5 is an example of how the XQuery language can be used to mine XML data [11]. This algorithm is similar to the XMINE operator, in that it allows subsets of information in the file to be mined.

```
let $src:=document("/transactions.xml")//items
let $minsup:=0.4
let $total:=count($src)*1.00
let $C:=distinct-values($src/*)
let $l:=(for $itemset in $C
        let $item:=(for $item in $src/*
                where $itemset = $item
                return $item)
        let $sup:=(count($items)*1.00) div $total
        where $sup >=$minsup
        return <largeitemset>
                <items> {$itemset}</items>
                <support>{$sup}</support>
                </largeitemset>
let $L:=$l
return   <largeItemset> {Apriori($l, $L, $minsup, $total, $src)}</largeItemset>
```

Figure 5: Mining XML files using XQuery

Another algorithm to data mine XML data called TreeFinder was introduced to find frequent trees in the XML document [18]. The Apriori approach is used and

the results are trees, such that their exact or perturbed copies are frequent in a collection of labeled trees. This algorithm works well with semi-structured data, where an exact path to an item cannot be specified exactly. For example, the trees shown in Figure 6 can be mined to determine the frequent sub-trees. This algorithm

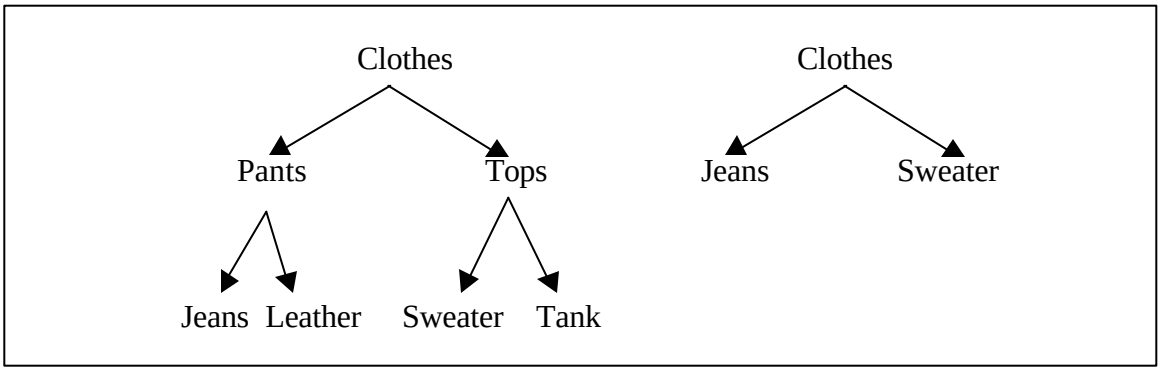


Figure 6: A variety of tree structures

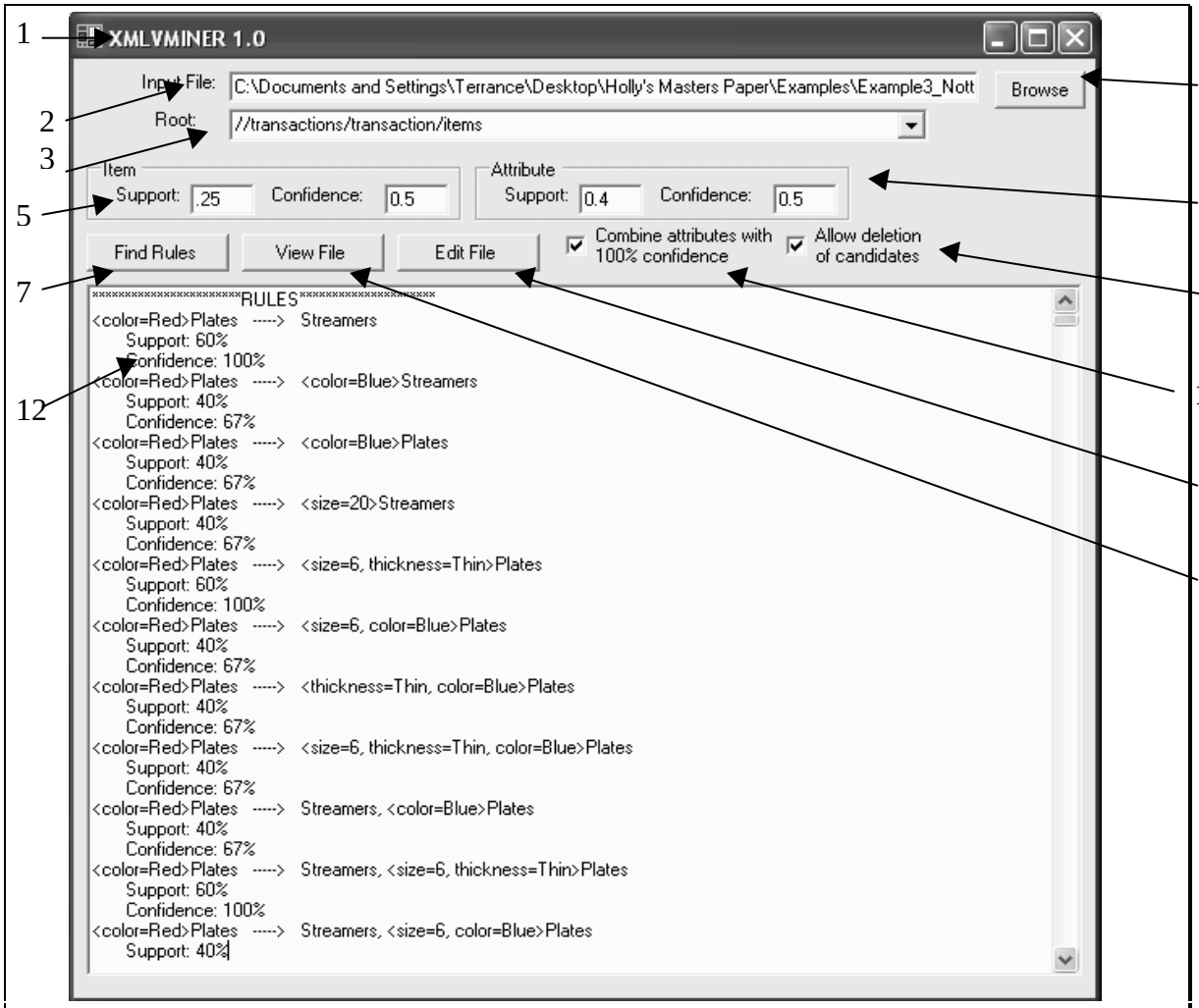
finds Clothes -> Jeans in both trees and can determine that this might be a frequent tree. This algorithm can be helpful if combined with our proposed algorithm to allow us to find frequent items by including all of the similar trees. The idea of semi-structured data is not discussed in this paper, but it is an item of interest that needs to be examined in conjunction with our efforts in the future.

3. The XMLVMINE Algorithm

The following describes the interface and the algorithm that we have designed to mine XML documents. We have callled the algorithm XMLVMINE (XML Vertical Miner). The algorithm takes an XML document along with an XPath expression. The user can also enter minimum support and confidence values that are required for the rule generation. The user can then choose to run the algorithm which returns the rules that are generated for the given input.

3.1 Interface

We first discuss the interface that is used to implement the proposed algorithm. This helps when describing the algorithm, because the inputs that are provided to the algorithm are described. The XMLVMINE interface has 12 items of



1. Title bar
2. Input File
3. XPath Root
4. File browse button
5. Item Supp./Conf.
6. Attribute Supp./Conf.
7. Find rules button
8. View file button
9. Edit file button
10. Combine attributes option
11. Allow deletion button
12. Text output field

Figure 7: XMLVMINE Interface

interest as listed in Figure 7. Item 1 is the title bar, which lists the name of the application along with the version number. Item 2 contains the input file name, which must be filled with a valid XML file when the mining is started or an error is returned. Item 3 contains an XPath expression, which allows a subset of the file of interest to be mined. Item 4 (Browse button) is used to help find the file that is chosen for mining. When the user selects this option, a typical Windows file browser is displayed as shown in Figure 8. Item 5 allows the user to enter the item support and item confidence values that are used when the rules contain only items (no attributes). Item 6 allows the user to enter attribute support and confidence values that are used when the candidate contains at least one item with at least one attribute. The values entered into the text box fields in both items 5 and 6 must be a number that is greater

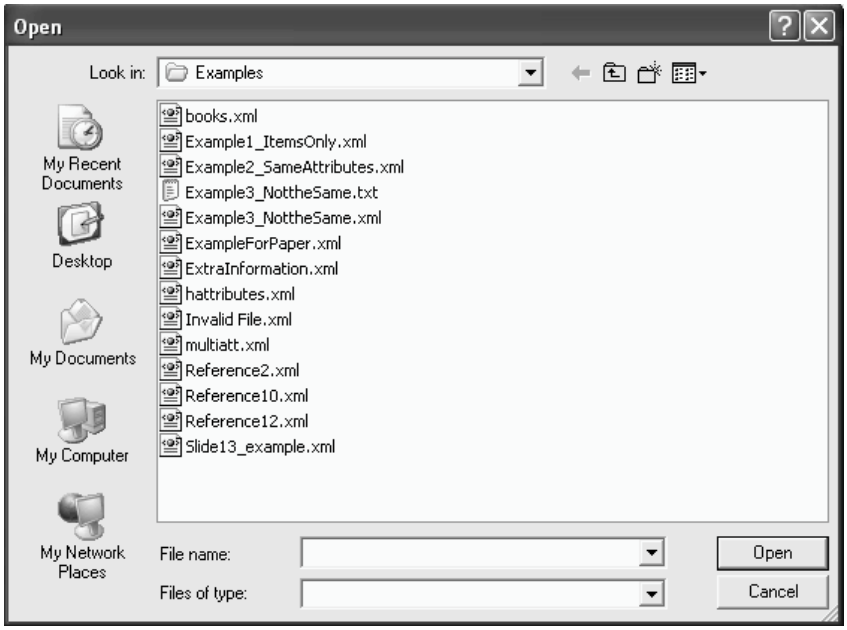


Figure 8: File Browser Interface

than zero and less than or equal to one. Item 7 is the button that allows the user to find all of the rules that meet the support and confidence values listed in items 5 and 6.

Item 8 (View File) allows the user to open the file in Internet Explorer and item 9 (Edit File) opens the XML file in a text editor. Item 10 allows the user to choose whether or not attributes with 100% confidence are combined. Item 11 allows the user to choose whether or not the candidates are modified before generating the rules. The rules when found are shown in item 11, the text area at the bottom of the screen.

3.2 Mining Logic

The XMLVMINE algorithm is used to take an XML file and mine association rules from the document. The algorithm consists of a number of steps that are detailed in the following sections. The input values are validated to make sure they are acceptable, the data is then processed to determine the frequent items in the file, these items are combined to generate candidates for rules, and then these rules are presented to the user. The way in which the inputs are validated is detailed in section 3.2.1 (Input Validate). The first logical section of the algorithm after receiving the inputs is to find the frequent items in the file. This is accomplished by reading the file and creating candidates with 1-item along with the 1-item, 1- attribute candidates. During this pass a tree is created to represent the items that are candidates. Each item or item/attribute pair in the tree contains a link to a candidate in a collection of candidates. Each candidate in the collection contains a count to represent the number of transactions where the item is present. Each candidate in the collection also contains a vertical array of bits which represents which transactions contain the item. The candidates also contain a vertical bit array that has details about transactions where a particular item was found multiple times in the same transaction. The collection of candidates is then used to prune those items that do not have enough

support. Section 3.2.2 (Round 1) has details on this first pass of the data. The second logical section of the algorithm is to generate all combinations of the attributes for a given item. For example, in this phase we may generate <color="blue", size="6">Plates. These can still be considered 1-item candidates, but they may have multiple attributes. This logical section of the algorithm is detailed in Section 3.2.3 (Subsequent Attribute Passes). The third logical section in generating candidates generates all candidates where the number of items > 1. During this stage, items from the $(n-1)^{th}$ pass are combined as long as n-2 items match. This logical section is described in detail in section 3.2.4 (Round 2). The final logical section generates and prints out the rules that are generated from the candidates that were created. This is detailed in section 3.2.5 (Rule Generation).

3.2.1 Input Validation

The first task in the mining process is to make sure that inputs are correct. The XMLVMINE algorithm makes sure that the XML file is a valid file and that it is in market-basket style. This means that there must be a list of transaction objects, which contain items that were purchased. The transactions can have any number of items and the items can have any number of attributes. The items can have any name, and the attributes can have any type with any value. The file can have additional information, but an XPath expression must be provided that returns the transaction items. The XPath expression is checked to make sure it is a valid path to items in a transaction. The support and confidence values are checked to make sure that they are between 0 and 1.

3.2.2 First Round

The first pass through the data generates the candidates that contain a single item and 1-attribute items. The following example shows an XML transaction and the candidates that are generated when processing this transaction. The candidates are

XML Transaction

```
<transaction id="1">
      <items>
        <item color="Red" size = "6" thickness="Thin"> Plates</item>
        <item color ="Blue" size="6" thickness = "Thick"> Plates</item>
    </items>
</transaction>
```

Candidates

```
Plates
<color = "Red">Plates
<size = "6">Plates
<thickness = "Thin">Plates
<color = "Blue">Plates
<thickness = "Thick">Plates
```

Figure 9: An XML transaction with the resulting candidates

generated by looping through each transaction in the file. A tree representing all candidates is generated to keep track of the items that are found in the file along with a collection of candidates. The candidate contains information about a candidate, most importantly the count of the number of times that an item was found in a transaction. A counter is used to keep track of which transaction is being processed. The following pages describe the steps needed to process each transaction.

If the item is not already in the candidate tree then the following tasks are completed:

- Add the item to the tree with a pointer to the new candidate (i.e. the current number of candidates + 1)

- Add 1 to the number of candidates.
- Create a new Candidate object and add it to the collection of candidates (collXMLCandidates) with the following properties.

  Count = 1
  VBitArray = a bit array that is the size of the current transaction, with the last bit set to True
  TreeNodes = a node that contains the item being processed.
  LastTID = the current TID
  Needed = True

If the item was already found in the candidate tree, then the pointer that is stored in the tree with the item that is found is used to access the candidate to update the fields as follows:

Count = Count +1
VBitArray = the current VBitArray with 0's added to the end to make the length equal to the current number of transactions – 1 and one 1 (true) bit added to the end
LastTID = the current TID

When updating the candidate, if the LastTID already matches the current TID, then the Multiitems array is updated to have a true value for the position that corresponds to the transaction being processed. The Multiitems array is an array used to determine which transactions must be looked at closer to determine This triggers the creation of the MultiBitArrays for this transaction after the transaction is processed. The MultiBitArrays object contains the a record for each of the items in a transaction where an item appears more than one time. This allows for the items to be distinguished from one another.

The next step is to process each attribute of the current item to see if the attribute is in the tree as a child of the item. If the attribute is not found in the candidate tree, then the following tasks are completed.

- Add the attribute to the tree as a child of the item along with a pointer to the new candidate (the current number of candidates + 1). If the attribute type is already found, but the attribute value is not, then the new value is added as a child of the type. Figure 10 shows an example of a tree that contains multiple values for the same attribute type (color="Blue", color="Red").

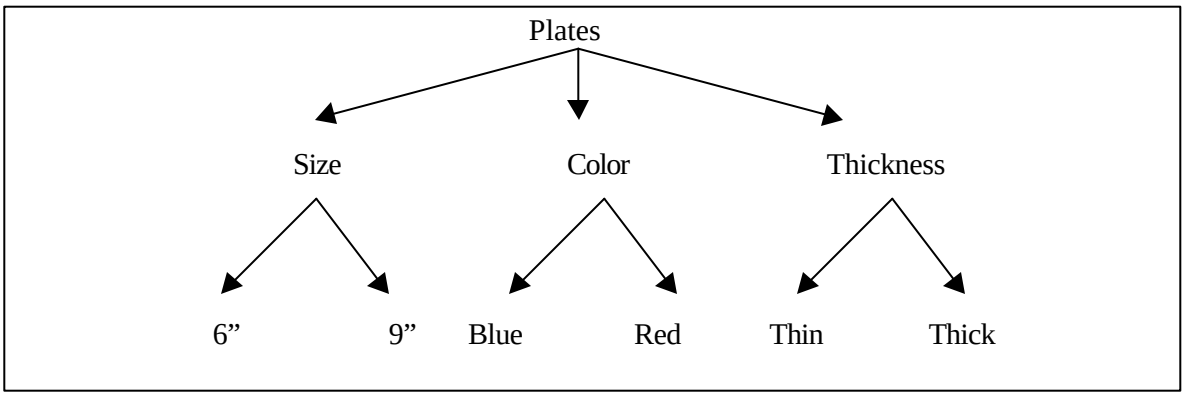


Figure 10: A candidate tree with multiple attributes for the same item.

- Add 1 to the number of candidates and add a reference from the new tree node to the collection of candidates using this count.
- Create a new Candidate object and add it to the collection of candidates with the following properties.

  Count = 1

  VBitArray = a bit array that is the size of the current transaction with the last bit set to 1 (True)

  TreeNodes = a collection of nodes that contains just the item that is being processed

  LastTID = the current TID

Needed = True

If the attribute was already found in the tree, then a pointer to the candidate in the tree is used to access the candidate and update the fields as follows:

Count = Count +1

VBitArray = the current VBitArray with 0's added to the end to make the length equal to the current number of transactions – 1 with a 1 (true) bit added to the end.

LastTID = the current TID

Needed = True

If the LastTID matches the current TID, then the MultiItems array is updated to have a true value for the position that corresponds to the transaction being processed.

If the transaction has the MultiItem array value set to true, then the next step is to break the transaction into its individual items. This is accomplished by completing the following tasks:

- Add 1 to the number of items that were found in transactions that had multiple items. This value gives us a reference to the items in the MultiBitArray object. The value of this is the total number of items that are in transactions that had the same item multiple times.
- The reference of the candidate being processed is pulled from the tree.
- Using the reference, the MultiBitArray object is updated to be the length of the current number of items that have been processed in the transactions that have the same item multiple times.
- For each attribute in the item the first three bulleted items are repeated.

After creating all of the possible candidates, the count of these candidates is evaluated and those candidates that do not meet the support requirements are removed from the collection of candidates. This leaves us with 1-item candidates with 0 or 1 attributes that have the needed support.

3.2.3 Subsequent Attribute Passes

The next task is to determine which attribute values can be combined. For example, after the first pass <color = “Blue”>Plates and <size=“6”>Plates may be candidates, but we need to determine if <color=“Blue” size=“6”>Plates is a candidate. This is accomplished with Apriori style logic. To calculate the n-attribute candidates, the n-1 attribute candidates are combined. The following steps are performed for each pass.

First, determine which candidates are combined to make the next round of candidates. The first criterion is that the items must have (*pass – 1*) attributes that are identical. The second criterion is that the last attribute of each of the candidates must be a different type of attribute. For example, <color=“Blue”>Plates and <color=“Red”>Plates are not be combined, because they are the same attribute type, but <color=“Blue”>Plates and <size=“6”>Plates are combined, because the attribute types are different.

The evaluation of the support of these combinations involves performing an intersection of the VBitArrays of the first item with the VBitArray of the second item and counting the number of 1’s in the resulting bit array. If the count is enough to meet the support, then the VBitArray is intersected with the MultiItems array, which lists transactions that were placed in the MultiBitArrays. The new support count

becomes the result of the first intersection minus the count of the second intersection. This returns the count of all transactions in which the candidate was found in transactions that are not in the MultiBitArrays. The next task is to add back those items that are in the MultiBitArrays. This is accomplished by intersecting the MultiBitArrays for each of the candidates and counting the number of transactions where the items were found in both of these arrays.

If a new candidate has the exact same support as each of the two candidates that were combined to create the new candidate, then the two candidates that were combined can be marked as not being needed. This is because the combined candidate can represent each of the pieces. We can easily tell that if <color="Blue" size="6" material="Plastic">Plates have a support of 60%, then each of the pieces have at least a support of 60%. If they are not shown with a separate rule (showing increased support for the piece), then the support of the piece is exactly 60%.

3.2.4 Second Round

After the first round (first pass and subsequent attribute passes), these items must be combined together to get the n-itemset (n>1) candidates. This is also accomplished with Apriori style logic with some additional logic due to the characteristics of the XML style data. To calculate the n-item candidates, the n-1 item candidates are combined. The first criterion is that the items must have pass – 1 items that are identical. For example, (<color = "Blue">Plates, Cups) and (<color = "Blue">Plates, Balloons) can be combined, because they have (*pass-1*) items that are the same. The resulting candidate are (<color = "Blue">Plates, Cups, Balloons). An additional check is performed to make sure that the additional item from the second

candidate is not a subset of any of the items in the first candidate. For example, <color="Blue" size="6" material="thin"> Plates, <color="Blue" size="6">Cups and <color="Blue" size ="6" material="thin">Plates, <color="Blue">Cups are not be combined.

The evaluation of the support of these combinations involves performing an intersection of the VBitArray of the first item with the VBitArray of the second item and counting the number of 1's in the resulting bit array. If the count is enough to meet the support, then the VBitArray is intersected with the array that lists which transactions were placed in the MultiBitArrays object. The new support count becomes the result of the first intersection minus the count of the second intersection. This returns the count of transactions in which the candidate was found in transactions that are not in the MultiBitArrays. The next task is to add back those items that are in the MultiBitArrays. Intersecting the MultiBitArrays for each of the candidates and counting the number of transactions where the items were found in both of these sets accomplish this.

3.2.5 Rule Generation

The generation of the rules is quite simple after generating the candidates. This process is defined in [2]. In our implementation we allow any number of items in the antecedent and any number in the consequent. To generate the rules, for every large candidate C, we find all non-empty subsets c of C. For every subset that we find, we generate the rules a -> (c - a). We then output this rule if the ratio of support(c) to the support(a) is at least the minimum confidence that was specified.

3.3 Detailed Example

Let's assume that there is a party supply store that sells the following items: Balloons, Cups, Plates and Streamers.

These three items have the following attributes that describe the product:

1. Balloons
    a. Color (Red, Blue, Green)
    b. Size (Small, Medium, Large)
    c. Shape (Round, Oval)
2. Cups
    a. Color (Red, White, Blue, Green)
    b. Size (9oz, 12oz, 24oz)
    c. Material (Plastic, Styrofoam, Paper)
3. Plates
    a. Color (Red, Blue, Green, White)
    b. Size (6", 9", 12")
    c. Thickness (Thin, Medium, Sturdy)
4. Streamers
    a. Color (Red, White, Blue, Green)
    b. Size (12', 20', 30')

Let's say that there are 4 transactions during the time period of interest.

1. Blue 6" Thin Plates, Blue 12' Streamers
2. Red 9" Thin Plates, Blue 6" Thin Plates, Blue 12' Streamers, Red 9oz Cups, Blue 6" Thin

3. Blue 9” Medium Plates, Blue 20‘ Streamers, Red Small Round Balloons, Red 9oz Cups
4. White 20’ Streamers, Red Small Round Balloons

```
<orders>
	<transaction id=“1”>
		<client>Dr. Qin Ding</client>
		<dateofpurchase>10/12/2004</dateofpurchase>
		<items_purchased>
			<item color=“Blue” size = “6” thickness=“Thin”> Plates</item>
			<item color = “Blue” size=“12”> Streamers</item>
		</items_purchased>
	</transaction>
	<transaction id=“2”>
		<client>Dr. Thang Bui</client>
		<dateofpurchase>10/15/2004</dateofpurchase>
		<items_purchased >
			<item color=“Red” size = “9” thickness=“Thin”> Plates</item>
			<item color=“Blue” size = “6” thickness=“Thin”> Plates</item>
			<item color = “Blue” size = “12”> Streamers</item>
			<item color = “Red” size = “9”> Cups</item>
		 </items_purchased>
	</transaction>
	<transaction id=“3”>
		<client>Dr. Sukmoon Chang</client>
		<dateofpurchase>11/1/2004</dateofpurchase>
		<items_purchased >
			<item color = “Blue” size = “9” thickness=“Medium”> Plates</item>
			<item color = “Blue” size = “20”> Streamers </item>
			<item color = “Red” size = “Small” shape=“Round”> Balloons</item>
			<item color = “Red” size = “9”> Cups</item>
		</items_purchased >
	</transaction>
	<transaction id=“4”>
		<client>Dr. Linda Null</client>
		<dateofpurchase>10/15/2004</dateofpurchase>
		<items_purchased >
			<item color=“Red” size=“Small” shape=“Round”> Balloons</item>
			<item color=“white” size=“20”> Streamers</item>
		</items_purchased >
	</transaction>
</orders>
```

Figure 11: XML file used in detailed example

Let's say that there are 4 transactions during the time period of interest.

5. Blue 6" Thin Plates, Blue 12' Streamers
6. Red 9" Thin Plates, Blue 6" Thin Plates, Blue 12' Streamers, Red 9oz Cups, Blue 6" Thin
7. Blue 9" Medium Plates, Blue 20' Streamers, Red Small Round Balloons, Red 9oz Cups
8. White 20' Streamers, Red Small Round Balloons

Figure 11 shows an XML file that represents this scenario. Let's assume that we are looking for items that have 50% support and 75% confidence. The tree shown in Figure 12 shows the candidate tree that is generated during the first pass.

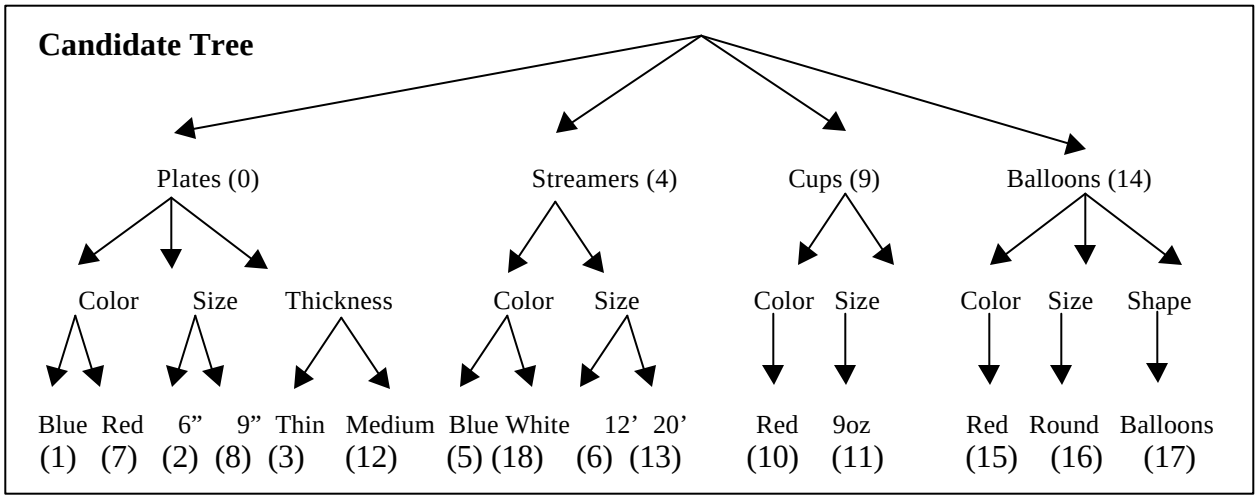


Figure 12: The candidate tree for the detailed example

Table 1 shows the candidate with the details for the candidate. The candidate contains a count along with a bit array for each transaction, along with the bit array for the transactions appear more than one time.

The items that do not have the correct support after the first pass are removed from the collection. This is accomplished by checking the count stored with each candidate. This results in the list of items in Table 2.

VbitArrays																			
ID	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Count	3	3	2	2	4	3	2	1	1	1	1	1	1	2	2	2	2	2	1
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
3	1	1			1	1							1	1	1	1	1	1	0
4					1									1	1	1	1	1	1
MultiBitArrays																			
2	1	0	1	1	0	0	0	1	1	0	0	0							
2	1	1	1	1	0	0	0		0	0	0	0							
2					1	1	1		0	0	0	0							
2									0	1	1	1							

Table 1: VBitArrays and MultiBitArrays for items in example XML document

The next step is to mark items that have the same support as a child node as not being needed for future candidate generation. This allows us to eliminate some of the candidates. In our example, items 0 and 15 are marked as not being needed for

VbitArrays												
ID	0	1	2	3	4	5	6	13	14	15	16	17
Count	3	3	2	2	4	3	2	2	2	2	2	2
1	1	1	1	1	1	1	1	0	0	0	0	0
2	1	1	1	1	1	1	1	0	0	0	0	0
3	1	1			1	1		1	1	1	1	1
4					1			1	1	1	1	1
MultiBitArrays												
2	1	0	1	1	0	0	0					
2	1	1	1	1	0	0	0					
2					1	1	1					
2												

Table 2: VBitArrays and MultiBitArrays for items in XML document after pruning items without enough support.

future passes because an attribute item is found in exactly the same number of transactions as the item. This leaves us with the 0 and 1-attribute candidates. The next step is to generate combinations of items that are of the same item type to get all of the possible 1-itemsets with more than one attribute. In this example the following 2-attribute candidates are created: Plate Combinations (1,2), (1,3), (2,3), Streamer

Plates
    Support: 75%
<color=Blue>Plates
    Support: 75%
Streamers
    Support: 100%
<color=Blue>Streamers
    Support: 75%
<size=12>Streamers
    Support: 50%
<size=9>Plates
    Support: 50%
Cups
    Support: 50%
<size=20>Streamers
    Support: 50%
Balloons
    Support: 50%
<color=Blue, size=12>Streamers
    Support: 50%
<color=Red, size=9>Cups
    Support: 50%
<color=Blue, size=6, thickness=Thin>Plates
    Support: 50%
<color=Red, size=Small, shape=Round>Balloons
    Support: 50%

Figure 13: 1-item candidate sets for detailed example

combinations (5,6) and Balloon Combinations (16,17), (16,18), and (17,18). Notice that items 2 and 3 are removed from the candidate set when 2,3 are combined because 2 and 3 have the same support and that support stays the same when combined.

<color=Blue>Plates, Streamers
<color=Blue>Plates, <color=Blue>Streamers
<color=Blue>Plates, <size=12>Streamers
<color=Blue>Plates, <size=9>Plates
<color=Blue>Plates, <color=Blue, size=12>Streamers
<color=Blue>Plates, <color=Red, size=9>Cups
Streamers, <size=9>Plates
Streamers, <color=Red, size=9>Cups
Streamers, <color=Blue, size=6, thickness=Thin>Plates
Streamers, <color=Red, size=Small, shape=Round>Balloons
<color=Blue>Streamers, <size=12>Streamers
<color=Blue>Streamers, <size=9>Plates
<color=Blue>Streamers, <color=Red, size=9>Cups
<color=Blue>Streamers, <color=Blue, size=6, thickness=Thin>Plates
<size=12>Streamers, <color=Blue, size=6, thickness=Thin>Plates
<size=9>Plates, <color=Red, size=9>Cups
<size=20>Streamers, <color=Red, size=Small, shape=Round>Balloons
<color=Blue, size=12>Streamers, <color=Blue, size=6, thickness=Thin>Plates
<color=Blue>Plates, Streamers, <size=9>Plates
<color=Blue>Plates, Streamers, <color=Red, size=9>Cups
<color=Blue>Plates, <color=Blue>Streamers, <size=12>Streamers
<color=Blue>Plates, <color=Blue>Streamers, <size=9>Plates
<color=Blue>Plates, <color=Blue>Streamers, <color=Red, size=9>Cups
<color=Blue>Plates, <size=9>Plates, <color=Red, size=9>Cups
Streamers, <size=9>Plates, <color=Red, size=9>Cups
<color=Blue>Streamers, <size=12>Streamers, <color=Blue, size=6, thickness=Thin>Plates
<color=Blue>Streamers, <size=9>Plates, <color=Red, size=9>Cups
<color=Blue>Plates, Streamers, <size=9>Plates, <color=Red, size=9>Cups
<color=Blue>Plates, <color=Blue>Streamers, <size=9>Plates, <color=Red, size=9>Cups

Figure 14: Large candidates for detailed example (n > 1)

These combinations are then combined in additional passes as long as n-1 items are the same. For example in pass 2, the first 1 items must match. In our example, we can therefore combine items (1,2) and (1,3) to create the candidate (1,2,3). Notice that when these items are combined the support is not the same. Item 1 has higher support than both 2 and 3. Item 2,3 is then checked to see if it should be removed. Because item (2,3) has the same support as item (1,2,3) item (2,3) is marked as not being needed. We can also combine item (16,17) and (16,18) to create the candidate (16,17, 18). Notice that the items (16,17) and (16,18) are also removed

for the same reason as previously noted. The item (17,18) is also found and removed because it also has the same support and can be represented by (16,17,18).

```
**********RULES**********
<color=Blue>Plates  ----->  Streamers
    Support: 75%
    Confidence: 100%
<color=Blue>Plates  ----->  <color=Blue>Streamers
    Support: 75%
    Confidence: 100%
Streamers  ----->  <color=Blue>Plates
    Support: 75%
    Confidence: 75%
<color=Blue>Streamers  ----->  <color=Blue>Plates
    Support: 75%
    Confidence: 100%
<size=12>Streamers  ----->  <color=Blue>Plates
    Support: 50%
    Confidence: 100%
<size=12>Streamers  ----->  <color=Blue>Streamers
    Support: 50%
    Confidence: 100%
<size=12>Streamers  ----->  <color=Blue, size=6, thickness=Thin>Plates
    Support: 50%
    Confidence: 100%
<size=12>Streamers  ----->  <color=Blue>Plates, <color=Blue>Streamers
    Support: 50%
    Confidence: 100%
<size=12>Streamers  ----->  <color=Blue>Streamers, <color=Blue, size=6, thickness=Thin>Plates
    Support: 50%
    Confidence: 100%
```

Figure 15: Rules generated for detailed example

None of the 3 itemsets have two matching items, so the attribute passes are complete and we are left with the candidates shown in Figures 13 and 14. Each combination of these items is combined to create the n-itemsets. Notice that Streamers is not combined with Blue Streamers, because the single 0-attribute candidates are not combined with the same single item with attributes. This leaves us with the candidate sets described in Figures 12 and 13. Also notice that <color=Blue>Streamers is not combined with <color=Blue,size=12>Streamers, because the <color=Blue>Streamers is a superset of <color=Blue,size=12>Streamers.

Each combination of the large candidates is used to create the rules. The combinations are evaluated to see if the rule has enough support. Some of the rules that are generated are shown in Figure 15.

3.4 Performance Evaluation

The performance tests were performed on a Compaq Presario 700 series Notebook computer with Windows XP Version 5.1 (Build 2600.xpsp2.030422-1633: Service Pack 1) with 245,228 KB of physical memory and 20 GB Hard Disk space. The implementation was written in the Microsoft Development Environment 2003 Version 7.1.3088 a part of the Microsoft .NET Framework Version 1.1.4322. The programming environment used was Microsoft Visual Basic .NET. There are two types of performance tests that we have decided to perform. The first is to ensure that the algorithm returns the appropriate results and the second so that the results are returned in an efficient manner.

The goal of this research was to be able to mine the attributes along with the items in an XML file. Several tests were performed to make sure that the wide variety of how items appear in XML documents can be mined. The input files and output files are contained in Appendix B.

Test: Presence of attributes

Test 1: Items only

Test 2: Items with attributes

The algorithm must still work when there are only items in the file. Test 1 is run with a file that has only items in the file, to show that the algorithm operates correctly. Test 2 shows a simple example where attributes are found.

Test: Number of attributes

Test 3: 1 attribute for each item

Test 4: 5 attributes for each item

Test 3 and 4 show that it does not matter how many attributes are given for an item. The algorithm can handle any number of attributes for the items.

Test: Different attributes

Test 5: Items appear in transactions with different attributes

Test 6: Items appear in same transaction with different attributes

Test 5 and 6 show that items can have the same attribute in different transactions or they can have different attributes and the algorithm calculates the correct support and confidence.

Test: Minimum support

Test 7: Minimum item support is met

Test 8: Minimum item support is met, but minimum attribute support is not met.

Test 9: Minimum item support is not met, but attribute support is met

Test 10: Minimum item support is not met, and attribute support is not met.

Tests 7 through 10 detail when the algorithm is run for XML files that meet the required support and likewise when they do not meet the required support.

Test: Minimum confidence

Test 11: Minimum item confidence is not met

Test 12: Minimum item confidence is met, but minimum attribute confidence is not met.

Test 13: Minimum item confidence is not met, but attribute confidence is met

Test 14: Minimum item confidence is not met, and attribute confidence is not met.

Tests 11 through 14 detail when the algorithm is run for XML files that meet the required confidence and likewise when they do not meet the required confidence.

The results of these tests show that the algorithm functions properly for the wide variety of scenarios that occur in XML documents. The rules that are generated take into account both item and attribute support and confidence.

The following charts show how the algorithm performs in regard to execution time. For this test 100 transactions were generated each with 3 items. The 3 items each had 3 attributes and these attributes had 3 possible values. The 3 values were found equally in the file, but they were each found with each of the other possible attributes. This means that the resulting rules were combinations of all of the individual items with the attributes. None of the items can be combined, because the confidence was never 100%. The additional files that were tested for this graph

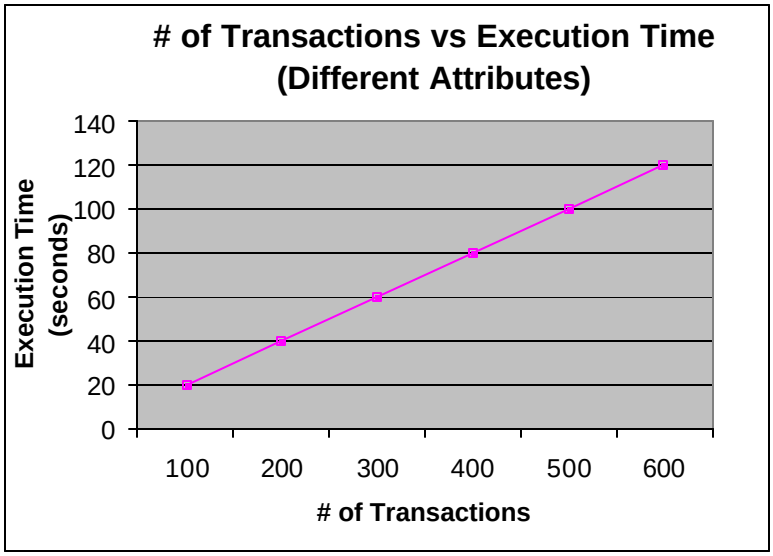


Figure 16: Different attributes performance evaluation

contained 200, 300, 400, 500, and 600 transactions. Each of the additional files were the original file copied a number of times to reach the goal of the file size that was being tested. This allowed for the test to show how the number of transactions affects

the file. This shows that the algorithm performs in a manner similar to other Apriori-style algorithms.

The second graph shows the performance of the algorithm if the same tests were performed with files where the items always had the same attributes. This allows the attributes to be combined in the early stages (candidate generation) and the number of candidates are reduced along with the number of rules that were generated. By combining items that always have the same confidence value we are able to reduce the number of candidates, the number of rules, and the execution time of the algorithm.

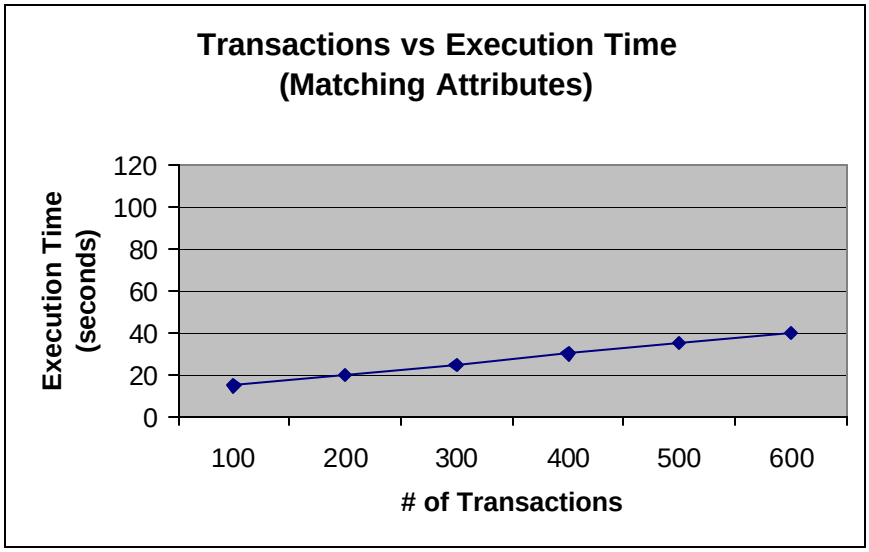


Figure 17: Matching attributes performance evaluation

The execution time of our algorithm changes based on the number of candidates that are generated. If multiple attributes can be combined, then the execution time decreases even though there are the same number of items, the same number of attributes, and the same number of attribute values. The worst case

execution time of the algorithm however is exponential due to the fact that the rules are combinations of all candidates.

4. Conclusion

The algorithm discussed in this paper shows that XML data can be mined using the vertical approach. The approach that is given allows for both items and the attributes of these items to be mined. As the performance evaluation shows, the algorithm takes into account the wide variety of ways that attributes of items can appear in an XML file. This performance evaluation also shows that the attributes can be mined in a similar fashion to that of previous algorithms that focused on the items alone.

The current algorithm generates association rules for XML items and attributes. The next step allows for more than one level of attributes (i.e. attributes of the attributes). For example, the file shown in Figure 18 cannot be mined with the current algorithm.

```
<transaction id="1">
      <items>
        <item size = "6" thickness="Thin"> Plates
          <color shade="Dark">Red</Color>
        </item>
        <item size = "6" thickness="Thick"> Plates
          <color shade="Light">Blue</Color>
        </item>
</items>
</transaction>
```

Figure 18: File with properties that cannot be mined using XMLVMINE

Another area of research is to combine the research done here with research done to allow XQuery statements to help define the data that is useful to mine. Algorithms such as XMINE, which allow for more detailed querying of the data can help in real world use of this algorithm.

Other enhancements to the application can involve some of the performance enhancements of the VIPER algorithm. For example, we can add compression of the bit arrays. Note that in certain implementations the bit array objects are compressed. In Visual Basic .NET the Bit Array object is not compressed automatically, so this can be added.

It would also be useful to research whether or not the order in which the candidates are examined effects the execution time of the algoirithm. It might be useful when generating candidates to order the items in the candidate by the support values. It might be possible to decrease the number of candidates that need to be checked for adequate support depending on the order that the candidates are generated.

Although there is much more to be done in dealing with mining association rules from XML documents, the proposed algorithm shows that the attributes of the items in an XML document can be mined in a similar fashion to items in the document. This can be done in a way that allows users to choose what is included in the rules.

References

1. R. Agrawal, T. Imielinski, and A. Swami, “Mining Association Rules Between Sets of Items in Large Database”, SIGMOD 93.

2. R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” VLDB 94.

3. M. Bawa, G. Bhalotia, J. Haritsa, P. Shah, P. Shenoy, S. Sudarshan. “Turbo-charging Vertical Mining of Large Databases.”

4. A. Berson, S. Smith, K. Thearling. Building Data Mining Applications for CRM. McGraw-Hill, New York, NY, 1999.

5. F. Bodon. “A fast APRIORI implementation,” FIMI 03

6. C. Borgelt. “Efficient Implementation of Apriori and Eclat,” FIMI 03.

7. D. Braga, A. Campi, S. Ceri, M. Klemettinen, PL. Lanzi, “Discovering Interesting Information in XML Data with Association Rules”, in Proceedings of ACM Symposium on Applied Computing, Melbourne, USA, Mar. 2003, pp. 450-454.

8. D. Braga, A. Campi, S. Ceri, M. Klemettinen, PL. Lanzi, “ Mining Association Rules from XML Data”, in Proceedings of DEXA 2002 (DaWaK), LNCS 2454, Aix-en-Provence, France, Sep. 2002, pp. 21-30.

9. D. Braga, A. Campi, S. Ceri, M. Klemettinen, PL. Lanzi, “A Tool for Extracting XML Association Rules from XML Documents”, in Proceedings of IEEE-ICTAI 2002, Washington DC, USA, Nov. 2002, pp. 57-64.

10. S. Brin et al, “Dynamic Itemset Counting and Implication Rules for Market Basket Data”, SIGMOD 97.

11. G. Dobbie, J. W. W. Wan. “Mining Association Rules from XML Data using XQuery.” Proceedings of DMWI’04, Dunedin, New Zealand, January 2004.

12. G. Dobbie, J. W. W. Wan. “Extracting Association Rules from XML Documents using XQuery.” Proceedings of Fifth Internation Workshop on Web Information and Data Management (WIDM 03), New Orleans, LA, USA. Novermber 2003.

13. J. Han, J. Pei and Y. Yin, “Mining Frequent Patterns without Candidate Generation”, SIGMOD 2000.

14. J. Han and Y. Fu, “Discovery of Multiple-level Association Rules from Large Databases”, VLDB 95.

15. J. Han and M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann, San Francisco, CA, 2001.

16. A. Homer, XML IE5 Programmer's Reference, Wrox Press, Canada, 1999.

17. Jong Soo Park, Ming-Syan Chen and Philip S. Yu, "An effective Hash-Based Algorithm for Mining Association Rules," SIGMOD 95.

18. A. Termier, M-C. Rousset, M. Sebag, "TreeFinder: a First Step towards XML Data Mining", in Proceedings of IEEE International Conference on Data Mining, 2002.

19. World Wide Web Consortium. Document Object Model (DOM), http://www.w3.org/ DOM/, April 2004.

20. World Wide Web Consortium. XML Path Language (XPath) 2.0, W3C Working Draft. http://www.w3.org/TR/2004/WD-xpath20-20041029/, October 2004.

21. World Wide Web Consortium. XML Path Language (XPath) 2.0, W3C Working Draft. http://www.w3.org/TR/2004/WD-xquery-20041029/, October 2004.

Appendix A: Object Descriptions

1. CandidateTree – The CandidateTree object stores all possible candidates, along with a link to an XMLCandidate object (described below) which stores details of the candidate. An XML file and the resulting candidate tree are shown in Figure 19. The numbers in the parentheses are the references to the XMLCandidates in the collXML Candidates collection (see item 2).

```
<transactions>
  <transaction id="1">
    <items>
      <item color="Red" size = "6" thickness="Thin"> Plates</item>
      <item color="Blue" size = "12" thickness="Thick"> Plates</item>
    </items>
  </transaction>
  </transaction>
</transactions>
```

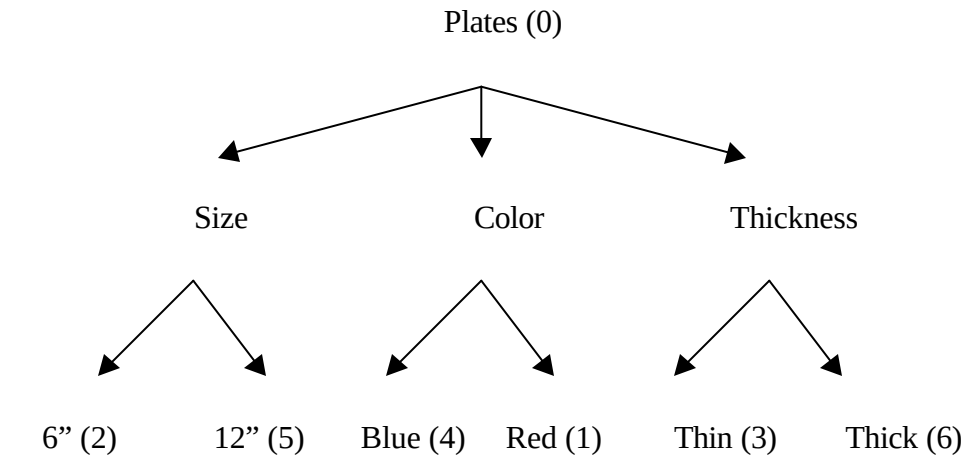


Figure 19: A candidate tree

2. CollNodes – The CollNodes (collection of nodes) object allows a candidate to have a property (TreeNodes) which is a collection of nodes. Each node represents an

item in the candidate set. This allows each candidate to be a collection of items with attributes.

3. **collXMLCandidates-** The collXMLCandidates is a collection of XMLCandidates, which contains all candidates that have the necessary support.

4. **MultiItems -** This MultiItems object is a bit array which contains a True/False value for each transaction, which contains whether or not the transaction had to be broken into its parts. This is only set to True if the same item is found more than one time in a transaction.

5. **MultiTransactionIDs** – The MultiTransactionIDs array contains a list of the transaction IDs that needed to be broken apart. It contains a value for each item in these transactions. For example, the following XML file has a MultiTransactionIDs array of [1, 1, 2, 2, 2, 4, 4].

```
<transaction id="1">
<items>
  <item color="Red" size = "6" thickness="Thin"> Plates</item>
  <item color ="Blue" size="12" thickness = "Thick"> Plates</item>
</items>
<items>
  <item color="Red" size = "6" thickness="Thin"> Plates</item>
  <item color ="Blue" size="12" thickness = "Thick"> Plates</item>
  <item color ="Blue" size="12" material="Plastic"> Cups</item>
</items>
<items>
  <item color="Red" size = "6" thickness="Thin"> Plates</item>
  <item color ="Blue" size="12" thickness = "Thick"> Cups</item>
</items>
<items>
  <item color="Red" size = "6" thickness="Thin"> Plates</item>
  <item color ="Blue" size="12" thickness = "Thick"> Plates</item>
</items>
</transaction>
```

Figure 20: An item in multiple transactions

This allows us to determine if the candidate appears in the item of interest. If we are examining transaction 2, we know that we need to look at items 3, 4, and 5

in the MultiBitArray object. The intersection of the VBitArrays and the MultiItems bit array indicates which transactions we need to examine further. We can then use this bit array to determine which items in the MultiBitArrays need to be examined. In addition, a pointer to the first item in the MultiBitArrays for each transaction is stored so that we can easily find the starting point of a particular transaction.

6. **TID -** The TID (transaction ID) is the current transaction counter.
7. **XMLCandidates** – An XML Candidate object contains a number of properties: Count, VBitArray, MultiBitArray, TreeNodes, LastTID, FirstItem and SecondItem.
   - **Count:** The count is simply the number of times that the candidate appears in the transaction list. The count divided by the number of transactions gives the support of a candidate.
   - **VBitArray:** The VBitArray is a bit array that has a maximum length equal to the number of transactions. The actual length of a VBitArray is the number of the last TID where this candidate was found. Each index in the bit array represents a transaction and has the value of 1 (True) if the candidate is in the transaction, otherwise it is 0 (False).
   - **MultiBitArray:** The MultiBitArray contains the details for each transaction that has the same item listed multiple times. This allows items in a transaction to be segregated into the detailed items in the list. For example, if you had the following transaction then there are entries in the MultiBitArray for this transaction because the item “Plates” appears more than once.

The MultiBitArray for the object <color="Red">Plates is 1,0. The MultiBitArray for the object <size="6">Plates is 1,0. The MultiBitArray for the object <size = "12">Plates is 0,1. This allows us to determine that there are <color="Red" size="6">Plates, but not <color="Red" size="12">Plates, because the intersection of Red Plates and 6" Plates is 1,0 and the intersection of Red Plates and 12" Plates is 0,0.

```
<transaction id="1">
  <items>
    <item color="Red" size = "6" thickness="Thin"> Plates</item>
    <item color="Blue" size="12" thickness="Thick"> Plates</item>
  </items>
</transaction>
```

Figure 21: An item multiple times in the same transaction

- **TreeNodes:** The TreeNode object is a collection of nodes (see collNodes above) which contain the items that are part of the candidate. This allows us to store and combine the items for subsequent passes, and is also used when printing out the rules. The trees can easily be printed out in XML Format.
- **LastTID:** The LastTID property contains the last TID that contains this candidate. This allows us to determine if the item is found twice in the same transaction, which triggers the creation of the MultiBitArray.
- **FirstItem:** Each item except for the 1-itemsets with 0 or 1 attributes are a combination of two other itemsets. This property allows access to the candidates that are used to create the new candidate. This is used during creation of the n-item candidates where n > 1. We need to make sure that the items being combined do not have pieces that are the same. For example, we

do not want to combine <color="Blue" size="6">Plates and <color="Blue"> Plates. The <color="Blue" size="6">Plates has a FirstItem that is the same as the <color="Blue">Plates. We therefore are able to determine that these items cannot be combined.

- **SecondItem:** This contains the second item that was used to create the candidate. See FirstItem above for details on how this is used.

Appendix B: Test Cases

The performance section of the paper gives details on why these tests were run. The results of running the following tests through the XMLVMINE algorithm are shown in this section. The "Test" section describes the reason for the test cases. The test cases (Tests 1 through 14) are the actual tests that were run to show that the algorithm works correctly.

Test: Presence of attributes

Test 1: Items only
Test 2: Items with attributes

Test: Number of attributes

Test 3: 1 attribute for each item
Test 4: 5 attributes for each item

Test: Different attributes

Test 5: Items appear in transactions with different attributes
Test 6: Items appear in same transaction with different attributes

Test: Minimum support

Test 7: Minimum item support is met
Test 8: Minimum item support is met, but minimum attribute support is not met
Test 9: Minimum item support is not met, but attribute support is met
Test 10: Minimum item support is not met, and attribute support is not met

Test: Minimum confidence

Test 11: Minimum item confidence is not met
Test 12: Minimum item confidence is met, but not minimum attribute confidence
Test 13: Minimum item confidence is not met, but attribute confidence is met
Test 14: Minimum item confidence is not met, and attribute confidence is not met

Test 1: Items only

Input:

```
<transactions>
  <transaction id="1">
    <items>
      <item>A</item>
      <item>B</item>
    </items>
  </transaction>
  <transaction id="2">
    <items>
      <item>A</item>
      <item>D</item>
      <item>E</item>
    </items>
  </transaction>
  <transaction id="3">
    <items>
      <item>A</item>
      <item>B</item>
      <item>E</item>
    </items>
  </transaction>
</transactions>
```

Minimum Support: 0.5, Minimum Confidence 0.75

Output:

**********RULES**********

B -----> A
Support: 67%
Confidence: 100%

E -----> A
Support: 67%
Confidence: 100%

Test 2: Items with attributes

Input:

```
<transactions>
  <transaction id="1">
    <items>
      <item color="Blue" size = "6" thickness="Thin"> Plates</item>
      <item color = "Blue" size="12"> Streamers</item>
    </items>
  </transaction>
  <transaction id="2">
    <items>
      <item color="Red" size = "6" thickness="Thin"> Plates</item>
      <item color = "Red" size = "9oz"> Cups</item>
    </items>
  </transaction>
  <transaction id="3">
    <items>
      <item color = "Blue" size = "6" thickness="Medium"> Plates</item>
      <item color = "Red" size = "Small" shape="Round"> Balloons</item>
    </items>
  </transaction>
  <transaction id="4">
    <items>
      <item color="Red" size = "6" thickness="Thin"> Plates</item>
      <item color = "Red" size = "Small" shape="Round"> Balloons</item>
    </items>
  </transaction>
</transactions>
```

Minimum Support: 0.75, Minimum Confidence 0.75
Minimum Attribute Minimum Support: 0.75, Attribute Confidence 0.75

Output:

```
**********RULES**********
<color=Red, size=9oz>Cups   ----->   <color=Blue, size=6, thickness=Thin>Plates
      Support: 75%
      Confidence: 100%
<color=Blue, size=6, thickness=Thin>Plates   ----->   <color=Red, size=9oz>Cups
      Support: 75%
      Confidence: 75%
```

Test 3: 1 attribute for each item

Input:

```
<transactions>
  <transaction id="1">
    <items>
      <item color="Blue"> Plates</item>
      <item color = "Blue"> Streamers</item>
    </items>
  </transaction>
  <transaction id="2">
    <items>
      <item color="Red"> Plates</item>
      <item color = "Red"> Cups</item>
    </items>
  </transaction>
  <transaction id="3">
    <items>
      <item color = "Blue"> Plates</item>
      <item color = "Red"> Balloons</item>
    </items>
  </transaction>
  <transaction id="4">
    <items>
      <item color="Red"> Plates</item>
      <item color = "Red"> Balloons</item>
    </items>
  </transaction>
</transactions>
```

Minimum Support: 0.75, Minimum Confidence 0.75
Minimum Attribute Support: 0.5, Minimum Attribute Confidence 0.75

Output:

**********RULES**********
<color=Red>Balloons -----> Plates
Support: 50%
Confidence: 100%

Test 4: 5 attributes for each item

```
<transactions>
  <transaction id="1">
    <items>
      <item color="Blue" size="small" material="plastic" pattern="none"
            brand="generic"> Plates</item>
    </items>
  </transaction>
  <transaction id="2">
    <items>
      <item color="Red" size="medium" material="plastic" pattern="striped"
            brand="generic"> Plates</item>
    </items>
  </transaction>
  <transaction id="3">
    <items>
      <item color="Green" size="large" material="plastic" pattern="solid"
            brand="dole"> Plates</item>
    </items>
  </transaction>
  <transaction id="4">
    <items>
      <item color="Yellow" size="small" material="plastic" pattern="striped"
            brand="generic"> Plates</item>
    </items>
  </transaction>
</transactions>
```

Minimum Support: 0.75, Minimum Confidence 0.75
Minimum Attribute Support: 0.5, Minimum Attribute Confidence 0.75

Output:

```
**********RULES**********
Plates-----><material=plastic>Plates
    Support: 100%
    Confidence: 100%
Plates-----><brand=generic>Plates
    Support: 75%
    Confidence: 75%
Plates-----><material=plastic, brand=generic>Plates
    Support: 75%
    Confidence: 75%
Plates-----><material=plastic>Plates, <brand=generic>Plates
    Support: 75%
    Confidence: 75%
<material=plastic>Plates-----><brand=generic>Plates
    Support: 75%
    Confidence: 75%
<brand=generic>Plates-----><material=plastic>Plates
    Support: 75%
    Confidence: 100%
<brand=generic>Plates-----><material=plastic, brand=generic>Plates
    Support: 75%
    Confidence: 100%
```

Test 5: Items appear in transactions with different attributes

Input:

```
<transactions>
  <transaction id="1">
    <items>
      <item color="Blue"> Plates</item>
      <item color = "Blue"> Streamers</item>
    </items>
  </transaction>
  <transaction id="2">
    <items>
      <item color="Red"> Plates</item>
      <item color = "Blue"> Streamers</item>
    </items>
  </transaction>
  <transaction id="3">
    <items>
      <item color = "Blue"> Plates</item>
      <item color = "Red"> Streamers</item>
    </items>
  </transaction>
  <transaction id="4">
    <items>
      <item color="Blue"> Plates</item>
      <item color = "Green"> Streamers</item>
    </items>
  </transaction>
</transactions>
```

Minimum Support: 0.75, Minimum Confidence 0.75
Minimum Attribute Support: 0.75, Minimum Attribute Confidence 0.75

Portion of Output:

```
Plates-----><color=Blue>Plates
      Support: 75%, Confidence: 75%
Plates----->Streamers
      Support: 100%, Confidence: 100%
Plates-----><color=Blue>Plates, Streamers
      Support: 75%, Confidence: 75%
<color=Blue>Plates----->Streamers
      Support: 75%, Confidence: 100%
<color=Blue>Plates----->Plates, Streamers
      Support: 75%, Confidence: 100%
Streamers----->Plates
      Support: 100%
      Confidence: 100%
Streamers-----><color=Blue>Plates
      Support: 75%
      Confidence: 75%
```

Test 6: Items appear in same transaction with different attributes

Input:

```
<transactions>
  <transaction id="1">
    <items>
      <item color="Blue"> Plates</item>
      <item color="Red"> Plates</item>
    </items>
  </transaction>
  <transaction id="2">
    <items>
      <item color="Blue"> Plates</item>
      <item color="Red"> Plates</item>
    </items>
  </transaction>
  <transaction id="3">
    <items>
      <item color="Blue"> Plates</item>
      <item color="Red"> Plates</item>
    </items>
  </transaction>
  <transaction id="4">
    <items>
      <item color="Blue"> Plates</item>
      <item color = "Green"> Streamers</item>
    </items>
  </transaction>
</transactions>
```

Minimum Support: 0.75, Minimum Confidence 0.75
Minimum Attribute Support: 0.75, Minimum Attribute Confidence 0.75

Output:

```
**********RULES**********
Plates-----><color=Blue>Plates
      Support: 100%
      Confidence: 100%
Plates-----><color=Red>Plates
      Support: 75%
      Confidence: 75%
Plates-----><color=Blue>Plates, <color=Red>Plates
      Support: 75%
      Confidence: 75%
<color=Blue>Plates-----><color=Red>Plates
      Support: 75%
      Confidence: 75%
<color=Red>Plates-----><color=Blue>Plates
      Support: 75%
      Confidence: 100%
```

Test 7: Minimum item support is met

Input:

```
<transactions>
  <transaction id="1">
    <items>
      <item>A</item>
      <item>B</item>
    </items>
  </transaction>
 <transaction id="2">
    <items>
      <item>A</item>
      <item>B</item>
      <item>C</item>
    </items>
  </transaction>
 <transaction id="3">
    <items>
      <item>A</item>
      <item>B</item>
      <item>D</item>
    </items>
  </transaction>
 <transaction id="4">
    <items>
      <item>A</item>
      <item>E</item>
      <item>F</item>
    </items>
  </transaction>
</transactions>
```

Minimum Support: 0.75, Minimum Confidence 0.75
Minimum Attribute Support: 0.75, Minimum Attribute Confidence 0.75

Output:

```
**********RULES**********
A----->B
      Support: 75%
      Confidence: 75%
B----->A
      Support: 75%
      Confidence: 100%
```

Test 8: Minimum item support is met, but minimum attribute support is not met.

Input:

```
<transactions>
  <transaction id="1">
    <items>
      <item color="Blue"> Plates</item>
      <item color = "Blue"> Balloons</item>
    </items>
  </transaction>
  <transaction id="2">
    <items>
      <item color="Red"> Plates</item>
      <item color = "Red"> Balloons</item>
    </items>
  </transaction>
  <transaction id="3">
    <items>
      <item color = "Green"> Plates</item>
      <item color = "Green"> Balloons</item>
    </items>
  </transaction>
  <transaction id="4">
    <items>
      <item color="Yellow"> Plates</item>
      <item color = "Yellow"> Balloons</item>
    </items>
  </transaction>
</transactions>
```

Minimum Support: 0.5, Minimum Confidence 0.75
Minimum Attribute Support: 0.75, Minimum Attribute Confidence 0.75

Output:

```
**********RULES**********
Plates----->Balloons
      Support: 100%
      Confidence: 100%
Balloons----->Plates
      Support: 100%
      Confidence: 100%
```

Test 9: Minimum item support is not met, but attribute support is met

Input:

```
<transactions>
  <transaction id="1">
    <items>
      <item color="Blue"> Plates</item>
      <item color = "Blue"> Balloons</item>
    </items>
  </transaction>
  <transaction id="2">
    <items>
      <item color="Blue"> Plates</item>
      <item color = "Red"> Balloons</item>
    </items>
  </transaction>
  <transaction id="3">
    <items>
      <item color = "Blue"> Cups</item>
      <item color = "Blue"> Streamers</item>
    </items>
  </transaction>
  <transaction id="4">
    <items>
      <item color="Yellow">Cups</item>
      <item color = "Yellow">Streamers</item>
    </items>
  </transaction>
</transactions>
```

Minimum Support: 0.75, Minimum Confidence 0.75
Minimum Attribute Support: 0.5, Minimum Attribute Confidence 0.75

Output:

```
**********RULES**********
Plates-----><color=Blue>Plates
      Support: 50%
      Confidence: 100%
Plates-----><color=Blue>Plates, Balloons
      Support: 50%
      Confidence: 100%
<color=Blue>Plates-----><color=Blue>Balloons
      Support: 50%
      Confidence: 100%
Balloons-----><color=Blue>Plates
      Support: 50%
      Confidence: 100%
```

Test 10: Mimimum item support is not met, and attribute support is not met.

Input:

```
<transactions>
  <transaction id="1">
    <items>
      <item color="Blue"> Plates</item>
      <item color = "Blue"> Balloons</item>
    </items>
  </transaction>
  <transaction id="2">
    <items>
      <item color="Blue"> Plates</item>
      <item color = "Red"> Balloons</item>
    </items>
  </transaction>
  <transaction id="3">
    <items>
      <item color = "Blue"> Cups</item>
      <item color = "Blue"> Streamers</item>
    </items>
  </transaction>
  <transaction id="4">
    <items>
      <item color="Yellow">Cups</item>
      <item color = "Yellow">Streamers</item>
    </items>
  </transaction>
</transactions>
```

Minimum Support: 0.75, Minimum Confidence 0.75
Minimum Attribute Support: 0.75, Minimum Attribute Confidence 0.75

Output:

**********RULES**********

Test 11: Minimum item confidence is not met

Output:

```
<transactions>
  <transaction id="1">
    <items>
      <item color="Blue"> Plates</item>
      <item color = "Blue"> Streamers</item>
    </items>
  </transaction>
  <transaction id="2">
    <items>
      <item color="Blue"> Plates</item>
      <item color = "Blue"> Streamers</item>
    </items>
  </transaction>
  <transaction id="3">
    <items>
      <item color = "Blue"> Plates</item>
      <item color = "Red"> Balloons</item>
    </items>
  </transaction>
  <transaction id="4">
    <items>
      <item color="Red"> Plates</item>
      <item color = "Red"> Balloons</item>
    </items>
  </transaction>
</transactions>
```

Minimum Support: 0.5, Minimum Confidence 0.75
Minimum Attribute Support: 0.75, Minimum Attribute Confidence 0.75

Output:

```
**********RULES**********
Plates-----><color=Blue>Plates
      Support: 75%
      Confidence: 75%
Streamers----->Plates
      Support: 50%
      Confidence: 100%
Balloons----->Plates
      Support: 50%
      Confidence: 100%
```

*Note that Plates → Streamers is not found.

Test 12: Minimum item confidence is met, but minimum attribute confidence is not.

Input:

```
<transactions>
  <transaction id="1">
    <items>
      <item color="Blue"> Plates</item>
      <item color = "Blue"> Streamers</item>
    </items>
  </transaction>
  <transaction id="2">
    <items>
      <item color="Blue"> Plates</item>
      <item color = "Blue"> Streamers</item>
    </items>
  </transaction>
  <transaction id="3">
    <items>
      <item color = "Blue"> Plates</item>
      <item color = "Red"> Balloons</item>
    </items>
  </transaction>
  <transaction id="4">
    <items>
      <item color="Red"> Plates</item>
      <item color = "Red"> Balloons</item>
    </items>
  </transaction>
</transactions>
```

Minimum Support: 0.5, Minimum Confidence 0.5
Minimum Attribute Support: 0.8, Minimum Attribute Confidence 0.8

Output:

```
**********RULES**********
Plates----->Streamers
      Support: 50%
      Confidence: 50%
Plates----->Balloons
      Support: 50%
      Confidence: 50%
Streamers----->Plates
      Support: 50%
      Confidence: 100%
Balloons----->Plates
      Support: 50%
      Confidence: 100%
```

Test 13: Minimum item confidence is not met, but attribute confidence is met.

Input

```
<transactions>
  <transaction id="1">
    <items>
      <item color="Blue"> Plates</item>
      <item color = "Blue"> Streamers</item>
    </items>
  </transaction>
  <transaction id="2">
    <items>
      <item color="Blue"> Plates</item>
      <item color = "Blue"> Streamers</item>
    </items>
  </transaction>
  <transaction id="3">
    <items>
      <item color = "Blue"> Plates</item>
      <item color = "Blue"> Streamers</item>
    </items>
  </transaction>
  <transaction id="4">
    <items>
      <item color="Red"> Plates</item>
    </items>
  </transaction>
</transactions>
```

Minimum Support: 0.75, Minimum Confidence 1
Minimum Attribute Support: 0.75, Minimum Attribute Confidence 0.75

Portion of Output beginning with Plates:

Notice that Plates → Streamers is not a rule, because no attributes are found in the rule.

```
**********RULES**********
Plates-----><color=Blue>Plates
      Support: 75%
      Confidence: 75%
Plates-----><color=Blue>Streamers
      Support: 75%
      Confidence: 75%
Plates-----><color=Blue>Plates, Streamers
      Support: 75%
      Confidence: 75%
Plates-----><color=Blue>Plates, <color=Blue>Streamers
      Support: 75%
      Confidence: 75%
```

Test 14: Mimimum item confidence is not met, and attribute confidence is not met.

Output:

```
<transactions>
  <transaction id="1">
    <items>
      <item color="Blue" size = "6" thickness="Thin"> Plates</item>
      <item color = "Blue" size="12"> Streamers</item>
    </items>
  </transaction>
  <transaction id="2">
    <items>
      <item color="Red" size = "6" thickness="Thin"> Plates</item>
      <item color="Blue" size = "6" thickness="Thin"> Plates</item>
      <item color = "Blue" size = "12"> Streamers</item>
      <item color = "Red" size = "9oz"> Cups</item>
    </items>
  </transaction>
  <transaction id="3">
    <items>
      <item color = "Blue" size = "9" thickness="Medium"> Plates</item>
      <item color = "Blue" size = "20"> Streamers </item>
      <item color = "Red" size = "Small" shape="Round"> Balloons</item>
    </items>
  </transaction>
  <transaction id="4">
    <items>
      <item color="white" size="20"> Streamers</item>
    </items>
  </transaction>
  <transaction id="5">
    <items>
      <item color="Red" size="Small" shape="Round"> Balloons</item>
    </items>
  </transaction>
</transactions>
```

Minimum Support: 0.8, Minimum Confidence 0.8
Minimum Attribute Support: 0.8 Minimum Attribute Confidence 0.8

Output:

**********RULES**********