International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 6 (2015) pp. 16129-16147

http://www.ripublication.com

An Efficient Association Rule Based Dynamic Support Count Adaptation Model for XML Databases Using XQuery

D Sathyanarayanan[1], M. Krishnamurthy[2]

[1]Research Scholar, AMET University, Chennai, Tamil Nadu, India.

[2]Professor Department of CSE, KCG College of Technology, Chennai, Tamil Nadu, India.

dsathya@rediffmail.com, mkrish@kcgcollege.com

Abstract

Rapidly increasing the XML technology usage for data storage and data transformation between the web applications is the necessity of mining XML data are raising quickly in this internet world. Hence, there is a need to consider the problem of Mining Association Rules between items in XML data. This paper propose a new algorithm that will generate the dynamic support count with less comparison and less calculations on time by introducing a new user preference database for mining the XML data. The principle purpose of this study is applying association rule algorithms directly to the XML data's using XQuery which is a functional expression language that can be used to query or process XML data. From the experiments conducted in this research work, it is proved that decision making with this algorithm is more efficient than the existing algorithms with respect to computation time and also in terms of provision of dynamic support count.

Keywords **-** Minimum Support Degree, Multidimensional Association Mining, User Preference Table, Dynamic Support.

1. INTRODUCTION

Data Mining is the discovery of hidden information found in large quantities of data and can be viewed as a step in the knowledge discovery process [15]. Data mining defined as a set of computer-assisted techniques designed to automatically mine large volumes of integrated data for new, hidden or unexpected information, or interesting patterns. With small set of data, traditional statistical analysis can be efficiently used. The first and simplest analytical step in data mining is to describe the data summarize its statistical attributes such as means and standard deviations, visually review it using charts and graphs, and look for potentially meaningful links among variables such as values that often occur together [16]. Data mining can be used in different kinds of databases (e.g. relational database, transactional database, object-oriented database and data warehouse) or other kinds of information repositories (e.g. spatial database, time-series database, text or multimedia database, legacy database and the World Wide Web) [4].

Association rule mining (ARM) has become one of the core data mining tasks and has attracted tremendous interest among data mining researchers. ARM is an undirected or unsupervised data mining technique which works on variable length data, and produces clear and understandable results. There are two dominant approaches for utilizing multiple processors that have emerged; distributed memory in which each processor has a private memory and shared memory in which all processors access common memory [5]. Shared memory

architecture has many desirable properties. Each processor has direct and equal access to all memory in the system. Parallel programs are easy to implement on such a system. In distributed memory architecture each processor has its own local memory that can only be accessed directly by that processor [10]. For a processor to have access to data in the local memory of another processor a copy of the desired data element must be sent from one processor to the other through message passing. XML data are used with the Optimized Distributed Association Rule Mining Algorithm.

The database researchers moves to XML (eXtensible Markup Language [20]) as an expressive and flexible hierarchical model suitable to represent huge amounts of data with no absolute and fixed schema, and with a possibly irregular and incomplete structure. Despite its impressive growth in popularity, XML is still lacking appropriate techniques to retrieve datasets available to casual users; such datasets, on one hand, have a limited or absent structure, and on the other hand contain a huge amount of data. Data mining, emerging during the late 1980s, has made great strides during the 1990s in transforming vast amounts of data into useful knowledge, and is expected to continue to nourish into the new millennium [10]. Nevertheless, compared to the fruitful achievements in mining well-structured data, mining in the semi-structured XML world still remains at a preliminary stage.

The XML document structure proposed by the authors to mine association rules over XML reflects simply the relational model of the Association Rule Mining problem. The set of transactions is identified by the tag <transactions> and each transaction in the transactions set is identified by the tag <transaction>. The set of items in each transaction is identified by the tag <items> and an item is identified by the tag <item>. A new dynamic support count based algorithm is implemented in a classical way: first an XQuery expression is used to create the set of frequent items then another XQuery expression is used to obtain the association rules from the frequent item sets using user preference table.

The rest of this paper is organized as follows: Section 2 provides a survey of related works. Section 3 depicts the architecture of the system proposed. Section 4 explains the proposed algorithms and mathematical analysis. Section 5 gives the results and discussion of the proposed work. Section 6 gives the conclusion and future enhancements of this proposed work.

2. LITERATURE SURVEY

Agarwal et al [1] introduced association rules for discovering regularities between products in large scale transaction data in supermarkets. Kamber et al. [3] discussed all data mining concepts with association rule mining. Zhang et al [8] provided a polynomial strategy to map the user specified fuzzy support threshold to an actual minimum support. More specifically, this algorithm automatically generates actual minimum-supports according to users mining requirements. Wu et al [7] deals to providing a user with a favorable support threshold suggestion from previous user are mining experience.

Liu et al [5] proposed a fuzzy matching technique. In this technique, the user is first asked to provide his expected patterns according to his past knowledge. Given these expectations, the system uses a fuzzy matching technique to match the discovered patterns against the user's expectations, and then rank the discovered patterns according to the matching results. Tan et al [6] described several key properties one should consider before deciding what is the right measure to use for a given application domain. Lenca et al [10] collected many measures with their properties and using a multi-criteria aid to suggest good measures that are suitable to the user specified requirements. Han et al [4] integrates both constraint-based and multidimensional mining into one framework provides an interactive, exploratory environment for effective and efficient data analysis and mining.

More recently the problem has been investigated also in the XML context [18, 19, 21]. In [21] the authors use XQuery [20] to extract association rules from simple XML documents. They propose a set of functions written only in XQuery which implement together the Apriori algorithm [2]. The same authors show in [21] that their approach performs well on simple XML documents; however it is very difficult to apply this proposal to complex XML documents with an irregular structure. This limitation has been overcome in [18], where the authors introduce a proposal to enrich XQuery with data mining and knowledge discovery capabilities, by introducing XMINE RULE, a specific operator for mining association rules for native XML documents. They formalize the syntax and an intuitive semantics for the operator and propose some examples of complex association rules. Wan and Dobbie presented a new native XML data mining approach in [17].The authors show that extracting association rules from XML documents without any preprocessing or post-

processing using XQuery is possible. They propose the XQuery implementation of the well-known Apriori algorithm.

Chit Nilar Win [11] propose that extracting association rules from XML documents without any preprocessing or post processing using XML query language XQuery is possible and analyze the XQuery implementation of the efficient FP-tree based mining method, FP-growth, for mining the complete set of frequent patterns by pattern fragment growth. FP-tree based mining adopts a pattern fragment growth method to avoid the costly generation of a large number of candidate sets and a partition-based, divide-and-conquer method is used. The authors shows how to use XQuery for native XML processing and data integration, briefly explores other technologies used in the same space, and discusses some XQuery extensions for scripting and updates that are under way [12]. Bhuvaneswari et al [13] proposed an improved framework for mining association rules from XML data using XQUERY. Authors [14] describe an integrated database architecture that enables SQL applications with XML extensions as well as XQuery applications to operate on the same data. The architecture allows for a seamless flow from relational data to XML and back.

M. Krishnamurthy et al (2011a) a new algorithm called Temporal Pattern Mining has been proposed to find the frequent temporal pattern based on Clustering, Bit Vector and Variable Threshold. Clustering and Bit Vector Association Rule by M. Krishnamurthy et al (2011b) for frequent itemsets generation uses single scan towards the database. M. Krishnamurthy et al (2010), an improved algorithm is proposed for generating frequent k-itemsets

3. PROPOSED MODEL

3.1 XML Data Extraction

Java provides excellent support for handling XML documents. Programs written in Java can access XML documents in one of the following two ways.

1) Document Object Model (DOM): This allows programs to randomly access any node in the XML document, and requires that the entire document is loaded into memory.
2) Simple API for XML (SAX): This approach follows event-driven model and allows programs to perform only sequential access on the XML document.

This does not load the entire document into memory. Since Apriori algorithm needs to scan the input data many times, we used DOM for implementing this algorithm. Similarly SAX is the natural choice for FP-Growth, since it needs to scan the input data only once and works with the FP-Tree constructed in memory for further processing.

An XML document contains one root level element with the corresponding opening and closing tags. These tags surround all other data content within the XML document. The format of a sample XML data used to test our algorithm is shown in Figure 1.

The transactions tag is the root element that contains many transaction elements. Each transaction element is uniquely identified by its id attribute. Each transaction element contains one items element which in turn contains many item elements. An item element has the name of the particular item in the given transaction. Note that the input XML document can have a very complicated structure, containing the transaction data at different depths. We assume in this case that the input document is preprocessed by using an XML style sheet language, like XSLT, to convert it into a simply structured document format as shown in Figure 1.

This preprocessing can be done quickly and easily, and is not in the scope of this paper.

```
<transactions>
<transaction id="1">
<items>
<item>Bread</item>
<item>Milk</item>
</items>
</transaction>
<transaction id="2">
<items>
...
</items>
</transaction> ...
</transactions>
```

Figure 1 XML Input File Format

The configuration setting for our implementation is given in Figure 2. These configurations are stored in a Java property file as property name-value pairs. The first four properties are self-explanatory. In order to make our implementation more generic in being able to work with any XML tag names, we allow the user to pass the name of these tags through the properties in lines 5 through 9.

```
INPUT FILE NAME=data.xml
OUTPUT FILE NAME=rules.xml
MINIMUM SUPPORT=0.6
MINIMUM CONFIDENCE=1.0
FIRSTLEVELELEMENT NAME=transactions
SECOND LEVEL ELEMENT NAME=transaction
THIRD LEVEL ELEMENT NAME=items
FOURTH LEVEL ELEMENT NAME=item
SECOND LEVEL ELEMENT UNIQUE
ATTRIBUTE NAME=id
```

Figure 2 XML Input File Format

Our implementation outputs the association rules in XML format as shown in Figure 3. The root level element name is a rule which contains zero or more rule elements. Each rule element has one antecedent and one consequent element, and each rule has two attributes: support and confidence.

```
<rules>
<rule support="0.6" confidence="1.0">
<antecedent>
<item>Bread</item>
</antecedent>
<consequent>
<item>Milk</item>
</consequent>
</rule>
...
</rules>
```

Figure 3 XML Output File Format

3.2 XPath and XQuery

Simple XQuery expressions are also valid XPath expressions. And XPath is a language for navigating through

the XML document. Loosely speaking, a path to XML parts is similar to the paths we get used to in our operation systems. For example, the path to the <address> tag looks as follows:

/kml/Response/Placemark/address

However, we need one more thing before this query will work with our example. The top node *<kml>* contains an attribute *xmlns="http://earth.google.com/kml/2.0"*. This attribute defines a namespace, which makes the elements and attributes unique. Since the namespace is present in the top element, we have to declare it and use it to specify all the elements (unless there is another namespace is introduced), so our query should look as follows (*see the attached simpleQueries.fmw*):

declare namespace x=http://earth.google.com/kml/2.0";

/x:kml/x:Response/x:Placement/x:address

A double slash "//" indicates that the query should look down the whole tree, not necessarily at the specified level, so the query can be maa bit simpler:

declare namespace x="http://earth.google.com/kml/2.0";
Both queries above will give us the following result on the original XML:
<address xmlns="http://earth.google.com/kml/2.0">7455 132 St, Surrey, BC, Canada</address>
There is another namespace in this XML defined at the <AddressDetails> level. This namespace specifies the unique names of the xAL. If we make a query that goes into this part of the XML, we should declare two namespaces and use them both, each at the appropriate levels:
declare namespace x="http://earth.google.com/kml/2.0";
Declare namespace y="urn: oasis: names: tc: ciq: xsdschema: xAL: 2.0";
/x:kml/x:Response/x:Placemark/y:Addressdetails/y:country/y:Countryname
or, if we don't use elements above the second namespace:
declare namespace y="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0";
// y: CountryName
Both queries above will give us the following result:
<CountryName xmlns = "urn:oasis:names:tc:ciq:xsdschema:xAL:2.0"> Canada</CountryName>
Of course, XQuery (and XPath) are much more powerful than simple path expressions they have a rich set of functions that help to extract and transform data. For example, we often don't need the XML tag, rather, we want to get the contents of the tags. The modified query #1 after extending it with a string() function will look as follows:

Table 1 Transaction Table

Tid	t_G		t_M		
	Cust ID	**Date**	**Age**	**Gender**	**Prod Name**
1	C001	12-01-09	20-30	Male	Bread, Butter
2	C002	12-13-09	40-50	Male	Bread, Jam
3	C003	01-10-10	40-50	Female	Ice Cream
4	C004	02-05-10	30-40	Male	Bread, Butter, Jam
5	C005	02-09-10	40-50	Female	Cake

```
declare namespace x=”http://earth.google.com/kml/2.0”;
String(/x:kml / x:Response/x:Placemark/x:address)
```

The result is:
7455 132 St, Surrey, BC, Canada

If we need a certain level of accuracy, we can explicitly indicate it like this:

```
Declare namespace x=”http://earth.google.com/kml/2.0”;
Declare namespace y=”urn:oasis:names:tc:ciq:xsdschema:xAL:2.0”;

For $n in // x: Placemark
  Where $n // y:Addresdetails:number(@Accuracy) = 3
Return data($n/x:Address)
```

And the result is:

Greater Vancouver regional district, british Columbia, Canada

3.3 Multidimensional Association Mining

A multidimensional association rule is an association rule that involves one or more dimensions (attributes). The main use of multiple dimensions is when the users can be allowed to specify their need more accurately[7].

The multidimensional association mining query is defined as follows:

$$\mathbf{MP: < t_G, t_M, [wc], ms, mc >}$$

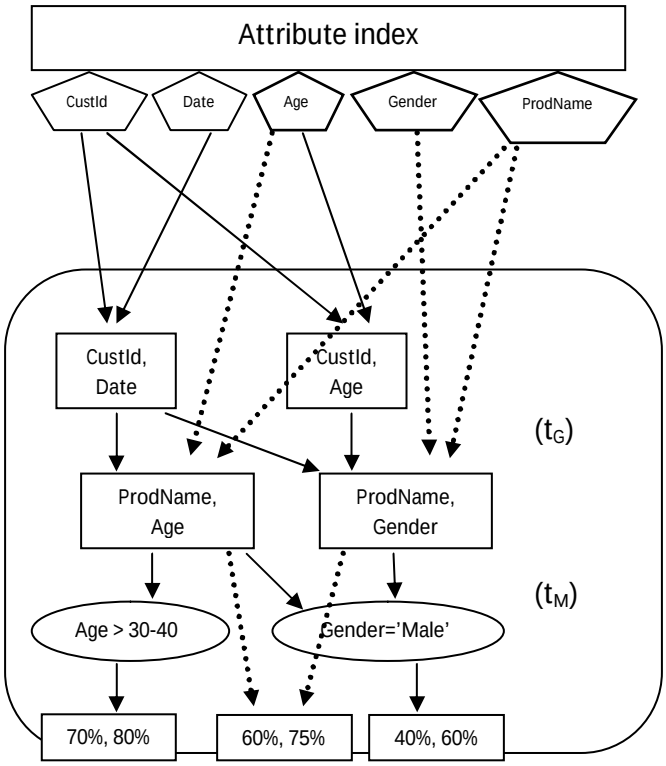
where t_G, t_M, wc, ms and mc are components of a query, described below:

t_G: the set of transaction,
t_M: the set of interested mining attributes,
wc: the optional “where” condition(s),
ms: the minimum support and
mc: the minimum confidence.

In Table 1, CustId, Date are mentioned as t_G (transaction) and Age, Gender, ProdName are mentioned as t_M (Mining attributes). This table is distributed based on the mining attributes.

3.4 The User Preference Table

In user preference ontology the attributes are related each other in the following Architecture diagram (Figure 4).



In Figure 4, clearly described the relation that, if the user submits the query that has t_G = {CustId, Date} and t_M= {ProdName, Age} and wc = Gender='Male' then the minimum support is 40% and minimum Confidence is 60%. If there is no where condition means then the minimum support is 60% and minimum Confidence is 75%. Like this in all cases the system extracts the correct minimum support and confidence from the user preference table.

3.5 Dynamic Support Count

For finding the dynamic support count, first we have to compare the user query with the ontology query[7]. The steps for calculating the dynamic support count are:

- Matching transaction ID (t_G)
- Matching interested mining attributes (t_M)
- Matching "where" condition (wc)

  (1) Attribute matching

  (2) Value matching
- Aggregating query similarity and suggesting

  the minimum support range.

A. Matching transaction ID (t_G)

Let G_a be the t_G formulated by users and G_b be the t_G from the user preference table, respectively. Suppose G_a = $\{g_{a1}, g_{a2}, \ldots, g_{ai}\}$ and G_b = $\{g_{b1}, g_{b2}, \ldots, g_{bj}\}$. And dist($g_{ax}$) and dist($g_{bw}$) denote the domain cardinality of attribute g_{ax} and g_{bw}, respectively, where $1 \le x \le i$, $1 \le w \le j$. The computation of the matching weight, T_G, of each g_{ax} and g_{bw} in G_a and G_b is defined as follows:

$$T_G(g_{ax},g_{bw}) \begin{cases} = 0 \text{ if } g_{ax},g_{bw} \text{ No hierarchical relationship} \\ \dfrac{\text{Min}(\text{dist}(g_{ax}),\text{dist}(g_{bw}))}{\text{Max}(\text{dist}(g_{ax}),\text{dist}(g_{bw}))} \quad \text{Otherwise} \end{cases} \quad (5.1)$$

The similarity of G_a and G_b, $\text{match}_G(G_a, G_b)$, is computed as follows:

$$\text{match}_G(G_a,G_b) = \sum_{x=1}^{i}\sum_{w=1}^{j} \frac{T_G(g_{ax},g_{bw})}{|G_a| \text{ X } |G_b|} \quad (5.2)$$

B. Matching Interested Mining Attributes (t_M)

Let M_a and M_b be the t_M formulated by a user and t_M from the user preference table, respectively. Suppose $M_a = \{m_{a1}, m_{a2},\ldots, m_{ai}\}$ and $M_b = \{m_{b1}, m_{b2}, \ldots m_{bj}\}$. The similarity of M_a and M_b is defined as follows:

$$\text{MatchM}(M_a,M_b) = \left| \frac{M_a \cap M_b}{M_a \cup M_b} \right| \quad (5.3)$$

C. Matching "where" condition (wc)

The format of a 'where' condition is (Attribute Name) Op (Value), (5.4)

Proposed Algorithm (Dynamic Support Count Based Algorithm)

Input: Database, User Query
Output: Minimum Support Range
begin
Take the first Query in the User preference Ontology
While (the User preference ontology is not empty)
begin
Value = strcmp(User Query, Current Query) // Compare t_G
if Value = 0 then
Ignore the Current Query and take the Next Query
Continue;
end if
Value = strcmp(User Query, Current Query) // Compare t_M
if Value = 0 then
Ignore the Current Query and take the Next Query
Continue;
end if
Value = strcmp(User Query, Current Query) // Compare wc
if Value = 0 then
Ignore the Current Query and take the Next Query
Continue;
end if
Take the Product of the three matching result
Apply temporal constraints on the results
Store the results in an array
end
Sort the support value of the array
Find the lower bound by taking the average of first half
Find the upper bound by taking the average of last half
if (lower bound > small) then // small = 1/number of transaction
Lower bound = small
if(higher bound > large) then // large = max. frequency of the 1 item set.
Upper bound = large
Return the lower and upper bound value
end

Where: Op $\in \{\leq, \geq, =, \neq\}$ is the relational operator.

For example, Where: Country ="India" There are two matching conditions in the "where" clause. They are attribute matching and value matching.

i) Attribute Matching

The attribute match of a "where" condition is computed in the same way as the matching of the interested mining attribute. Suppose WA_a is the set of attribute(s) in the wc formulated by users and WA_b is the set of attribute(s) in the wc from the user preference table respectively. Suppose $WA_a = \{wa_{a1}, wa_{a2},\ldots,wa_{ai}\}$ and $WA_b = \{wa_{b1}, wa_{b2}, \ldots, wa_{bj}\}$. The similarity of WA_a and WA_b is defined as follows:

$$Match_{WA}(WA_a, WA_b) = \left| \frac{WA_a \cap WA_b}{WA_a \cup WA_b} \right| \qquad (5.5)$$

ii) Value Matching

Let W_a be the wc formulated by the user and W_b be the wc from the user preference table. Suppose $W_a = \{wc_{a1}, wc_{a2},\ldots, wc_{ai}\}$ and $W_b = \{wc_{b1}, wc_{b2},\ldots, wc_{bj}\}$, where $wc_{ax} = wa_{ax}\ Op_{ax}\ v_{ax}$ and $wc_{bz} = wa_{bz}\ Op_{bz}\ v_{bz}$, $1 \leq x \leq i$, $1 \leq z \leq j$. wa_{ax} is the attribute, Op_{ax} is the relational operator while v_{ax} and v_{bz} denote the "value" of the conditional expression. To compute V, the matching degree of the "values" in conditional expressions, we consider the following cases:

Case 1: $wc_{ax} = wc_{bz}$, V = 1;
Case 2: $wa_{ax} \neq wa_{bz}$, V = 0;
Case 3: $wa_{ax} = wa_{bz}$ and ($Op_{ax} \neq Op_{bz}$ or $v_{ax} \neq v_{bz}$), V= CV;

The computation of CV, the complex conditional "value", is defined as follows:

$$CV(wc_{ax}, wc_{bz}) = \begin{cases} \dfrac{\mu - \ell}{dist(wa_{ax})} & \mu \geq \ell \\ 0 & \mu \leq \ell \end{cases} \qquad (5.6)$$

$dist(wa_{ax})$ denotes the counts of distinct values within wa_{ax}. The numerator is the overlapping degree of the "values" in the conditional expressions between wc_{ax} and wc_{bz}.

$$\mu = \max(Op_{ax}\ v_{ax}, Op_{bz}\ v_{bz})$$
$$\ell = \min(Op_{ax}\ v_{ax}, Op_{bz}\ v_{bz})$$

The values of μ and ℓ depend on the relational operators. Table 2 shows all cases of μ and ℓ while $wa_{ax} = wa_{bz}$

Table 2 All cases of μ and ℓ

Op1	**Op2**	$\boldsymbol{\mu}$	$\boldsymbol{\ell}$
$\leq$	$\leq$	$\min(v_{le}, v_{le})$	0
$\geq$	$\geq$	$dist(wa_{ax})$	$\max(v_{ge}, v_{ge})$
=	=	0	0
$\neq$	$\neq$	$dist(wa_{ax}) - 2$	0
=	$\neq$	0	0

≥	=	1 if $v_{eq} > v_{ge}$ 0 if $v_{eq} < v_{ge}$	0
≥	≠	$v_{ge} - 1$ if $v_{ne} > v_{ge}$ v_{ge} if $v_{ne} < v_{ge}$	v_{ge}
≤	=	1 if $v_{eq} < v_{le}$ 0 if $v_{eq} > v_{le}$	0
≤	≠	v_{le} if $v_{ne} > v_{le}$ $v_{le} - 1$ if $v_{ne} < v_{le}$	0
≤	≥	$v_{le} - v_{ge}$ if $v_{le} > v_{ge}$ 0 otherwise	0
Subscript notation of v: ≤ le, ≥ ge, = eq, ≠ ne			

v with an notation in the subscript denotes the "*value*" of a conditional expression that has the corresponding operator.
The matching degree of the conditional "*values*" is defined as follows:

$$\text{Match}_W(W_a, W_b) = \frac{\sum_{x=1}^{i}\sum_{z=1}^{j} V(wc_{ax}, wc_{bz})}{\text{Match}_{WA}(wa_{ax}, wa_{bz})}$$

3.6 Aggregating Query Similarity and Suggesting the Minimum Support Range

Let Q_a, Q_b be the user defined query and the queries in the user preference table respectively[7]. Suppose the t_G, t_M and wc of Q_a and Q_b are as follows:

$$Q_a = \langle t_G: G_a, t_M : M_a , wc: W_a \rangle$$
$$Q_b = \langle t_G: G_b, t_M : M_b , wc: W_b \rangle$$

The overall query similarity between Q_a, Q_b is defined as follows:

$$\text{QSimilarity}(Q_a, Q_b) = \text{Match}_G(G_a, G_b) \times \text{Match}_M(M_a, M_b) \times \text{Match}(W_a, W_b)$$

Through the similarity comparison of these queries, the top N similar queries from the user preference table are obtained. The final range of minimum support to be offered to the user is computed by the following steps:

Sort supports of the top N queries;
Average the smaller half of the N queries as the lower bound;
Average the bigger half of the N queries as the upper bound;

Validate the lower bound with the smallest possible value which is 1/n, where n is the count of the transactions after grouping. If the lower bound is beyond the smallest possible value, the smallest possible value becomes the lower bound;
Validate the upper bound with the largest possible value which is the maximum frequency among all the one-itemsets. If the upper bound is beyond the maximum possible support, the maximum possible support becomes the upper bound.

4. IMPLEMENTATION

4.1 Similarity Queries

For example Suppose, the inexperienced user want to set a minimum support for the multidimensional association mining, using the following table

Table 3 User Transaction Table

Tid	**t_G**		**t_M**		
	Cust ID	Date	Age	Gender	Prod Name
1	C001	1/12/09	20-30	Male	B,E
2	C002	13/12/09	40-50	Male	A,B,C,E
3	C003	10/1/10	40-50	Female	A,C,D
4	C004	5/2/10	30-40	Male	B,C,E
5	C005	9/3/10	40-50	Female	B,E
6	C001	15/4/10	20-30	Male	A,E

And the User XQuery (Q_a) format is like this:

t_G: CustId, Date (G_a)
t_M: prodName (M_a)
wc: Age > 30-40, Gender='Male' (W_a)
ms: 54%
mc: 80%

This XQuery is submitted to the system. Then the system compares the XQuery to already present queries (ie XQuery in the user preference table). First, it takes the first query in the user preference table and compares with the above query. Suppose the first table is look like this (Table 4).

Table 4 User Preference Table 1

Tid	**t_G**		**t_M**		
	Cust ID	Month	Age	City	Prod Name
1	C111	January	25-35	Chennai	A,B,C
2	C222	January	20-25	Chennai	A,C
3	C333	March	30-35	Chennai	B,C,D
4	C444	March	60-70	Chennai	C,D,F
5	C555	April	40-50	Chennai	A,B,D

and the XQuery (Qb) format is like this,

t_G: CustId, Month (G_b)

t_M: prodName, City (M_b)
wc: Age < 60-70 (W_b)
ms: 65%
mc: 86%

Then the comparison starts between the two queries and the result will be stored in the database in the form of array.

4.2 Matching G_a with G_b

The user XQuery of the transaction Id (G_a) and the table query of the transaction Id (G_b) are shown below:

G_a = {CustId, Date}
G_b = {CustId, Month}

Compare each G_a's attribute with G_b's attribute then,
CustId & CustId, Date & Month → has Hierarchical relationship.
CustId&Month,Date & CustId → has no Hierarchical relationship.

Dist(CustId) =5
Dist(Date) = 6
Dist(Month) =3

Then, $match_G$({CustId,Date}, {CustId,Month}) is:

$$\frac{\frac{\min(5,5)}{\max(5,5)} + 0 + 0 + \frac{\min(6,3)}{\max(6,3)}}{2 * 2} = 0.375$$

4.3 Matching M_a with M_b

The user XQuery of the Mining attribute (M_a) and the ontology query of the Mining attribute (M_b) is shown below:

M_a = { prodName}
M_b = { prodName, City}
Then, $match_M$({prodName}, {prodName, City}) is,

$$\left| \frac{\{prodName\} \cap \{prodName, City\}}{\{prodName\} \cup \{prodName, City\}} \right|$$

= 1/2 = 0.5

4.4 Matching W_a with W_b

The user XQuery of the Where condition (W_a) and the ontology query of the Where condition (W_b) is shown below:

W_a= {Age > 30-40, Gender='Male'}

W_b= {Age < 60-70}

A. Attribute Matching

First match the attributes of user query and ontology query.

Then, $match_W$({Age, Gender}, {Age}) is,

$$\left| \frac{\{ \text{Age, Gender} \} \cap \{ \text{Age} \}}{\{ \text{Age, Gender} \} \cup \{ \text{Age} \}} \right|$$

$= 1/2 = 0.5$

B. Value Matching

By Comparing W_a's Age condition with W_b's Age condition, it differs from the operator and value. In this matching, it satisfies the Case 3.Condition. Then compute CV.

Wc_{ax}= Age > 30-40

Wc_{bz}= Age < 60-70

Using the Table 2 get the μ and ℓ *value is*

$\mu = v_{le}\text{-}v_{ge}$ = (60-70) - (30-40)

μ = 3 and ℓ = 0

And Age has 3 distinct range values then,

CV(Age, Age) = (3-0)/3 = 1

And,

$Match_W$({Age, Gender}, {Age}) is,

$$= \frac{1 + 0}{0.5} = 2$$

C. Matching Q_a with Q_b

At last, find the XQuery similarity by taking the product of all comparisons. The final value is stored in the array.

QSimilarity (Q_a, Q_b) = 0.375 x 0.5 x 2 = 0.375

The above comparison result will be stored in the array. Like this, all similar queries values stored in the array. And the support values of the stored values are sorted. Finally take the lower and upper bound that will give the minimum support range of the user specified query.

4.6. Two Queries are not similar

Suppose if the queries are not matching, then the new algorithm ignore the query. But the old algorithm must compare with further comparison.

Suppose the comparison of the user XQuery with second XQuery (Q_b) in the user preference ontology as follows.

t_G: CustName (G_b)
t_M: prodName, Gender (M_b)
wc: State='Tamilnadu' (W_b)
ms: 54%
mc: 72%
and the table format is shown below (Table 5).

Table 5 User Preference Table II

Tid	**t_G**	**t_M**		
	CustName	Gender	State	ProdName
1	Kumar	Male	Tamilnadu	A,E
2	Mala	Female	Kerala	B,D
3	Raja	Male	Tamilnadu	C,E,F
4	Varun	Male	Andra	A,C,F
5	Lakshmi	Female	Tamilnadu	A,B,D

Now the t_G of the second query with the user XQuery:

G_a = {CustId, Date}
G_b = {CustName }

Compare each G_a's attribute with G_b's attribute then,

CustId & CustName, → Has no Hierarchical
Date & CustName → relationship

Because CustId is a unique one and CustName is not a unique. Therefore by comparing the t_G the result will be zero.

Then, $match_G$({CustId,Date}, {CustName}) is,

$$\frac{0 + 0}{2 \times 1} = 0$$

It makes the final result be zero. Therefore, no need for further comparison**.** In the case of the previous algorithm it will compare mining attribute (t_M) and also the where condition (wc). This is waste of time.

So this efficient algorithm leaves this query and takes the next query in the user preference table for comparison. This will reduce the time and memory.

5. EXPERIMENTAL RESULTS

By comparing the previous algorithm the proposed algorithm will reduce the comparison time and computation work and hence will be efficient. This will efficiently track the database and find the useful knowledge very quickly. The comparison chart between Efficient Ontology query and existing ontology query is shown in the Figure 5.

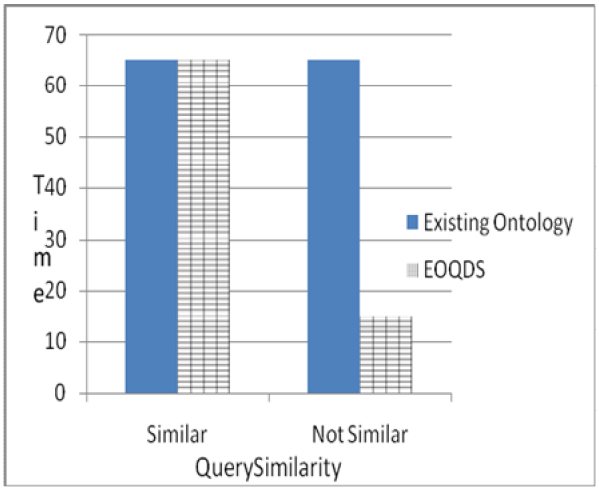


Figure 5 Comparisons between EOQDS and Existing Ontology

From Figure 5 shows the time analysis for DSCBA using similar queries with UPT and without UPT. From this Figure 5, it can be observed that provide better performance when it is compared without UPT in DSCBA.

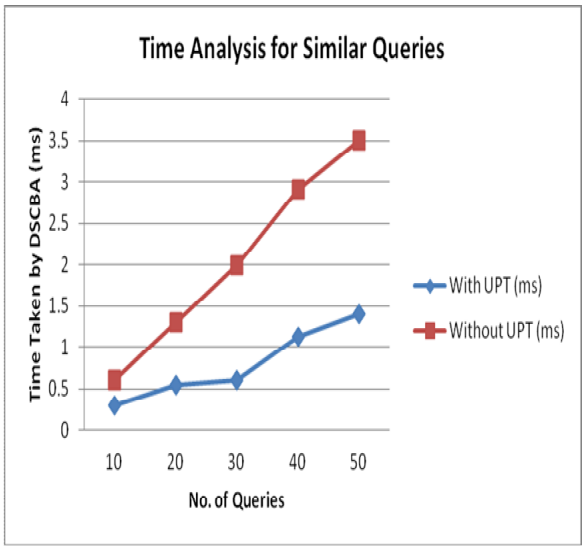


Figure 6 Time analysis for DSCBA using Similar Queries with UPT and without UPT

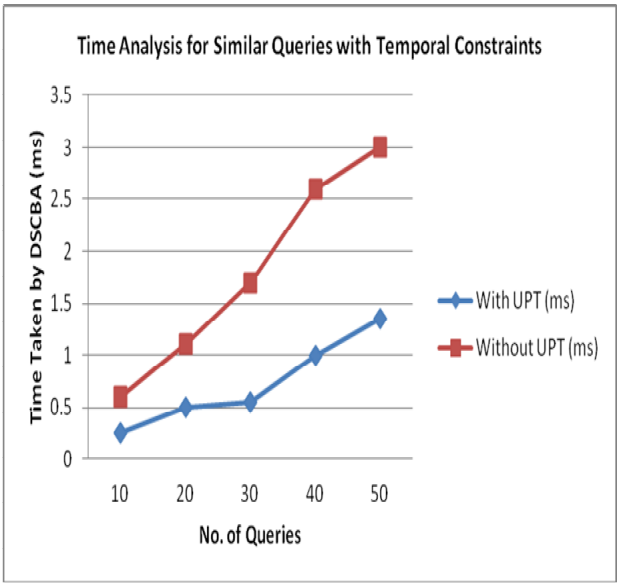


Figure 7 Time analysis for DSCBA using Non-Similar Queries with UPT and without UPT

From Figure 7 shows the time analysis for DSCBA using non-similar queries with UPT and without UPT. From this Figure 7, it can be observed that the performance of non-similar query execution with UPT is better when it is compared without UPT in DSCBA.

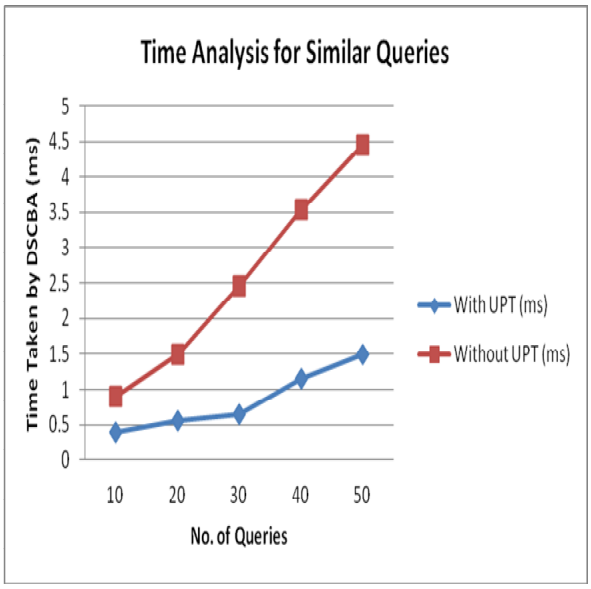


Figure 8 Time analysis for DSCBA using Similar Queries with Temporal Constraints in with UPT and without UPT

From Figure 8 shows the time analysis for DSCBA using similar queries with UPT and without UPT. From this Figure 8, it can be observed that the time analysis of similar query execution with UPT is better when it is compared without UPT in DSCBA.

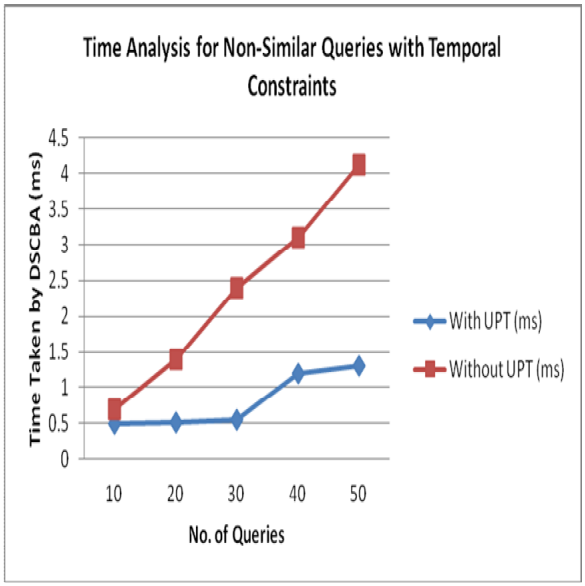


Figure 9 Time analysis for DSCBA using Non-Similar Queries with Temporal Constraints in with UPT and without UPT

From Figure 9 shows the time analysis for DSCBA using similar queries with UPT and without UPT. From this Figure 9, it can be observed that the time analysis of similar query execution with UPT is better when it is compared without UTP in DSCBA.

6. CONCLUSION

In this methodology, the system gives the minimum support range to the inexperienced user by comparing and extracting the most similar queries in the user preference table to the user-specified query, aggregates them and obtains the favorable support range for the user very quickly. This system helps the inexperienced people with respect to computation time. Hence it reduces the time and computation work.

REFERENCES

1. R. Agrawal, T. Imielinski, and A. Swami, “Mining Association Rules between Sets of Items in Large Databases,” in Proceedings of the ACM-SIGMOD International Conference on Management of Data, pp. 207-216, 1993.
2. R. Agrawal and R. Srikant, “Fast Algorithm for Mining Association Rules”, in Proceedings of International Conference of Very Large Databases, pp. 487-499, 1994.
3. J. Han and M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann, 2001.
4. J. Han, L.V.S. Lakshmanan, and R.T. Ng, “Constraint-based, multi-dimensional data mining,” IEEE Computer, Vol. 32, No. 8, pp. 46-50, 1999.
5. B. Liu, W. Hsu, L.F. Mun, and H.Y. Lee, “Finding Interesting Patterns using User Expectations,” IEEE Transaction on Knowledge and Data Engineering, Vol. 11, No. 9, pp. 817-831, 1999.
6. P. Tan, V. Kumar, and J. Srivastava, “Selecting the Right Interestingness Measure for Association Patterns”, in Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp.

32–41, 2002.
7. Chin-Ang Wu, Wen-Yang Lin, Chang-Long Jiang and Chuan-Chun Wu, "Favorable Support Threshold Recommendation for Multidimensional Association Mining Using User Preference Ontology," Granular Computing, 2009, GRC '09. IEEE International Conference, pp.586-591, 2009.
8. S. Zhang, X. Wu, C. Zhang, and J. Lu, "Computing the Minimum-Support for Mining Frequent Patterns," Knowledge and Information Systems, Vol. 15, No. 2, pp. 233-257, 2008.
9. H. Zhu, "On-line Analytical Mining of Association Rules", Master Thesis, SIMON FRASER University, 1998.
10. P. Lenca, P. Meyer, B. Vaillant, S. Lallich, "On Selecting Interestingness Measures for Association Rules: User Oriented Description and Multiple Criteria Decision Aid", European Journal of Operational Research, Vol.184, No. 2, pp. 610-626, 2008.
11. Chit Nilar Win, "Mining frequent patterns from XML data", Proceedings of 6th Asia-Pacific Symposium on Information and Telecommunication Technologies, pp. 208-212, 2005.
12. Robie, J. "XML Processing and Data Integration with XQuery", IEEE Journal of Internet Computing, Vol. 11, No.4, pp.62-67, 2007.
13. Bhuvaneswari V., Rajesh R, Amudha T, "An Improved Association Rule Mining Technique for XML Data Using XQuery and Apriori Algorithm", IEEE International Conference on Advanced Computing, pp. 1510-1514, 2009.
14. Funderburk J.E, Malaika S, Reinwald B, "XML Programming with SQL/XML and XQuery", IBM System Journal, Vol.41, No.4, pp. 642-665, 2002.
15. Fayyad, U., Shapiro, G. And Smyth, P. 1996. Data Mining and Knowledge Discovery in Databases: An overview. *Communications of ACM* 39(11):27-34.
16. Edelstein, H. 1998. *Introduction to Data Mining and Knowledge Discovery*. Two Crows Corporation 2nd Edition, ISBN 1892095009.
17. Wan, J. and Dobbie, G. 2003. Extracting association rules from XML documents using XQuery. *Proceedings of the 5th ACM international workshop on Web information and data management,* 94-97.
18. D. Braga, A. Campi, S. Ceri, M. Klemettinen, and P. Lanzi. Discovering interesting information in xml data with association rules. In SAC '03: Proceedings of the 2003 ACM symposium on Applied computing, pp. 450-454, New York, NY, USA, 2003. ACM Press.
19. J. Paik, H. Y. Youn, and U. M. Kim. A new method for mining association rules from a collection of xml documents. In Computational Science and its Applications ICCSA 2005, pp. 936-945, 2005.
20. World Wide Web Consortium. XQuery 1.0: An XML query language, 2007.
21. J. W. W. Wan and G. Dobbie. Extracting association rules from xml documents using xquery. In WIDM '03: Proceedings of the 5th ACM international workshop on Web information and data management, pp. 94-97, New York, NY, USA, 2003. ACM Press.
22. Krishnamurthy, M., Kannan, A., Baskaran, R. and Bhuvaneswari, G. "Hybrid Temporal Mining for Finding out Frequent Itemsets in Temporal Databases Using Clustering and Bit Vector Methods" Communications in Computer and Information Science, Springer,Vol. 141, No.5, pp. 245-255, 2011.
23. Krishnamurthy M, A. Kannan, R. Baskaran , and M. Kavitha ," Cluster based Bit Vector Mining Algorithm for Finding Frequent Itemsets in Temporal Databases", In Journal of Elsevier on Procedia ComputerScience , Vol. 3, pp. 513-523, 2011.
24. Krishnamurthy, M., Kannan., Baskaran, R. and Malini, V. "An Escalated Space Preservation Mining Algorithm for Finding Frequent Itemsets" International Journal of Computer Engineering, Vol.2 No.2, pp. 133–138, 2010.