# INCREMENTAL DEPENDENCY AIDED METAPATTERN GENERATOR

***Issam A.R. Moghrabi***
*M.I.S Department,*
*School of Business Administration,*
*Gulf University for Science and Technology, Kuwait*
*Gulf University for Science and Technology P.O. Box 7207 Hawally 32093 Kuwait*
*Email: Moughrabi.i@gust.edu.kw*

***ABSTRACT***

*Accumulation of data in electronic format is increasing at an exponential rate. Valuable transaction data will remain static and unexploited until analyzed to acquire knowledge. This work aims at enhancing a data mining technique called Metapattern generation. Domain knowledge is exploited in our approach and integrated with an existing Data Mining algorithm to generate interesting patterns that were not generated by the existing traditional techniques. Furthermore, new techniques are employed to enable the original algorithm to cope with incremental data.*

***Keywords:*** *Data mining, database systems, metapatterns, association rules.*

## 1.0 INTRODUCTION

Data mining, often referred to as Knowledge Discovery in Databases (KDD), deals with pattern recognition and creating knowledge models. It describes a multidisciplinary field of research that includes machine learning, statistics, database technology, rule based systems, neural networks, and visualization. Looking at the practical use of data mining and knowledge acquisition, the discovered data can be applied to information management, query processing, decision making, process control, and many other applications [2,4,11]. Several applications in information providing services, such as electronic libraries, on-line services, and World Wide Web have used mining to better understand user behavior so as to enhance the services provided and increase business opportunities.

In this work we will concentrate on a technique called Metapattern generation [8,9,12,13]. The standard Metapattern Generator algorithm is improved and considerably modified by integrating it with Induced Functional Dependencies, Join dependencies and foreign dependencies. This integration enhances the generation of transitive patterns and also makes the generation of non-transitive patterns possible.

## 2.0 THE ORIGINAL METAPATTERN GENERATOR

Classification algorithms require considerable user interference and lack proper knowledge representation as they generate structures that are hard to understand. Such algorithms are also time consuming. The new approach focuses on minimizing the user interference and generating interesting Value Sensitive rules [9,14,16]. Classification algorithms usually generate decision trees instead of implication rules. Although this kind of algorithms were applied and used in more than one domain and generated acceptable results, it has many drawbacks as it requires considerable user interference and lacks proper knowledge representation as it generates structures that are hard to understand. Such algorithms are also too time consuming.

The meta pattern generation algorithm extends applicability to intelligent retrieval planning and record/case clustering in record/case-based systems. To achieve this, we make use of the unsupervised learning and integrated knowledge discovery systems. Integrated knowledge discovery systems integrate Induction, Deduction, and human external knowledge in an iterative discovery loop [3, 4, 5, 6, 9,10].
Metapattern is proposed as a template or a second order expression in a language (*L)* that describes a type of pattern to be discovered. A metapattern is a generalization of similar patterns. It is expressed in form of predicates where predicate names generalize table names and predicate parameters generalize attribute names. For example the metapattern

$$P(X, Y) \wedge Q(Y, Z) \Rightarrow R(X, Z),$$

specifies that the patterns to be discovered are transitive. The result of executing a metapattern is a set of patterns whose left-hand sides are instantiated forms of the left-hand side of the metapattern, and whose right-hand sides are the results of the corresponding metapattern action.

In fact the more precise formulation of the above metapattern is:

$$P(X, Y) \wedge Q(Y, Z) \Rightarrow \text{CompStrength}(R(X, Z))$$

CompStrength(R (X, Z)) is an action that computes the strength of the pattern. Each pattern is evaluated against the database $U_{db}$ by two values:

The Strength value $P_S$, which is the probability of seeing the right hand side of *p* being true when the left-hand side of *p* is true. Based on "Laplace's Rule of succession" [1], the strength value of *p* can be computed as:

$$p_s = Prob(RHS|LHS, U_{db}, I_O) = (|S_{RHS}|+1) / (|S_{LHS}|+2)$$

The Base value $P_b$, which estimates how likely the left-hand side of *p* occurs in databases that have the same schema of $U_{db}$ is estimated by the following formula:

$$p_b = (|S_{LHS}|) / DOM(LHS)$$

where LHS represents the left-hand side of *p*, RHS the right-hand side of *p*, $S_{LHS}$ the set of tuples in $U_{db}$ that satisfy LHS, $S_{RHS}$ the set of tuples in $S_{LHS}$ that satisfy RHS, *DOM* (LHS) is the product of sizes of the tables that appear in LHS and $I_O$ is the assumption that the prior distribution of $S_{RHS}$ in $S_{LHS}$ is uniform. Intuitively, this $p_s$ value is the probability of seeing a tuple that satisfies RHS given the condition that the tuple satisfies LHS [12].

A pattern is said to be "interesting" only if its strength is within the user-specified threshold. It is said to be plausible if the base is above the user-specified threshold. When $p_s \geq s$ or $p_s \leq 1-s$, the pattern is accepted. Otherwise, if the base value is still above its threshold (i.e., $p_b \geq b$), then the pattern is considered plausible.

### 2.1 The Original Algorithm

The metapatterns are generated based on the data and the patterns. Humans will calibrate the generated metapatterns based on their external knowledge. The system should provide suggestions and feedback of metapatterns so that experts can discover new knowledge and try better metapatterns. For this purpose, the metapattern-based discovery loop was developed in [9,13,14,16].

The original metapattern generator looks for overlaps between attributes of different tables. The overlap is measured in terms of finding the ratio of the intersection of values in the underlying database of the two columns. Given two columns $C_x$ and $C_y$, the overlap is defined as follows [7,9]:

$$\text{Overlap}(C_x, C_y) = \max(|V_x \cap V_y| / |V_x|, |V_x \cap V_y| / |V_y|), \quad (1)$$

where $V_x$ and $V_y$ are the value sets of $C_x$, and $C_y$ respectively. If the computed overlap is greater or equal to the user defined threshold, then a reference is created for the two columns and inserted to a table called a Significant Connection Table (SCT). The SCT is then tested for finding cycles with alternating edges. Out of the found cycles the transitive metapatterns are generated. The algorithm can be outlined as follows:

**Procedure MetaPatternGenerator (D as DataBase, S as Schema, o, b, s as Threshold)**

*D is the underlying database*
*S is the schema of the database*
*o, b, and s are the user defined overlap, base and strength thresholds respectively*
For each two columns $C_x$ & $C_y$ from different tables

If $C_x$ & $C_y$ are of the same type then

OL = Overlap ($C_x$, $C_y$)

If OL >= o then
Add the term of these attributes to Significant Connection Table

End For

Find Cycles in the graph that can be generated from the Significant Connection Table.
Generate MetaPatterns from the generated cycles.

End.

Based on the above presentation, the algorithm for the pattern generation in the automated discovery loop can be outlined as follows:

***Procedure PatternGenerator*** **(s,b as threshold)**

M: = MetaPatternGenerator (D,S,o,b,s)

Loop
Order and Display M;
Let User examine, create, and reorder M;
Select *m* from M;
For each Pattern *p* instantiated from *m*;

Compute the strength value Ps and the base value $P_b$;
If $P_s$>= s or Ps <= 1-s then
Output (*p*);
Else if $P_b$>= b then
Select a set C of constraints for *p*;

Create new metapattern by adding the constraint to the left-hand side of *p*;
Insert the new mettapattern to M;
Until M is empty or the user aborts the process;

## 2.2 The Computational Complexity Of The Metapattern Generator

First, the computation cost of the overlap defined above is O ($p^2$), where *p* is the cardinality of the table. Since, we have $n^2$ overlaps to be computed, where *n* is the number of attributes, it follows that the time complexity of computing all overlaps is O ($p^2n^2$).

After generating the SCT or the graph, the algorithm looks for cycles in the graph. The graph generated has at most $n^2$ nodes. Finding cycles in the graph can be solved either using the Warshall's or Strassen's algorithms [18]. Warshall's algorithm solves the problem in O ($m^3$). Strassen's algorithm solves it in O ($m^{2.81}$), where m is the number of nodes [14,15].

In our case, the number of nodes in the graph is $n^2$, where *n* is the number of attributes. When substituting *m* for $n^2$, we obtain the time complexity of finding cycles in terms of attributes. Thus, the problem is solved in O ($n^{2*2.81}$) or O ($n^{5.62}$). This means that the computational complexity of generating metapatterns is O ($p^2n^2$) + O ($n^{5.62}$)

The maximum number of metapatterns that can be generated from *n* attributes is $n^2$ and the maximum length of a metapattern is $n^2$. From each metapattern of length $n^2$, we can generate $n^2$ patterns.

Each pattern of length L requires L joins to test its strength. The length of the pattern is the same as its metapattern length. This means that the pattern length L is $n^2$. Since the join operation is recursive over the whole pattern, the cost of these joins is $p^{n^2}$. The worst case has an order of $n^4p^{n^2}$.
Therefore, the computational complexity of whole algorithm is given by:

$O(p^2n^2) + O(n^{5.62}) + O(n^4p^{n^2}) = \text{Max}(O(n^4p^{n^2}), O(n^{5.62}))$

## 3.0 THE NEW ALGORITHM

As evident from our experimentation on the traditional metapattern generation technique and by tracing the steps of the algorithm in [14, 15], the original algorithm can be enhanced in both performance efficiency and in the quality of the generated metapatterns. The main observation is that the traditional algorithm does not make use of domain knowledge and is incapable of generating patterns over sub-domains. It only generates transitivity patterns that hold over the whole underlying domain. Furthermore, the algorithm in its present form is impractical partially because it is incapable of updating previously generated metapatterns. Another drawback is that the evaluation of the patterns against the underlying database mainly relies on carrying out joins among several tables.

By incorporating the domain knowledge and exploiting existing data dependencies, we are able to generate transitivity patterns, pseudo-transitive patterns, additive patterns and extended transitive patterns that the original algorithm is not able to generate.

Since patterns can be defined over sub-domains rather than over a whole domain, we make use of value sensitive dependencies and incorporate the idea of finding overlaps in sub domains to generate value sensitive or class based patterns.

Not only Domain knowledge is used in our new approach to generate previously non-generated patterns, but also it is used to deduce overlaps. Given the Foreign dependencies, we can deduce some overlapping attributes without computing the overlap. This is explained in details in section 3.1.

In the original algorithm, each pattern is evaluated against the underlying database. This evaluation can be achieved by joining tables instantiated in the pattern over the overlapping attributes. To minimize this cost, we have developed a new way that reduces vastly the high cost of joining tables with very large instances. The algorithm is detailed in the Appendix.

### 3.1 Deducing Overlaps From Functional Dependencies

Functional dependencies can be used to deduce Foreign dependencies or overlaps. Given two attributes $C_x$, and $C_y$ then the overlap between the two attributes is defined as follows:

$$\text{Overlap}(C_x, C_y) = \max(|V_x \cap V_y| / |V_x|, |V_x \cap V_y| / |V_y|),$$

where $V_x$ and $V_y$, are the value sets of $C_x$, and $C_y$ respectively.

The computation requires an equi-join between the two tables over the values $V_x$ and $V_y$. To avoid such joins, in some cases, we can make use of Domain Knowledge and functional dependencies.

To illustrate, we give the following example.
Let OL be the user defined threshold and given the relation R (A, B, C, D, E) with the following dependencies

$A \rightarrow B, C$

$AD \rightarrow E$

When normalizing, R will be decomposed into R1 (<u>A</u>, B, C) and R2 (<u>A, D</u>, E). Since in R2 A is a foreign key, we can state the following:

$$\text{ValueSet}(R2.A) \subseteq \text{ValueSet}(R1.A) \Rightarrow \text{ValueSet}(R2.A) \cap \text{ValueSet}(R1.A) = \text{ValueSet}(R2.A) \quad (1)$$

Let $C_x$ = R1.A and $C_y$ = R2.A and let $V_x$ = ValueSet (R1.A) and $V_y$ = ValueSet (R2.A). Substituting the values in this last equation we obtain $V_y \subseteq V_x \Rightarrow V_y \cap V_x = V_y$. Then, in this particular case of $V_y \cap V_x = V_y$, we can substitute

the deduced value in the overlap equation. Thus, the overlap between two columns $C_x$ and $C_y$ can now be computed as:

$$\text{Overlap}(C_x, C_y) = \max(|V_x \cap V_y| / |V_x|, |V_x \cap V_y| / |V_y|)$$

$$= \text{Max}(|V_y| / |V_x|, |V_y| / |V_y|)$$

$$= \text{Max}(|V_y| / |V_x|, 1)$$

$$= 1 \text{ since } (|V_y| / |V_x|) \leq 1$$

Since Overlap $(C_x, C_y)=1$ and $0 \leq OL \leq 1$ then Overlap $(C_x, C_y) \geq OL$, this means that in the case of foreign keys the overlap is always equal to 1, which is the maximum value an overlap can have.

### 3.2 Computation of A Pattern Strength

Given the pattern $T_1 (X_1, Y_1) \wedge T_2 (X_2, Y_2) \wedge \ldots \wedge T_n (X_n, Y_n) \rightarrow T_m (X_m, Y_m)$ and $Y_i$ overlaps $X_{i+1}$, then the only way to compute the strength and confidence of the LHS is to make a join between tables $T_1 \ldots T_n$, over the overlapping attributes. In essence, we are not interested at this stage in computing the strength using the resulting set of records, but rather interested in the count of the resulting set instead. Consequently, we have derived a simple way to compute $|S_{LHS}|$ such that the computational complexity is reduced.

The new algorithm will find the number of distinct occurrences of values of the column $T_1.Y_1$ and put the resulting tuples in a temporary table or view, which we call Tcnt1. The next step is to do a join between Tcnt1 and $T_2$ over Tcnt1.$Y_1$and $T_2.X_2$ and compute the resulting distinct occurrences of $T_2.Y_2$. The resulting tuples then overwrite Tcnt. This action is done iteratively until we reach $T_{n-1}$. On $T_n$, we do the same operation we did on $T_1$ and put the resulting set of records in Tcntn. Finally, for computing the cardinality of $S_{LHS}$ is done by computing the sum of the product from Tcnt1 and Tcntn.

The main enhancement in this algorithm is that we are always joining a relatively small table with a very large table. In other words, the number of tuples in Tcnt1 is always less or equal to the number of values in the domain of the $Y_i$ attribute. The cardinality of $S_{LHS}$ can thus be computed by the following algorithm (figure 1):

```
Function S_lhs (T1 (X1, Y1)^T2 (X2, Y2)^…^Tn (Xn, Yn)): integer
  Tcnt1, Tcntn, Tmp = table
  Begin
        If n = 1 then S_lhs = Select Count (*) from T1;
        Else
              Tcnt1 = Select Y1, Count (*) as Cnt1 From T1 Group by Y1
              Tcntn = Select Xn, Count (*) as Cntn From Tn Group by Xn
              For i =2 to n-1 do
                    Tmp=   Select Ti.Yi, Sum (Tcnt1.Cnt1) as Cnt1
                           From Tcnt1, TiWhere Tcnt1.Y1= Ti XiGroup by Ti.Yi
                    Tcnt1=Tmp
              End for
              S_lhs = select sum (Cnt1*Cntn) from Tcnt1, Tcntn where Tcnt1.Y1= Tcntn.Xn
        End if
  End
```

Function S_lhs ($T_1 (X_1, Y_1) \wedge T_2 (X_2, Y_2) \wedge \ldots \wedge T_n (X_n, Y_n)$): integer

Tcnt1, Tcntn, Tmp = table

Begin

If n = 1 then S_lhs = Select Count (*) from $T_1$;

```
Else
        Tcnt1 = Select Y1, Count (*) as Cnt1 From T1 Group by Y1
        Tcntn = Select Xn, Count (*) as Cntn From Tn Group by Xn
        For i =2 to n-1 do
            Tmp=  Select Ti.Yi, Sum (Tcnt1.Cnt1) as Cnt1
                  From Tcnt1, Ti Where Tcnt1.Y1= Ti Xi Group by Ti.Yi
            Tcnt1=Tmp
        End for
        S_lhs = select sum (Cnt1*Cntn) from Tcnt1, Tcntn where Tcnt1.Y1= Tcntn.Xn
End if  End
```

**Fig. 1:** Computing $|S_{LHS}|$ algorithm

For example, let us take the following transitive pattern generated from the database in figure 2. We will use the database example to illustrate the computation of the cardinality of the left-hand side in $T_3$ ($C_{31}$, $C_{32}$)$\wedge T_1$ ($C_{11}$, $C_{12}$)$\wedge T_4$ ($C_{43}$, $C_{43}$)$\rightarrow T_2$ ($C_{21}$, $C_{23}$)

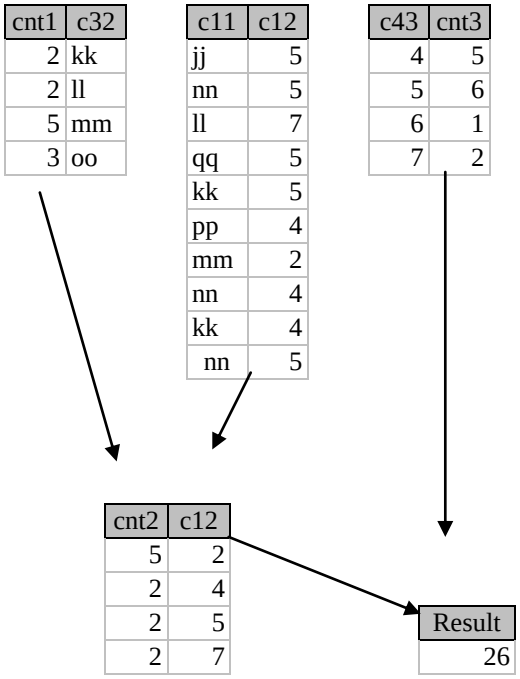


**Fig.2**: An example for computing $|S_{LHS}|$

### 3.2.1 The Computational Complexity

Let the table $T_i$ have $p_i$ tuples and the view Tcntn be defined as Tcntn = Select $X_i$, Count (*) as Cntr from $T_i$ group by $X_i$. Also let the number of tuples in Tcntr = r, where r<<p and the number of tuples in Tcnt1 = q where q <<p. Because of using a *Group By*, the data selected is sorted implicitly. Thus the complexity of the first select statement is O ($p_1 \log p_1$) and the second is O ($p_n \log p_n$). The cost of the select statement within the loop is O ($q_i p_i \log q_i p_i$). For simplicity, assume that $p = p_1 = p_2 = \ldots = p_n$ and $q = q_1 = q_2 = \ldots = q_n$ then the computational cost of the entire loop is O (nqplog qp). Also, the cost of the last select statement is O ($r_i q_n$). As a result, the time complexity of the whole algorithm is as follows:

$O(p_1 \log p_1)+O(p_n \log p_n)+O(nqp \log qp)+O(r_i q_n)$. This is $O(nqp \log qp)$.

In the worst case where q = p, the time complexity is $O(np^2 \log p^2) << O(p^n)$, the original time complexity of computing $|S_{LHS}|$.

### 3.3 Incremental Updating Of Discovered Patterns

In our approach, incremental updating techniques are developed for efficient maintenance of discovered rules in databases with data insertion. The major idea is to reuse the computed overlaps and strengths of each pattern. Our new techniques are described in the following sections.

#### 3.3.1 Computing the Overlap For Incremental Metapattern

Given two attributes $C_x$ and $C_y$ , the overlap between the two attributes is given by:

$$\text{Overlap}(C_x, C_y) = \max(|V_x \cap V_y| / |V_x|, |V_x \cap V_y| / |V_y|) \text{ [17,18],}$$

where $V_x$ and $V_y$, are the values of $C_x$ and $C_y$ respectively.

Let $V'_x$ be the values added to $C_x$.

Let $V'_y$ be the values added to $C_y$.

New Overlap ($C_x$, $C_y$) =

$$\text{Max}(|(V_x \cup V'_x) \cap (V_y \cup V'_y)| / |(V_x \cup V'_x)|, |(V_x \cup V'_x) \cap (V_y \cup V'_y)| / |(V_x \cup V'_x)|).$$

Computing the overlap for every set of updates is very expensive since $V_x$ and $V_y$ are very large sets of tuples. By using the distributive law:

$(V_x \cup V'_x) \cap (V_y \cup V'_y) = (V_x \cap V_y) \cup (V_x \cap V'_y) \cup (V_y \cap V'_x) \cup (V'_x \cap V'_y)$, and instead of computing $(V_x \cap V_y)$ after each set of updates, we will only compute $(V_x \cap V'_y)$, $(V_y \cap V'_x)$, and $(V'_x \cap V'_y)$. The first two intersections are between a large set and a small set. The last one is an intersection between two small sets. The main save here is that $(V_x \cap V_y)$ is computed just once.

### 3.4 Generating Dependency Aided Patterns

By introducing dependencies as one kind of domain knowledge, the algorithm will be able to generate new form of rules that the original algorithm was not able to generate. The dependencies will be used to guide the algorithm in the generation process. The dependencies that can be exploited are Functional Dependencies, Induced Dependencies, Value Sensitive Induced Dependencies, Weak Induced Dependencies [6], Join Dependencies [6].

By removing unimportant attributes, we are not only minimizing the number of attributes that the dependency generator will run over but also minimizing the possibility of generating uninteresting rules. Unimportant attributes are likely to result in rules with no predictive value.

Interesting patterns are not necessarily transitive and our approach seeks to uncover interesting hidden patterns, whether transitive or not, which the original algorithm fails to generate. In the following part, we discuss the new types of patterns generated. These are Pseudo transitive, Additive, Extended transitive and Relational.

The original algorithm has several drawbacks even when generating transitivity patterns. It uses no criteria except the overlap for the selection of the attributes. As a result, many attributes could be selected that will result in patterns that have no predictive value. We can overcome this weakness by selecting attributes using the interdependency measure.

Again, only attributes that are functionally interdependent are selected. And for consistency with Functional interdependency, foreign interdependency is used in our approach instead of overlap. The following is the definition of the two different kinds of interdependencies

***DEFINITION*- Foreign Interdependency**. *Let r be a nonempty (base or view) relation defined on R and let X and Y be subsets of attributes from R. A Foreign interdependency* X⇔Y *between X and Y exists in r if:*
X→Y (*k*) and Y → X (*k'*) are weak Foreign dependencies and *k* or *k'* is above the user-defined threshold

***DEFINITION*- Functional Interdependency**. *Let r be a nonempty (base or view) relation defined on R and let X and Y be subsets of attributes from R. A functional interdependency* X↔Y *between X and Y exists in r if:*
X→Y (*k*) and Y → X (*k'*) are weak functional dependencies and *k* or *k'* is above the user-defined threshold

In our approach, these two kinds of interdependencies are used in the generation process of both the transitive and non-transitive patterns that the original algorithm fails to generate.

The ***compare*** function that compares the values of the attributes and returns the operator that has the highest number of tuples is defined as follows:

***Function compare ($X_3$, $X_4$): relational operator***
*Max=0*
*For all relational operators*
*Op = operator*
*Select count (*) from $V_1$, $V_2$, $V_3$*
*Where $V_1.X_1 = V_2.X_1$ and $V_2.X_2 = V_3.X_2$*
*And $V_1.X_3$ Op $V_3.X_4$*

*If count (*)>Max then*
*Compare = Op*
*Max = count(*)*
*End If*
*End For*
*End.*

To illustrate, we consider the following table:

**Table 1** An Employee Database Example

| Emp ID | Education | Field | Position | Experience | Salary |
|---|---|---|---|---|---|
| 1 | Bac II | Math | Secretary | 1 | 500 |
| 2 | Bac II | Exp | Cash Reg | 2 | 600 |
| 3 | Bac II | Math | Cash Reg | 2 | 600 |
| 4 | Bac II | Phy | Show Rm | 3 | 650 |
| 5 | BS | CompSc | Programmer | 1 | 750 |
| 6 | BS | CompSc | Programmer | 2 | 850 |
| 7 | BA | Business | Sales | 3 | 1000 |
| 8 | MS | CompSc | Analyst | 2 | 1500 |
| 9 | MBA | Business | SalesMngr | 1 | 1250 |
| 10 | MBA | Intr Affairs | Comunic | 1 | 1000 |
| 11 | MS | CompSc | Proj Mngr | 4 | 2000 |
| 12 | PhD | CompSc | GM | 10 | 3500 |
| 13 | BacII | Phy | Sales | 7 | 700 |

From the example database in table 1, the *UpdateDependencies* function can find the following weak induced functional dependencies:

Position→Education [11/13] WFD-1

Position, Experience→ Salary [13/13] WFD-2
Education, Experience→ Salary [11/13] WFD-3

If the weak dependency threshold is less or equal to 11/13 then the above weak induced dependencies will be accepted. After updating SDT and CRT tables, the *Update (SCT)* can implicitly generate views to compute foreign dependencies or overlaps. The views $V_1$, $V_2$, $V_3$ correspond to the weak dependencies WFD-1, WFD-2, WFD-3 respectively.

$V_1$

| Education | Position |
|---|---|
| Bac II | Secretary |
| Bac II | Cash Reg |
| Bac II | Cash Reg |
| Bac II | Show Rm |
| BS | Programmer |
| BS | Programmer |
| BA | Sales |
| MS | Analyst |
| MBA | SalesMngr |
| MBA | Comunic |
| MS | Proj Mngr |
| PhD | GM |
| BacII | Sales |

$V_2$

| Position | Experience | Salary |
|---|---|---|
| Secretary | 1 | 500 |
| Cash Reg | 2 | 600 |
| Cash Reg | 2 | 600 |
| Show Rm | 3 | 650 |
| Programmer | 1 | 750 |
| Programmer | 2 | 850 |
| Sales | 3 | 1000 |
| Analyst | 2 | 1500 |
| SalesMngr | 1 | 1250 |
| Comunic | 1 | 1000 |
| Proj Mngr | 4 | 2000 |
| GM | 10 | 3500 |
| Sales | 7 | 700 |

$V_3$

| Education | Experience | Salary |
|---|---|---|
| Bac II | 1 | 500 |
| Bac II | 2 | 600 |
| Bac II | 2 | 600 |
| Bac II | 3 | 650 |
| BS | 1 | 750 |
| BS | 2 | 850 |
| BA | 3 | 1000 |
| MS | 2 | 1500 |
| MBA | 1 | 1250 |
| MBA | 1 | 1000 |
| MS | 4 | 2000 |
| PhD | 10 | 3500 |
| BacII | 7 | 700 |

**Fig.3**: Example for Database

From the views $V_1$, $V_2$, $V_3$ we have the following weak induced Foreign dependencies:

$V_1$.Position →$V_2$.Position [1]
$V_2$.Experience→ $V_3$.Experience [1]
$V_2$.Salary→ $V_3$.Salary [1]
$V_3$.Education→ $V_1$.Education [1]

From the above set weak functional and foreign dependencies we will deduce the following set of functional and foreign interdependencies. These interdependencies are inserted to the new SCT that can keep information about interdependencies as well as on dependencies.

$V_1$.Education↔ $V_1$.Position
$V_1$.Position⇔ $V_2$.Position
$V_2$.Position, Experience→ $V_2$.Salary
$V_2$.Experience⇔ $V_3$.Experience
$V_2$.Salary⇔ $V_3$.Salary
$V_3$.Education, Experience→ $V_3$.Salary
$V_3$.Education⇔ $V_1$.Education

Based on the definition of pseudo transitive Patterns and for the above set of interdependencies the

*Generate PseudoTransitive Metapatterns(DP)* will generate the following metapattern:

P (x, y)∧Q (y, z, w)→R (x, z, w) and finally the *GeneratePatterns(...)* will generate the following pseudo transitive patterns:

$V_1$ (Education, Position)∧$V_2$ (Position, Experience, Salary) → $V_3$ (Education, Experience, Salary)
$P_s$=17/19, $P_b$=19/169

$V_1$ (Education, Position)∧$V_3$ (Education, Experience, Salary) → $V_2$ (Position, Experience, Salary)
$P_s$=21/31, $P_b$=31/169

***Additive***

From the example database in fig. 2, the *Update Dependencies (D, S)* will find the following weak induced functional dependencies:

| | |
|---|---|
| Position→Education [11/13] | WFD-1 |
| Position→Experience [9/13] | WFD-2 |
| Education, Experience→ Salary [11/13] | WFD-3 |

If the weak dependency threshold is less or equal to 9/13 then the above weak induced dependencies will be accepted. After updating SDT and CRT tables, the *Update (SCT)* can implicitly generate views to compute foreign dependencies or overlaps. The vertical views $V_1$, $V_2$, $V_3$ correspond to the weak dependencies WFD-1, WFD-2, WFD-3 respectively.

The complete algorithm is presented in the Appendix. The following table illustrates a summarized comparison between the old and the new techniques where the merits of the new approach are clear:

| **Database (size)** | **Old metapatttern- number of generated rules (quality, time in sec)** | **Newmetapatttern number of generated rules (quality, time)** |
|---|---|---|
| Randomly generated (500 MB) | 42(50%, 4.4*60) | 68(89%, 2.24*60) |
| Randomly generated (990 MB) | 51(58%, 5.61*60) | 69(91%, 2.92*60) |
| National Bank of Kuwait mock database (13 GB) | 62(61%, 17*60) | 82(90%, 11.8*60) |

## 4.0 CONCLUSIONS

This paper has presented a robust and efficient approach for generating metapatterns that builds on and improves on existing techniques in terms of performance and the quality of the generated metapatterns. The original metapattern generator is modified to exploit dependencies, such as Functional Dependencies and Foreign Dependencies, to generate patterns other than transitivity patterns. Experimental results have shown that with this approach pseudo-transitive, additive, relational, and extended transitive patterns can be generated. This new form of patterns can now be generated by making use of domain knowledge and exploiting existing data dependencies.

Another contribution of this work is the notion of generating value sensitive patterns. Since patterns can be defined over sub-domains rather than over a whole domain, the new approach will make use of value sensitive dependencies and incorporate the idea of finding overlaps in sub-domains to generate value sensitive patterns. Furthermore, current algorithms are incapable of updating previously generated metapatterns. A simple, yet effective, technique

has been developed to update the set of previously generated patterns. Another drawback of the traditional current algorithm is that the evaluation of the patterns against the underlying database mainly relies on carrying out joins between several tables. This threatens seriously the practical feasibility of the technique as the database increases in size. This has been overcome by using a new technique for evaluating the patterns against the underlying database.

In the new algorithm, domain knowledge is not only used to generate previously non-generated patterns, but is also used to deduce overlaps. By deducing some of the overlapping attributes without computing the overlap we will improve the performance of the algorithm.

The selection of attributes is also a vital issue in inductive learning. In the traditional algorithm, the only criterion for selecting an attribute is to have an overlap with another attribute from a different table. This kind of selection of attributes can result in patterns with no predictive value. In this work, for an attribute to be selected, it should be interdependent with another attribute in addition to the overlap. Moreover, in our approach an overlap between two attributes of the same table is possible. This notion enables us to generate patterns from denormalized databases that the original algorithm failed to generate.

## REFERENCES


[1] R. Agrawal, and J. Shafer, "*Parallel Mining of Association Rules*," *IEEE Transactions on knowledge and Data Engineering*, Vol 8, No 6, December 1996, pp. 962-969.

[2] U. Fayyad, "Data Mining and Knowledge Discovery: Making Sense Out of Data.", IEEE Expert, October 1996.

[3] U. Fayyad, G. Piatetsky-Shapiro, and P.Smyth, " From Data Mining to Knowledge Discovery: An Overview," *Advances in knowledge Discovery and Data Mining*, MIT Press, Cambridge, Mass, 1996, pp. 1-36

[4] M. Holsheimer, A. Siebes, "Data Mining: The Search for Knowledge in Databases*.' Thesis Report, Computer Science Department*, Amesterdam, The Netherlands, 1994.

[5] IBM Corporation, "Data Management Solutions," *IBM's Data Mining Technology*, Stamford, Connecticut, April 1996.

[6] M. Kantola, H. Mannila, K. Raiha and H. Siirtola, "Discovering Functional and Inclusion Dependencies in Relational Databases," *Int'l J. Intelligent Systems*, Vol. 7, 1992, pp. 591-607.

[7] M. Kantola, H. Mannila, K. Raiha, "Design by Example: An Application of Armstrong Relations," *J. Computer Systems Science*, vol. 40, no 2, 1986, pp. 126-141.

[8] G. Mao; X. Wu; X. Zhu; G. Chen; C. Liu, "Mining maximal frequent itemsets from data streams", *Journal of Information Science*, 2007, Vol. 33 Issue 3, p251-262.

[9] M. Mehta, R. Agrawal, and J. Rissanen, "SLIQ: A Fast Scalable Classifier For Data Mining," *Proc. Int'l Conf. Extending Database Technology (EDBT' 96)*, Avignon, France, 1996, pp. 68-74.

[10] S. Ryszard "A Theory And Methodology Of Inductive Learning" *Machine Learning, an Artificial Intelligence approach*, Vol. 1. Morgan Kaufmann, San Mateo, California, 1983.

[11] Silberschatz, and A. Tuzhilin, "What Makes Patterns Interesting in Knowledge Discovery Systems," *IEEE* Transactions *on Knowledge and Data Engineering*, Vol. 8, no 6, Ded 1996, pp. 970-974.

[12] Wisse, P., "Metapattern: Context and Time in Information Models", 1st edition**,** Addison-Wesley**,** 2000, pp. 112-121.

[13] Wei-Min Shen, Bing Leng, "A Metapattern-Based Automated Discovery Loop For Integrated Data Mining --Unsupervised Learning of Relational Patterns," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, no 6, pp. 898-910, 1996.

[14] Wei-Min Shen, Bing Leng, "Metapattern Generation for Integrated Data Mining", *The 2nd International Conference on KDD*, Portland, Oregan, 1996.

[15] S. Yacout, M. Meshreki, H. Attia, "Monitoring and Control of Machining Process by Data Mining and Pattern Recognition", *Sixth International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, 2012, pp. 106-113.

[16] Yazici, and M. Sözat, "The Integrity Constraints for Similarity-Based Fuzzy Relational databases," *Int'l J. Intelligent Systems*, Vol. 13, 1998, pp. 641-659.

[17] Z. Zhao, J. Gao, H. Glotin, and X. Wu, "A Matrix Modular Neural Network Based on Task Decomposition with Subspace Division by Adaptive Affinity Propagation Clustering", *Applied Mathematical Modelling*, **34**(2010), 3884-3895.

[18] X. Wu, K. Yu, H. Wang, and W. Ding, "Online Streaming Feature Selection", *Proceedings of the 27th International Conference on Machine Learning (ICML 2010)*, Haifa, Israel, June 21-24, 2010, 1159-1166.

## BIOGRAPHY

Issam Moghrabi is a Professor of M.I.S/CS at GUST and is currently the Director of the M.B.A Program. He is a Fulbright Scholar and received several recognitions/awards for his achievements in research. He is a referee for several journals and serves on the editorial board of well-known publications. His main research interests are Database Systems, Nonlinear Programming, Data Mining and Information Retrieval.

## APPENDIX

## THE SYNTHESIZED ALGORITHM

Let us first make the following definitions:

*SDT = Sub-Domain Table (SD-Id, Table name, attribute name, Domain/Sub-domain)*

*CRT = Cardinality Table (SD1, SD2, card (SD1 ∩ SD2)) SD1&SD2 are of the same Type*

*SCT = Significant Connection Table*

*RFN = Reference Name*

*FD = Functional Dependencies*

*ID = Induced Dependencies*

*VSID = Value Sensitive Induced Dependencies*

*WID = Weak Induced Dependencies*

*JD = Join Dependencies*

**Procedure DataMiner (D as DataBase, S as Schema, OT, b, s as threshold)**

*D is the database instance*

*S is the table schemas and integrity constraints*

*OT the user defined overlap threshold*

*b is the user defined base threshold*

*s is the user defined strength threshold*

*The Dataminer is the main procedure that calls other procedures*

*It makes use of dependencies both functional and induced.*

*It looks for overlaps in sub domains that are in Sub-Domains Table (SDT).*

*SDT: is as defined above keeps entries of attribute sub-domains with overlaps.*

*DP1 is the set of defined functional and Foreign dependencies*

*DP2 is the set of induced functional and foreign dependencies.*

*M is the set of MetaPatterns*

Begin

DP1 = GetDependencies (D, S) //*reads dependencies defined at design time*

For each set of Updates

DP2=Update Dependencies (D, S)

DP=DP1 U DP2

Update CRT (D, SDT)

Update SCT (CRT, DP, OT)

Generate MetaPattern (SCT)

Generate Patterns (M, D, DP, b, s)

End for

End.

**Procedure Update Dependencies (D as DataBase, S as Schema, )**

*$T_l$ is the user-defined threshold of the number of attributes appearing on the LHS of a dependency.*

*$T_r$, $T_b$ are the ratio and base thresholds respectively of the value sensitive dependency*

*$T_c$ is the confidence or strength threshold of the weak dependencies*

*Each of the mentioned functions below will update the list of dependencies that reflect the under lying database after each set of updates. The functions will apply the definitions of the dependencies defined in previous sections.*

Begin

Update (ID, $T_l$) // *Updates Induced Dependencies with LHS length* $\leq T_l$

Update (VSID, $T_r$, $T_b$) // *Updates Value Sensitive Induced Dependencies and outputs those with ratio and base above $T_r$, $T_b$*

Update (WID, $T_c$) // *Updates Weak Induced Dependencies and outputs those with confidence above $T_c$*

Update (JD) // *Updates Join Dependencies*

Update (SDT)// *Updates this table with any new domains or sub-domains especially when new VSIDs are found*

End

**Procedure Update CRT (D as DataBase, SDT as table)**

*D is the database instance*

*SDT: is as defined above keeps entries of attribute sub-domains with overlaps.*

*Update CRT keeps an updated version of the intersection between any two sub-domains of the same type found in SDT. The purpose of this procedure is to keep the intersection between two sub-domains computed in previous runs and update it using the distributive law*

Begin

For each two sub-domains of the same type in SDT

Find Card (SD1 ∩ SD2)

If record not found then

Insert the record into CRT

Else

Update record in CRT

End If

End for

End

**Procedure Update SCT (CRT as table, DP as Dependencies, OT as Threshold)**

*CRT remembers the cardinality of the intersection between two sub-domains*

*DP is the set of dependencies*

*OT is the user defined Overlap Threshold*

*Update SCT updates the Significant Connection Table, which keeps entries of interdependencies and not only overlapping sub-domains.*

*It deduces overlaps from dependencies.*

*It computes the overlap only for promising sub-domains after each set of updates. See section* 3.3

*If no overlaps (foreign dependencys) were found between the whole domain of two attributes it searches for overlaps in the sub-domains. If patterns were then found for these attributes, Value Sensitive Patterns are then generated.*

Begin

For the changes in dependencies

Deduce Overlaps using ID and add an RFN to SCT for each

Deduce Overlaps add an RFN to SCT for each

Deduce Overlaps using JD and add an RFN to SCT for each

End For

For each two column sub-domains $C_x$& $C_y$ in SDT from different views

If $C_x$& $C_y$ are of the same type then

UL = Upper Limit ($C_x$, $C_y$). See section 3.3.

If UL>= OT then

OL =Overlap ($C_x$, $C_y$)

If OL >= OT and not in SCT then

Create RFN ($C_x$, $C_y$) and add to SCT

Else

```
                    Search for new sub-domains where OL>= OT
                    If any is found then
                        Create a new sub-domain entry in the SDT
                        Update CRT (D as DataBase, SDT as table)
                        Create RFN (Cx, Cy) and add to SCT
                End if
                End if
            End if
        End For
End.
```

**Procedure Generate MetaPattern (SCT as Table, DP as Dependencies)**

*SCT is the Significant Connection Table. (Interdependency Table)*

*DP is the set of dependencies.*

*Generate the metapatterns with the help of interdependencies stored in SCT.*

Begin

Generate Transitivity Metapattern(DP) // *Searches for transitivity patterns*

Generate PseudoTransitive Metapattern(DP) *Searches for Pseudo transitive*

Generate ExtendedTransitive Metapattern(DP) *Searches for Extended transitive patterns*

Generate Additive Metapattern(DP) *Searches for Additive patterns*

Generate Relational Metapattern(DP) *Searches for Relational patterns*

End

**Procedure Generate Patterns (M as MetaPatterns, D as DataBase, DP as Dependencies, b, s as Threshold)**

*M is the set of generated Metapatterns.*

*D is the database instance.*

*DP is the set of dependencies.*

*b is the user defined base threshold*

*s is the user defined strength threshold*

$P_s = Prob(RHS|LHS, U_{db}, I_o) = (|S_{RHS}|+1) / (|S_{LHS}|+2)$

$P_b = (|S_{LHS}|) / Dom(LHS)$

$S_{LHS}$ *is the set of tuples in* $U_{db}$ *that satisfy LHS.*

$S_{RHS}$ *is the set of tuples in* $S_{LHS}$ *that satisfy RHS.*

Loop

Order and Display M;

Let User examine, create, and reorder M;

Until M is empty or the user instructs to stop;

End.