# Intelligent Computation for Association Rule Mining

Hong-Cheu Liu
School of Economics and Information Systems
University of Wollongong
Wollongong, NSW 2522, Australia
hongcheu@uow.edu.au

John Zeleznikow
School of Information Systems
Victoria University
Melbourne, Vic. 8001 Australia
John.Zeleznikow@vu.edu.au

## Abstract

*Although there have been several encouraging attempts at developing SQL-based methods for data mining, simplicity and efficiency still remain significant impediments for further development. In this paper, we develop a fixpoint operator for computing frequent itemsets and demonstrate three query paradigm solutions for association rule mining that use the idea of least fixpoint computation. We consider the generate-and-test and the frequent-pattern growth approaches and propose an novel method to represent a frequent-pattern tree in an object-relational table and exploit a new join operator developed in the paper. The results of our research provide theoretical foundation for intelligent computation of association rules and could be useful for data mining query language design in the development of next generation of database management systems.*

## 1 Introduction

Knowledge discovery from large databases has gained popularity and its importance is well recognized. Most efforts have focused on developing novel algorithms and data structures to aid efficient computation of such rules. Much work has also been performed on data cleaning, preparation and transformation. While research into such procedural computation of association rules has been extensive, little object-relational technology has yet been significantly exploited in data mining even though data is often stored in (object)-relational databases.

Several encouraging attempts at developing methods for mining object-relational data have been proposed. In principle, we can express and implement association rule mining in conventional SQL language (transaction databases) or XQuery (XML data). This approach was examined by [4, 8], for instance. However, the resulting SQL (XQuery) code is less than intuitive, unnecessarily long and complicated. There has no relational optimization yet been exploited in their proposals. It was pointed out in the literature that current SQL systems are unable to compete with ad-hoc file processing algorithms in general purpose data mining systems such as the well known Apriori algorithm and its variants [7]. However most data is stored in (object-) relational database systems, it is meaningful to investigate intelligent computational methods for association rule mining by exploiting object-relational technology.

The integration of data mining functionality with database management systems is an essential component of advanced data retrieval and analysis applications. The main idea is to combine relational query languages with data mining primitives in an overall framework capable of specifying data mining tasks as object-relational queries. Logic-based database languages provide a flexible model of representing, maintaining and utilizing high-level knowledge. This motivates us to study a logic-based framework and develop relational operators (fixpoint and fp-join operators) for intelligent data analysis.

In this paper, we focus on computational methods from three paradigms that have been developed for querying relational databases. We demonstrate three paradigm solutions for association rule mining that use the idea of least fixpoint computation. We consider the generate-and-test and the frequent-pattern growth approaches and propose an novel method to represent a frequent-pattern tree in an object-relational table and exploit a new join operator developed in the paper. The results of our research provide theoretical foundation for intelligent computation of association rules and could be useful for the development of next generation of database management systems with data mining functionality.

The presentation of the paper is organized as follows. We briefly review the basic concepts in Section 2. In Section 3 we present three paradigms for data mining query languages. We develop a fixpoint operator for computing frequent itemsets and show how it is expressed in the three different paradigms. We then present datalog implementation for frequent itemset mining by using the frequent-pattern growth approach in Section 4. Finally we give a conclusion in Section 5.

## 2 Basic Concepts

In this section, we briefly review the basic concepts of association rule mining and data mining query languages.

### 2.1 Association Rules

While many forms of rule inductions are interesting, association rules were found to be appealing because of their simplicity and intuitiveness. In this paradigm, the rule mining process is divided into two distinct steps - discovering *large item sets* and generating rules.

The first work on mining association rules from large databases is the support-confidence framework established by Agrawal et al. [1]. Let $I = \{i_1, ... i_n\}$ be a set of item identifiers. An association rule is an implication of the form

$$X \Rightarrow Y, \text{ where } X, Y \subseteq I, \text{ and } X \cap Y = \emptyset$$

Association rules are characterized by two measures. The rule $A \Rightarrow B$ holds in the transaction set $D$ with support $s$, where $s$ is the percentage of transactions in $D$ that contain $A \cup B$. This is taken to be the probability, $P(A \cup B)$. The rule $A \Rightarrow B$ has confidence $c$ in the transaction set $D$ if $c$ is the percentage of transactions in $D$ containing A that also contain B. This is taken to be the conditional probability, $P(B \mid A)$. That is,

$$\begin{aligned} support(A \Rightarrow B) &= P(A \cup B) \\ confidence(A \Rightarrow B) &= P(B \mid A) \end{aligned}$$

The task of mining association rules is to generate all association rules that satisfy two user-defined threshold values: a minimum support and a minimum confidence.

### 2.2 Data Mining Query Languages

A desired feature of data mining systems is the ability to support ad hoc and interactive data mining in order to facilitate flexible and effective knowledge discovery. Data mining query languages can be designed to support such a feature [3]. In particular, declarative query language support acts an important role in the next generation of Web database systems with data mining functionality. Query systems should provide mechanism of obtaining, maintaining, representing and utilizing high level knowledge in a unified framework. A knowledge discovery support environment should be an integrated mining and querying system capable of representing domain knowledge, extracting useful knowledge and organizing in ontologies [2].

We will introduce three query language paradigms for association rule mining in the next section. The first paradigm is logic based. It is a variant of the complex value calculus. The second paradigm provides basic algebraic operations for manipulating (object-)relations to construct mining results to queries. It uses an aggregation operator in addition to the basic relational operations. The third paradigm stems from logic programming. We use Datalog$^{cv}$ with negation as a representative.

Designing a comprehensive data mining language is challenging because data mining covers a wide spectrum of tasks and each task has different requirements. In this paper we provide some theoretical foundations for relational computation of association rule mining.

## 3 Relational Computation for Association Rules

As most data is stored in (object-)relational databases, we can exploit object-relational technology to manage and mine interesting information from those data. In this section, we investigate relational computation methods and demonstrate three query paradigm solutions for association rule mining that use the idea of least fixpoint computation. The three query language paradigms, namely calculus, algebra and deductive rules, continue to play an important role in query languages of the next generation database systems.

### 3.1 Calculus+*Fixpoint*

We provide a noninflationary extension of the complex value calculus with recursion and aggregate operation. We define a fixpoint operator which allows the iteration of calculus formulas up to a fixpoint. In effect, this allows us to define frequent itemsets inductively using calculus formulas.

The motivation of defining a fixpoint operator in a data mining query language is to provide an alternative way to achieve association rule mining and to assist the development of a logic database language with data mining mechanisms for modeling extraction, representation and utilization of both induced and deduced knowledge.

**Definition 1** *Let $\mathcal{S}^k(V)$ denote the set of all degree-k subset of $V$. For any two sets S and s, s is said to be a degree-k subset of S if $s \in \mathcal{P}(S)$ and $|s| = k$. $\mathcal{P}(S)$ denotes the powerset of S.*

The noninflationary version of the fixpoint operator is presented as follows. Consider association rule mining from object-relational data. Suppose that raw data is first preprocessed to transform to an object-relational database. Let $D = (x, y)$ be a nested table in the mapped object-relational database. For example, $x = items, y = count$. *Items* is a set valued attribute. Let $S_x^k(D) = \{t \mid \exists u \in D, v = S^k(u[x]), t = (v,y)\}$. We develop a fixpoint operator for computing the frequent itemsets as follows. The relation $J_n$ holding the frequent itemsets with support value greater than a threshold $\delta$ can be defined inductively using the following formulas:

$$\varphi(T,k) = \sigma_{y\geq\delta}(\ _x\mathcal{G}_{sum(y)}\mathcal{S}_x^k(D)(x,y)) \longrightarrow T(x,y), \text{ if } k=1$$

$$\varphi(T,k) = T(x,y) \vee \sigma_{y\geq\delta}(_x\mathcal{G}_{sum(y)}(\exists u,v\{T(u,v) \wedge (\mathcal{S}_x^k(D)(x,y)) \wedge u \subset x \longrightarrow T(x,y)\})), \text{if } k \geq 1$$

as follows: $J_0 = \emptyset$; $J_n = \varphi(J_{n-1}, n), n > 0$. Where $\mathcal{G}$ is the aggregation operator. Here $\varphi(J_{n-1}, n)$ denotes the result of evaluating $\varphi(T,k)$ when the value of T is $J_{n-1}$ and the value of $k$ is n. Note that, for each input database $D$, and the support threshold $\delta$, the sequence $\{J_n\}_{n\geq 0}$ converges. That is, there exists some $k$ for which $J_k = J_j$ for every $j > k$. Clearly, $J_k$ holds the set of frequent itemsets of $D$. Thus the frequent itemsets can be defined as the limit of the forgoing sequence. Note that $J_k = \varphi(J_k, k+1)$, so $J_k$ is also a fixpoint of $\varphi(T,k)$. The relation $J_k$ thereby obtained is denoted by $\mu_T(\varphi(T,k))$. By definition, $\mu_T$ is an operator that produces a new nested relation (the fixpoint $J_k$) when applied to $\varphi(T,k)$.

In [6], the author proposed a fixpoint operator for computing frequent itemsets which is different from our definition. The least fixpoint operator of [6] is based on bottom-up computation approach which starts from the 'distance-1' subsets of the input database. We believe that our fixpoint operator is more appropriate as it can take advantage of anti-monotonicity property to do cross examination and make the computation method more efficient.

### 3.2 Algebra+*While*

Relational algebra is essentially a procedural language. The extension of the complex value algebra with recursion and incorporated with a *while* construct is consistent with the imperative paradigm and can express association rule mining queries.

We expect to have a function **sub** available in the next generation database systems that takes three arguments, two sets of values (Items) $V_1$ and $V_2$, and a natural number $k$ such that $|V_2| \leq k \leq |V_1|$, and returns the degree-$k$ subsets of the set $V_1$ that include $V_2$. We define a new join operator called *sub-join*.

**Definition 2** *Let us consider two relations with the same schemes* $\{Item, Count\}$. $r \bowtie^{sub,k} s = \{t \mid \exists u \in r, v \in s$ *such that* $u[Item] \subseteq v[Item] \wedge \exists t'$ *such that* $(u[Item] \subseteq t' \subseteq v[Item] \wedge |t'| = k), t = <t', v[Count]>\}$

Here, we treat the result of $r \bowtie^{sub,k} s$ as multiset meaning, as it may produce two tuples of $t'$ with the same support value. In the mining process we need to add all support values for each item.

**Example 1** *Given two relations r and s, the result of* $r \bowtie^{sub,2} s$ *is shown as follows.*

$r$

| *Items* | *Support* |
|---|---|
| $\{a\}$ | 0 |
| $\{b,f\}$ | 0 |
| $\{d,f\}$ | 0 |

$s$

| *Items* | *Support* |
|---|---|
| $\{a,b,c\}$ | 3 |
| $\{b,c,f\}$ | 4 |
| $\{d,e\}$ | 2 |

$r \bowtie^{sub,2} s$

| *Items* | *Support* |
|---|---|
| $\{a,b\}$ | 3 |
| $\{a,c\}$ | 3 |
| $\{b,f\}$ | 4 |

**Figure 1. An example of sub-join**

Given a database $D = (Item, Support)$ and support threshold $\delta$, the following fixpoint algorithm computes frequent itemset of $D$.

**Algorithm** *fixpoint*
Input: An object-relational database $D$ and support threshold $\delta$
Output: The frequent itemsets of $D$

```
begin
    $k := 1$
    $T := \sigma_{Support\geq\delta}(\ _{Item}\mathcal{G}_{sum(Support)}\mathcal{S}_{Item}^k(D)))$
    $P := \emptyset$
    $L := T$
    While $(L - P) \neq \emptyset$ do
    $P := L$
    $k := k+1$
    $T := \sigma_{Support\geq\delta}(\ _{Item}\mathcal{G}_{sum(Support)}(T \bowtie^{sub,k} (D))$
    $L := L \cup T$
    endwhile
end
```

**Example 2** *Let's look at an example of fixpoint algorithm, based on the transaction table, D, of figure 2.*

The figure 3 shows the computation steps of the *fixpoint* algorithm. It is easy to show that the above algorithm computes the fixpoint defined in the Calculus + $\mu$ language and hence the result below follows.

**Theorem 1** *For any object-relational database and minimum threshold* $\delta$*, the fixpoint defined in the association rule mining expressed in Calculus +* $\mu$ *and the fixpoint algorithm compute the identical frequent itemsets.*

$D$

| $TID$ | $item\_IDs$ |
|---|---|
| $T_1$ | $\{i_1,i_2,i_5\}$ |
| $T_2$ | $\{i_2,i_4\}$ |
| $T_3$ | $\{i_2,i_3\}$ |
| $T_4$ | $\{i_1,i_2,i_4\}$ |
| $T_5$ | $\{i_1,i_3\}$ |
| $T_6$ | $\{i_2,i_3\}$ |
| $T_7$ | $\{i_1,i_3\}$ |
| $T_8$ | $\{i_1,i_2,i_3,i_5\}$ |
| $T_9$ | $\{i_1,i_2,i_3\}$ |

**Figure 2. Transaction data**

$step1$

| $Items$ | $Support$ |
|---|---|
| $i_1$ | 6 |
| $i_2$ | 7 |
| $i_3$ | 6 |
| $i_4$ | 2 |
| $i_5$ | 2 |

$step2$

| $Items$ | $Support$ |
|---|---|
| $i_1$ | 6 |
| $i_2$ | 7 |
| $i_3$ | 6 |
| $i_4$ | 2 |
| $i_5$ | 2 |
| $\{i_1,i_2\}$ | 4 |
| $\{i_1,i_5\}$ | 2 |
| $\{i_1,i_3\}$ | 4 |
| $\{i_2,i_5\}$ | 2 |
| $\{i_2,i_4\}$ | 2 |
| $\{i_2,i_3\}$ | 4 |

$step3$

| $Items$ | $Support$ |
|---|---|
| $i_1$ | 6 |
| $i_2$ | 7 |
| $i_3$ | 6 |
| $i_4$ | 2 |
| $i_5$ | 2 |
| $\{i_1,i_2\}$ | 4 |
| $\{i_1,i_5\}$ | 2 |
| $\{i_1,i_3\}$ | 4 |
| $\{i_2,i_5\}$ | 2 |
| $\{i_2,i_4\}$ | 2 |
| $\{i_2,i_3\}$ | 4 |
| $\{i_1,i_2,i_5\}$ | 2 |
| $\{i_1,i_2,i_3\}$ | 2 |

**Figure 3. Computation steps of *fixpoint* algorithm**

Proof Sketch. The fixpoint operator in Calculus defines frequent itemsets inductively by using the basic definitions of the used-defined threshold and the aggregation operator. The *fixpoint* algorithm computes the frequent itemsets by performing the sub-join operation. This yields the result which is equivalent to the fixpoint defined in the Calculus.□

```
TABLE Corporation = (Doc-id, Sub-doc)
TABLE Products = (Product-id, prod-name, Warranty,
                  Composition, Distributor)
        Warranty = (premium, country, w-period)
        Composition = (Composition-id, c-name,
                       Component)
        Component = (part, quantity)
        Distributor = (company, fee)
TABLE Parts = (Part-id, part-name, weight, Warranty,
               Source)
        Warranty = (country, w-period)
        Source = (company, cost)
```

**Figure 4. Three mapped nested relational schemes.**

Consider the object-relational database shown in figure 4. An association rule describes regularities of component parts contained in products. For example, the rule $\{p_1,p_2,p_3\} \Rightarrow \{p_4\}$ states that if a product containing parts $\{p_1,p_2\ p_3\}$ is likely to also contain part $\{p_4\}$. We can apply the above *fixpoint* Algorithm to find frequent patterns and then generate such association rules.

## 3.3 Datalog$^{cv,\neg}$

In this section, we present an operational semantics for association rule mining queries expressed in Datalog$^{cv,\neg}$ program from fixpoint theory. The formal syntax and semantics of Datalog$^{cv,\neg}$ are straightforward extensions of those for Datalog$^{cv}$. A Datalog $^{cv,\neg}$ rule is an expression of the form $A \leftarrow L_1,...,L_n$, where $A$ is an atom and each $L_i$ is either an positive atom $B_i$ or a negated atom $\neg B_i$. A Datalog$^{cv,\neg}$ program is a nonempty finite set of Datalog$^{cv,\neg}$ rules.

The challenge is to develop declarative means of computing association rules so that we can mine interesting information from object-relational databases. It is difficult to cast inherent procedurality into the declarativity of logic-based systems.

We present a Datalog program as shown in the figure 5 which can compute the frequent itemsets. The rule 1 generates the set of *1-itemset* from the input frequency table. The rule 2 selects the frequent *1-itemset* whose support is greater than the threshold. Let us assume that we have a *sub-join* relation, where $sub\_join(J,I,k,x)$ is interpreted as '$x$ is obtained by applying **sub** function to two operands $J$ and $I$, i.e., $x = J \bowtie^{sub,k} I$. The rule 3 performs the *sub-join* operation on the table *large* generated in the rule 2 and the

```
1. $cand(J,C)$           $\leftarrow freq(I,C),\ J \subset I,\ |J| = 1$
2. $large(J,C)$          $\leftarrow cand(J,C), C > \delta$
3. $T(genid(),x,C_2)$    $\leftarrow large(J,C_1),\ freq(I,C_2),$
                         $k = max(|J|)+1, sub\_join(J,I,k,x)$
4. $cand(x, sum < C >)$  $\leftarrow T(id,x,C)$
5. $large(x,y)$          $\leftarrow cand(x,y), y > \delta$
```

**Figure 5. Deductive association rule mining program**

input frequency table.

Datalog system is of set semantics. In the above program, we treat $T$ facts as multisets, i.e., bag semantics, by using system generated *id* to simulate multiset operation. The rule 4 counts the sum total of all supports corresponding to each candidate item set generated in table $T$ so far. Finally, rule 5 computes the frequent itemsets by selecting the itemsets in the candidate set whose support is greater than the threshold. Suppose that $n$ is the maximum cardinality of the itemsets in the frequency table. The above program is bounded by $n$.

We now show the program that defines *sub-join*:

```
$to\_join(J,I)$         $\leftarrow A(J), B(I), J \subset I$
$sub\_join(J,I,k,x)$ $\leftarrow to\_join(J,I), J \subset I, x \subset I, |x| = k$
```

Once the frequent itemset table has been generated, we can easily apply the following rule, which was proposed in [5], to produce all association rules.

$$\begin{aligned} rules(I, J-I, support, conf) \leftarrow\ & large(I,C_I),\ large(J,C_J), \\ & support = C_J, \\ & conf = C_J/C_I,\ conf > \delta \end{aligned}$$

In the final step, the above generated rules will be represented in the output object-relational table.

## 4 The frequent-pattern growth approach

The frequent-pattern growth mining process consists of two steps [3]:

- Construct a compact frequent-pattern tree which retains the itemset association information in less space.
- Mine the FP-tree to find all frequent patterns recursively.

When the database is large, it is unrealistic to construct a main memory-based FP-tree. An interesting alternative is to store a FP-tree in an object-relational table. See Figure 6. The mining of the FP-tree proceeds as follows. Start from each frequent 1-itemset (as an initial suffix pattern), perform

*FP*

| *part* | *count* | pattern-base | |
|---|---|---|---|
| | | pattern | count |
| $p_5$ | 2 | $< p_2, p_1 >$ | 1 |
| | | $< p_2, p_1, p_3 >$ | 1 |
| $p_4$ | 2 | $< p_2, p_1 >$ | 1 |
| | | $< p_2 >$ | 1 |
| $p_3$ | 6 | ... | ... |
| $p_1$ | 6 | ... | ... |
| $p_2$ | 7 | ... | ... |

**Figure 6. An object-relational table representing FP-tree**

mining by applying a special kind of join, called fp-join which is defined below, on the pattern base attribute in the FP-tree table.

**Definition 3** *Given two arrays $a =< a_1,...,a_m >$ and $b =< b_1,...,b_n >$, where $m \leq n$, the join of two arrays is defined as $a \bowtie b$ =*

- *$< a_1,...,a_j >$, if ($a_1 = b_1$,...,$a_j = b_j$) and $a_{j+1} \neq b_{j+1}$ where $j < m$; or*
- *$< a_1,...,a_m >$, if $a_1 = b_1$,...,$a_m = b_m$*

For example, given two arrays $< i_2, i_1, i_5 >$ and $< i_2, i_1 >$, then $< i_2, i_1, i_5 > \bowtie < i_2, i_1 >$ = $< i_2, i_1 >$. Then we define fp-join for the conditional pattern base attribute in the FP-tree table.

**Definition 4** *Given two relations $u_1$ and $u_2$ with schemas $\{< pattern : array, count : integer >\}$, the fp-join of two relations is defined as follows:*

$$\begin{aligned} u_1 \bowtie^{fp} u_2 = \{t \mid\ & \exists t_1 \in u_1 \text{ and } t_2 \in u_2 \text{ such that} \\ & (t[pattern] = t_1[pattern] \bowtie t_2[pattern] \\ & \wedge t[count] = t_1[count] + t_2[count]) \\ & \vee (t \in u_1 \wedge (\forall t' \in u_2, t[pattern] \bowtie t'[pattern] = \emptyset) \\ & \vee (t \in u_2 \wedge (\forall t' \in u_1, t[pattern] \bowtie t'[pattern] = \emptyset) \end{aligned}$$

**Example 3** *Suppose there is a relation $R = \{<< i_2, i_1 >, 2 >, << i_2 >, 2 >, << i_1 >, 2 >\}$. $R \bowtie^{fp} R = \{<< i_2, i_1 >, 2 >, << i_2 >, 4 >, << i_1 >, 2 >\}$*

We present a Datalog program as shown in the figure 7 which can compute the frequent itemsets by using the FP-growth approach. Similar to the candidate generate-and-test approach, the rules 1 and 2 produce the frequent 1-itemset $L_1$. The rule 3 produces the prefix patterns for each item (i.e., part). The rule 4 counts the number of patterns for each prefix. The nest operator is applied to create nested schema $FP\text{-}base(J, C, pattern\text{-}base < K, PC >)$ in rule 5.

1. $freq(parts, count < company >)$
   $\leftarrow D(company, parts)$
2. $L_1(J,C)$
   $\leftarrow freq(I,C),\ J \subset I,\ |J| = 1,\ C > \delta$
3. $FP\text{-}pattern(J,C,T,K)$
   $\leftarrow L_1(J,C),\ D(T,I),\ J \subset I,\ K = I - J$
4. $FP\text{-}tree(J,C,K,count < T >)$
   $\leftarrow FP\text{-}pattern(J,C,T,K)$
5. $FP\text{-}base(J,C,pattern\text{-}base < K,PC >$
   $\leftarrow FP\text{-}tree(J,C,K,PC)$
6. $Cand\text{-}FP(J,C,CondFP < base, count >)$
   $\leftarrow FP\text{-}base(J,C,B),\ B \bowtie^{fp} B = CondFP$
7. $FP(I,PC)$
   $\leftarrow Cand\text{-}FP(J,C,CondFP < K,C >),$
   $Powerset(CondFP.K) \cup J = I,\ PC = C,\ C > \delta$
8. $FP(I,min(PC))$
   $\leftarrow FP(I,PC)$

**Figure 7. The FP-growth approach to frequent pattern mining**

The rule 6 applies the fp-join operator defined before to create the conditional pattern base, called *CondFP*. Finally, rules 7 and 8 form the frequent patterns by concatenating with the suffix pattern. In the program we use *Powerset* function which can be implemented in a sub-program and an aggregate function *min* to select the minimum support of the prefix patterns.

## 5 Conclusion

We have investigated data mining query languages from three paradigms that have been developed for querying relational databases. Three paradigm solutions for association rule mining that use the idea of least fixpoint computation have been demonstrated. In this paper, we have also shown that object-relational data can be mined in a declarative way so that extensive optimization task can be done in the underlying object-relational database engine. The main disadvantage of the deductive approach to data mining query languages is the concern of its performance. However, optimization techniques from deductive databases can be utilized and the most computationally intensive operations can be modularized. We have presented our preliminary ideas first and comprehensive query optimization and experimental work will be carried out at a later stage. The results of our research provide theoretical foundations for intelligent computation of association rules and could be useful for data mining query language design in the next generation of database systems.

## References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD conference on management of data*, pages 207–216, 1993.

[2] F. Giannotti, G. Manco, and F. Turini. Towards a logic query language for data mining. *Lecture Notes in Artificial Intelligence*, 2682:76–94, 2004.

[3] J. Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2000.

[4] H. M. Jamil. Ad hoc association rule mining as sql3 queries. In *Proceedings of international conference on data mining*, pages 609–612, 2001.

[5] H. M. Jamil. Mining first-order knowledge bases for association rules. In *Proceedings of 13th IEEE International conference on tools with Artificial intelligence*, 2001.

[6] H. M. Jamil. On the equivalence of top-down and bottom-up data mining in relational databases. In *Proceedings of the 4th international conference on data warehousing and knowledge discovery*, pages 41–50, 2001.

[7] D. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query flocks: A generalization of association-rule mining. In *Proceedings of ACM SIGMOD*, pages 1–12, 1998.

[8] J. W.W.Wan and G. Dobbie. Mining association rule from xml data using xquery. In *Proceedings of the Fifth International Workshop on Web Information and Data Management*, 2003.