OFGIM: A New Algorithm to Mine Generalized-Itemsets

Dr. Hussein K. Khafaji

Computer Communication Engineering Department
Email: dr.hkm1811@yahoo.com

Noora A. Mula

Computer Science Department
Email: noora_ahmed9@yahoo.com

Al-Rafidain University College

Abstract: *Most databases mined by a single layer/ crisp association rules algorithms are not flat but contain data in hierarchal/ generalized format. In spite of this fact, a few algorithms available to mine generalized itemsets to produce generalized association rules, GAR, which escort to mine more specific and concrete knowledge for decision makers. This research presents a new algorithm, (Optimized Frequent Generalized Itemsets Miner (OFGIM)), to mine generalized itemsets. Simply, it depends on extending the transactions of a database. The extension is done by adding the parent of an item to the transaction containing the item. The mining process is accomplished by the union of itemsets and the intersection of the tidsets. The algorithm requires two database scans only; the first one is for extending*

operation and the second scan is for mining process. The proposed algorithm does not need a specified data structure such as hash tree and prunes the apriori-based pruning steps.OFGJM was tested by using six synthetic databases. OFGJM overcomes apriori based algorithm in a ratio of ¼ in all the experiments, but it exhibitsits RAMappealing.

1. Introduction

All things by their nature can be classified. For examples clothes can be classified into winter clothes, summer clothes, sports clothes, formal dresses etc. Foods stuff can be classified into grains, fruit or drinks, etc. The computers can be

divided simply into portable and desktop computers, and each division can be divided into many divisions. Therefore, things in nature are governed by the relation 'is-a' used in artificial intelligence (AI) knowledge representation. For example, "Laptop is-a Portable" and "Portable is-a Computers". This is shown in Figure (1).

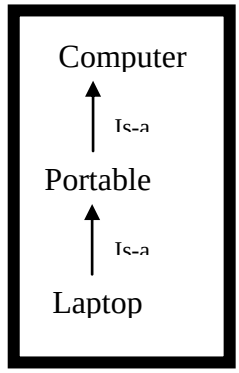


Figure (1) Is-a attribute

It should be mentioned that in AI literature the direction of the arrow is from subclass to supperclass and a label (is-a) is placed over it. While in Association Rules (AR) literature the direction of the arrow is in the reverse direction i.e. from supperclass to subclass. To make the classification clear and to elucidate the nature of the proposed algorithms, figure (2) shows the computer taxonomy and table(1) illustrates its relational database implantation.

Crisp Association Rules, CAR, deal with things and not with their classes [1, 2, 3]. For example you can conclude a rule or formula to predict that "20% of those who buy Laptop buy also Black/White printers". But decision makers need more general correlations from the rule, for instance, "50% of those who buy Portable computers buy Black/White printers" or "70% of those who buy Computers buy Black/White printers". This has led to the formulation of

Generalized Association Rules (GAR), depending on the hierarchy description of is-a, relation [4]. However, the support for the rule "Portable ⇒ColorPrinter" may not be the sum of the supports for the rules "Laptop ⇒ColorPrinter" and "IPad⇒ColorPrinter" since some people may have bought Laptop, IPad and ColorPrinter in the same transaction. Also, "Portable ⇒ColorPrinter" may be a valid rule, while "Laptop ⇒ColorPrinter" and "Computers ⇒ColorPrinter" may not. The former may not have minimum support, and the latter may not have minimum confidence.

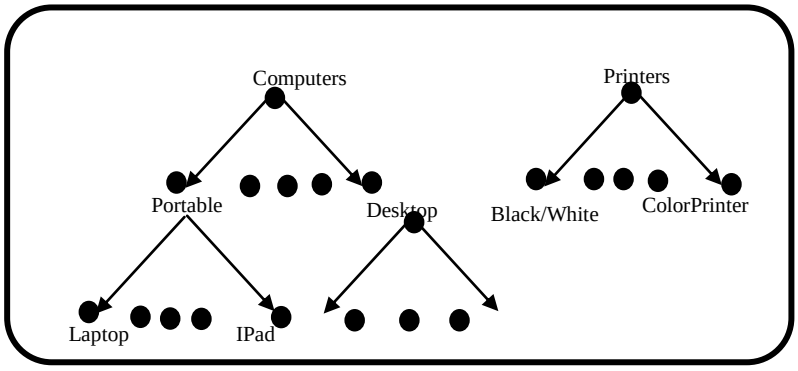


Figure (2) Example of Computers Taxonomy

Table(1) Relational Representation of Computers Taxonomy

Class	Super class (is-a)
Portable	Computers
Desktop	Computers
…	…
Laptop	Portable
IPad	Portable
…	…
Black/White	Printers
ColorPrinter	Printers
…	…

Earlier works on association rules [4, 5, 6, 7, 8] did not consider the presence of taxonomies and restricted the items in association rules to the leaf-level items in the taxonomy. However, finding rules across different levels of the taxonomy is valuable since[9]:

1) ***Rules*** at lower levels may not have minimum support. Few people may buy IPad with ColorPrinter, but many people may buy Portable computers with ColorPrinter. Thus many significant associations may not be discovered if we restrict rules to items at the leaves of the taxonomy. Since department stores or supermarkets typically have hundreds of thousands of items, the support for rules involving only leaf items tends to be extremely small.

2)*Taxonomies* can be used to prune uninteresting or redundant rules.

Multiple taxonomies may be presented. For example, there could be a taxonomy for the price of items (cheap, expensive, etc.), and another for the category. Multiple taxonomies may be modeled as a single taxonomy using a DAG (directed acyclic graph).

In this section, we introduce the problem of mining generalized association rules GARand explain the standard algorithm to this process.

This research presents a new algorithm to mine Generalized frequent Itemsets.

An obvious solution to the problem is to replace each transaction T with an "extended transaction" T', where T' contains all the items in T as well as all the ancestors of each items in T. for example, if the transaction contained Laptop, we would add Portable and Computers to get the extended-transaction. We can then run any of the algorithms for mining association ruleson the extended transactions to get generalized association rules. Some of the association rules mining algorithms are FARMA[1], GART[9], MARM[10] andQARMLDB[11]. All of these are apriori-based algorithms, therefore they inherits its drawbacks such as memory consumption, pruning steps, database multi scans, need for special data structure, i.e., hash tree. This research presents a new algorithm to overcome the mentioned drawbacks.

2. GAR Formal Description

Let $I=\{i_1,i_2,\ldots,i_m\}$ be a set of literals, called items. Let T be a directed acyclic graph, DAG, on the literals. An edge in T represents an is-a relationship, and T represents a set of taxonomies. If there is an edge in T from p to c, we call p a parent of c and c a child of p. (p represents a generalization of c.) We model the taxonomy as a DAG rather than a forest to allow multiple taxonomies.

We use lower case letters to denote items and upper case letters for sets of items (itemsets). We call ancestor(x) an ancestor of x,and x a descendant of ancestor(x) if there is an edge from ancestor(x) to x in the transitive-closure of T. Note that a node is not an ancestor of itself, since the graph is acyclic.

Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. (while we expect the items in T to be leaves in T, we do not require this). We say that a transaction T supports an item $x \in I$ if x is in T or x is an ancestor of some item in T. we say that a transaction T supports $X \subseteq I$ if T supports every item in X.

A generalized association rule is an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, $X \cap Y = \phi$, and no item in Y is an ancestor of any item in X. The rule $X \Rightarrow Y$ holds in the transaction set D with confidence c if c% of transactions in D that support X

also support Y. The rule X ⇒ Y has support s in the transaction set D if s% of transactions in D support X ∪ Y. The reason for the condition that no item in Y should be an ancestor of any item in X is that a rule of the form "x ⇒ ancestor (x)" is trivially true with 100% confidence, and hence redundant. These rules were called generalized association rules because both X and Y can contain items from any level of the taxonomy T[4].

Example (1)

Let I = {Printers, Black/White, ColorPrinter, Computers, Portable, Laptop, IPad, Desktop, etc.} and T the taxonomy show in Figure (2) and Table(1). Consider the database show in Table(2). Let minsup be 30% (that is, 2 transactions) and minconf 60%. Then the frequent itemsets are presented in table(3), and the rules corresponding to these itemsets are shown in Table(4). Note that the rules "IPad⇒ColorPrinter" and "Laptop ⇒ColorPrinter" do not have minimum support, but the rule "Portable ⇒ColorPrinter" does.

Table(2) Database

Transacti	Items Bought
1	Desktop
2	Laptop,
3	IPad,
4	Black/White
5	Black/White
6	Laptop

Table (3) Frequent Generalized Itemsets

Itemset	Support
{ Laptop }	2
{ Portable }	3
{ Computers }	4
{ Black/White }	2
{ ColorPrinter }	2
{ Printer }	4
{ Portable, ColorPrinter }	2
{ Computers, ColorPrinter	2
{ Portable, Printers }	2
{ Computers, Printers }	2

Table (4) Association Rules

Rule	Support	Conf.
Portable ⇒ColorPrinter	33%	66.6%
Portable ⇒ Printer	33%	66.6%
ColorPrinter⇒Portable	33%	100%
ColorPrinter⇒ Computers	33%	100%

Observation Let Pr(X) denote the probability that all the items in X are contained in a transaction. Then support (X ⇒Y) = Pr (X ∪ Y) and confidence (X ⇒ Y) = Pr (Y | X). (Note that Pr (X ∪ Y) is the probability that all the items in X ∪ Y are present in the transaction.)

If a set {x , y} has minimum support, so do {x , **ancestor(y)**}, { **ancestor(x)** , y} and { **ancestor(x)** , **ancestor(y)**}. However if the rule x ⇒ y has minimum support and confidence, only the rule x ⇒**ancestor(y)** is guaranteed to have both minimum support and confidence. While the rules **ancestor(x)**⇒ y and **ancestor(x)**⇒**ancestor(y)** will have minimum support, they may not have minimum confidence.

Note that, the support for an item in the taxonomy is not equal to the sum of the supports of its children, since

several of the children could be present in a single transaction. Hence we cannot directly infer rules about items at higher levels of the taxonomy from rules about the leaves.

3. Algorithm Basic

Consider the problem of deciding whether a transaction T supports an itemset X. If we take the raw transaction, this involves checking for each item x □X whether x or some descendant of x is present in the transaction. The task becomes much simpler if we first add all the ancestors of each item in T to T; let us call this extended transaction T'. Now T supports X if and only if T' is s superset of X. Hence a straight-forward way to find generalized association rules would be to run any of the algorithms for finding association rules on the extended transactions. We discuss below the generalization of the Apriori algorithm [5].

Figure (3) gives an overview of the algorithm, using the notation in Table (5). The first pass of the algorithm simply counts item occurrence to determine the frequent 1-itemsets. Note that items in the itemsets can come from the leaves of the taxonomy or from interior nodes.

Table (5) Notation of Algorithm Basic

k-itemset	An itemset having k items
L_k	Set of frequent k-itemsets (those with minimum support).
C_k	Set of candidate k-itemsets (potentially frequent itemsets).
T	Taxonomy

A subsequent pass, say pass k, consists of two phases. First, the frequent itemsets Lk-1 found in the (k-1)th pass are used to generate the candidate itemsetsCk, using the priori candidate generation function described in the next paragraph. The second phase, the database is scanned and the support of candidates in Ck is

counted. For fast counting, we need to efficiently determine the candidates in Ck that are contained in a given transaction t[5].

```
0 Begin {Basic}  (input D, output Answer)
1   L1:= {frequent 1-itemsets in D};
2   k := 2; // k represents the pass number
3   while( Lk-1≠ ϕ ) dobegin
4Ck:= New candidates of size k generated from Lk-1 .
5 For all transactions t ∈ D dobegin
6            Add all ancestors of each item in t to t, removing any duplicates.
7            Increment the count of all candidates in Ckthat are contained in t.
10         end
11Lk:= All candidates in Ckwith minimum support.
12k := k+1;
13end
14Answer :=∪kLk
15End; {Basic}
```

Figure (3) Algorithm Basic

Candidate Generation Given L_{k-1}, we want to generate a superset of all frequent k-itemsets. Candidates may include leaf-level items as well as interior nodes in the taxonomy. The intuition behind this procedure is that if an itemset X has minimum support, so do all subsets of X. For simplicity, assume the items in each itemset are kept sorted in lexicographic order. In the join step, towitemsets X and Y are joined, where $X=(x_1\ x_2\ x_3 \ldots x_{k-1}) \in L_{k-1}$ and $Y=(x_1\ x_2\ x_3 \ldots\ y_{k-1}) \in L_{k-1}$.

The result of the join is $Z= (x_1\ x_2\ x_3 \ldots\ y_{k-1}\ x_{k-1})$ if $x_{k-1} <= y_{k-1}$ or $Z= (x_1\ x_2\ x_3 \ldots\ y_{k-1}\ x_{k-1})$ otherwise. In the prune step, we delete all itemsets $c \in Z$ such that some (k-1)-subset of c is not in L_{k-1}.

4. Theoretical Manipulation of OFGIM

The proposed algorithm, OFGIM, depends on obeying the GAR mining to lattice theory and some optimization processes of candidates generating. This section elucidates this development.

4.1 Set and Lattice Theory

The database under association rules mining consists of two sets, the items set I and TIDs set T. The relation between these two sets decides attributes of the databases and shapes a bipartite graph[12]. The direction of the links of the bipartite graph is from item to transaction, which means "the item is member of transaction", i.e., the item is a subset of the transaction items. The "subset" relation forms partial ordered set over T and I.Also, it constructs a lattice from the frequent itemset in the database according to the following definitions [13].

Definition 4.1: Let P be a set, a partial order on P is a binary relation $\leq$,
such that for all X, Y, Z $\in$P, the relation is:

1) Reflexive X $\leq$X.
2) Anti-Symmetric: X $\leq$ Y and Y $\leq$ X, implies X=Y.
3) Transitive: X $\leq$ Y and Y$\leq$ *Z*, implies X $\leq$ Z.

The set P with the relation $\leq$ is called an **ordered set** and it is denoted as a pair (P,$\leq$).

Definition4.2: Let P be an ordered set, and let S $\subseteq$ P. An element X $\in$P is
an **upper bound** of S if s $\leq$. X for all s $\in$S. The least upper bound, also
called **meet,** of S is denoted as $\vee$S.

Definition 4.3: Let P be an ordered set, and let S $\subseteq$ P. An element X $\in$P be a lower bound of S if s $\geq$ X for all s $\in$S. The greatest lower bound, also called join, of S is denoted as $\wedge$S. If S={x, y}, then the meet is written as x$\vee$y, and the join is written as x $\wedge$y.

Definition 4.4: Let P be an ordered set. The greatest element of P, denoted
as$\lceil$, is called **top element,** and the least element of P, denoted as $\rfloor$, is called the **bottom element.**

Definition 4.5: An ordered set (L, ≤) is called lattice, if for any two elements x and y in L, the meet and join of x and y always exist. L is a complete lattice if ∨S and ∧Sexist for all S ⊆ L. Any finite lattice is complete. L is called meet semi-lattice if only the meet exists. L is called a join semi-lattice if only the join exists.

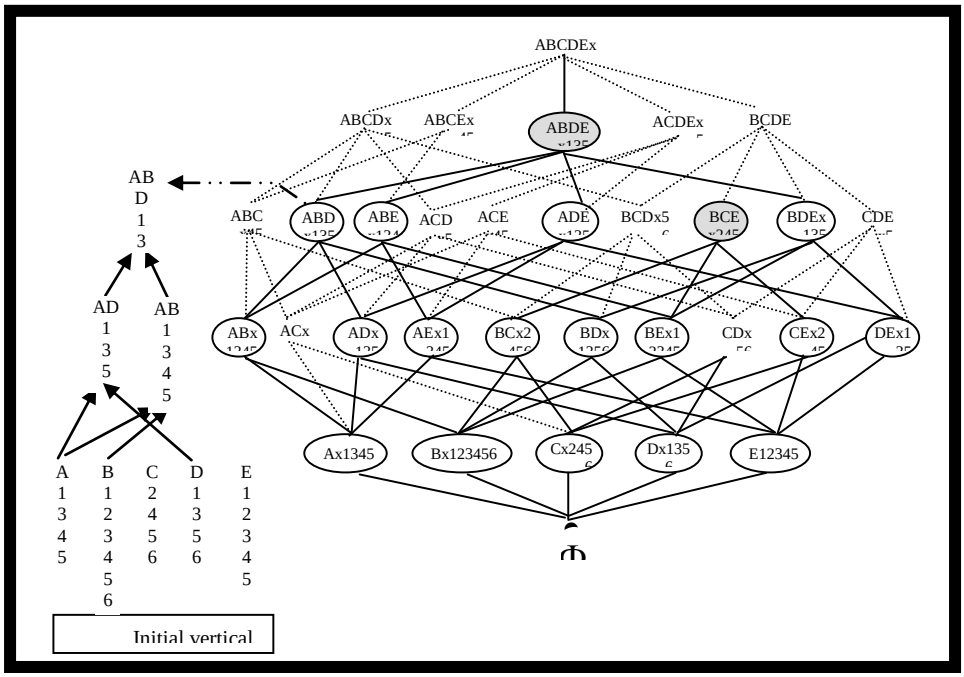


Figure (4) Join semi-lattice of frequentitemsets

To give more explanation and to tame the rule mining problem to the definitions above, let us return to example (1), where the set of frequent items is I_f= {A, B, C, D, E} and T={ 1,2,3,4,5,6}. Let P (I_f) denote the set of all subsets of I_f, (called power set of I_f) and P(T) is the power set of T. The partial orders (P(I_f),⊆) the set of all possible itemsets and (P(T),⊆) the set of all possible tid sets are both complete lattices where the join is given by set intersection and meet is given by set union. The ordered set (P(I_f),⊆) and (P(T),⊆) is represented in figure (4). Suppose a set S⊆P(I_f) ={ABD, ABE, ADE}. Then the itemsets***abcde*** is the upper bound of S and the empty set {} is the lower bound and the join of S.

The meet of S is ***abcde.*** The top element of $(P(I_f),\subseteq)$ is abcde and the bottom element is {}. The set of all frequent itemsets is a join semi-lattice, consider Figure (4), which depicts the join of all frequent itemsets extracted in example (1). The join of any two frequent itemsets is frequent, but their meet may or may not be frequent. This phenomenon is consistent with the apriori-fact that ***"all the subsets of a frequent itemset are frequent too".***

In figure (4), the join of the itemsets AC and AE is A which is frequent, $AC \curlywedge AE = AC \cap AE = A$, but the meet $AC \vee AE = AC \cup AE = ACE$ is not frequent. On the other hand, the meet $AB \vee AD = AB \cup AD = ABE$ is frequent. As mentioned previously, there are two maximal itemsets in example (1), *ABDE* and *BCE* that are shown in grayed circle; therefore, each of them is a top element of the semi graph.

Note that the dashed lines of figure (4) reach infrequent itemset from frequent or infrequent itemsets, while bold lines connect frequent itemsets. Frequent itemsets are enclosed in circles.

According to this taming of association rules mining to the graph theory, any (k+1)-itemset can be obtained from the union of any of two its k-itemsets subsets, also its tidlist can be obtained by intersection the tidlists of these two k-itemsets. Therefore, the proposed algorithm will convert the mining process to sequence of union and intersection operations.

4.2 Optimized Features of OFGIM

The second pillar in OFGIM is several optimizations, therefore it is named OFGIM. OFGIM algorithm depends on the following optimizations:

1- The number of ancestors added to transaction is reduced; in OFGIM not all ancestors of an item are added to transaction, T, which includes it. Instead, ancestors which were found in candidates itemsets at one stage. In fact, if the original item is not in any of the itemsets, it can be dropped from the transaction. For example, assume the parent of "Portable" is "Computers", Let {Computers, Black/White} be the only itemset being

counted, then in any transaction containing Laptop, we simply replace Laptop by Computers. We do not need to keep Laptop in the transaction, nor do we need to add Portable to the transaction.

2- Pre-computing ancestors; OFGIM does not find ancestors through search in the classification database. OFGIM deletes ancestors who are not already mentioned in candidate at a certain level.

3- Pruning of an itemset which includes an item and its ancestor according to the following logical observations:-

Observation(1): support itemset U which contains Item u and its ancestor ū which will be equal to itemset U- ū.

Observation(2): If L_k is a set of frequent k-itemsets which does not contain any itemsetthat contains Item u and the ancestor of U, the candidate set C_{k+1} generatedby candidate generation algorithm does not contain any itemset containing the ancestor of U.

```
0 Optimization (Lk),
1 Begin
2       If k=2
3           Delete l2 consist of an item and its ancestor; // optimization 3
4       Delete any ancestors in D' that are not presented in lk// optimization 1
5  End;
```

Figure (5) OFGIM Optimization Process.

These observations are useful in keeping away 2-itemsets generated from an item and its ancestor.

Observation (1) shows that we need not to count any itemset, which contains both an item and its ancestor. We add this optimization by pruning the candidate itemsets of size two, which consist of an item and its ancestor.

Observation(2)shows that pruning these candidates is sufficient to ensure that we never generate candidates in subsequent passes, which contain both an item and its ancestor.

Figure (5) presents two optimizations process for generalized itemset; optimization 3 and 1, while the second optimization occurs during the extending of the database. Optimization 2 happened automatically because the suggested algorithm depends on vertical layout. Vertical layout inserts records for each ancestor with its tid list and does not extend the transactions of the database. The vertical extending of the database is accomplished by a sub algorithm called Data Extend Sub algorithm, DESA for GAR, which presented in Figure (6). Step#2 of DESA for GAR copies frequent items, frequent leaf to D'. Then, it opens an entry in D' for each parent in taxonomic structure. When an item is fetched, its tid list is added to the tid lists of all its ancestors and then the support of the items is accumulated with the length of the tid list. For example, (recall Table 2), when Laptop is fetched its tid list, i.e., {200, 600} are added to the tid list of its ancestors that are Portable and Computers. For full illustration, Table (6) presents the vertical extended database of Table(2) with *minsup* equals 2.

```
 0 DESA for GAR (input D, output D');
1 begin
2   copy frequent items of D to D';
3    for all i x tid(i) ∈ D do
4     begin
5        for all ancestor(i) do begin
6 if ancestor(i) has no entry in D' open an entry for ancestor(i);
7               support(ancestor(i)) = support(ancestor(i)) + |tid(i)|;
8               tid(ancestor(i))= union (tid(ancestor(i)), tid(i));
9        end
10     end
11     Delete from D' infrequent ancestors;
12 end;
```

Figure (6) DESA for GAR

Table (6) Extended vertical Database

Itemset	Abbreviation	Tids	Suppor
{ Laptop }	A	2,6	2
{ Portable }	B	2,3,6	3
{ Computers }	C	1,2,3,6	4
{ Black/White	D	4,5	2
{ ColorPrinter	E	2,3	2
{ Printers }	F	2,3,4,5	4

5. OFGIM Algorithm

After explaining the theoretical aspects of OFGIM, figure (7) elucidates the steps of it. The input is the vertical extended database D'. The output is FGIDB, frequent generalized itemsets database. Step#3 stores frequent items in L1.

The loop presented in step #4 to step#10 gradually generates L2, L3, Ln by invoking Generalized_Itemset_Generator function and then it calls the optimization operation which illustrated in section 4.2 to optimize L_k. Step#16 selects two different last items itemsets X and Y. Step#18 finds the union of X and Y while step#19 calculates their intersection. Steps#20 computes the support of united itemsets without any scanning to the database, the support counting according to OFGIM is just the length of the tid list. Steps# 21 and 22 accept or reject the united itemset according to its support.

```
1  OFGIM (Input D', output FGIDB);
// D' Extended transaction DB
// FGIDB Frequent Generalized Itemsets DB
2  begin
3     L1 ={frequent 1-itemsets};
4     K=2;
5     While Lk-1# ∅ do
6      Begin
7          Lk = Generalized-Itemset-Gen(Lk-1);
8          Lk = optimize(Lk);
9           K=k+1;
10    End
11 End

12 Generalized -Itemset-Gen(Lk-1);
13 begin
14 Ck=∅
15 For all itemsets X∈Lk-1 and Y∈Lk-1 do
16    if X1=Y1∧…∧ Xk-2 = Yk-2∧ Xk-1< Yk-1 then
17     begin
18         C = union(X,Y);
19         CTID = intersect ( XTID, YTID);
20         Csupport = | CTID |;
21         If Csupport ≥ Dsup add C to Lk
22       Else ignore C
23    End;
24 End;
```

Figure (7) OFGIM algorithm

Recall table(6) which presents frequent items, L1, of figure (7) with minsup=2.

The followings shows k-itemsets presented by the algorithm:

L2={AB×26, AC×26, BC×236, BE×23, BF×23, CE×23, CF×23, DF×45}

L3={ABC×26, BCE×23, BCF×23, BEF×23, CEF×23}

L4={BCEF×23}

L5={ }.

6. Experimental Results

According to the previous sections, OFGIM consists of two phases; the first one is for extending the transactions of a database under mining according to the taxonomy database, and the second phase represents mining of generalized frequent itemsets. The second phase was tested and compared with apriori algorithm depending on many synthetic databases obtained by the program prepared by GavinShaw[11].These databases were composed of 10, 20, 50, 200, 500 and 5000 transactions and named as R1 to R6. Theirfeaturesare described in table(7).Apriori program is downloaded from www.**almaden**.**ibm**.com/st/info/studentopps/internopps/projects.

Table (8) and figure(8) present the execution times of OFGIM according to these databases and minsup=25%, 50%, 75%, and 100%.Figure (9) and Table (9) presents the execution time of IBM-Almaden implementation of the apriori algorithm.

Table(7) Features of randomly built databases

Datasets parameters	R1	R2	R3	R4	R5	R6
No. of transactions	10	20	50	200	500	5000
Average no. of items per transaction	5	7	7	7	20	100
No. of items on the top concept level	5	10	10	10	10	10
No. of levels in the hierarchal	3	4	4	4	4	4
Average no. of child items a given item has	3	4	4	4	4	4

Table (8) OFGIM execution time according to minsup=25,50, 75, and 100%

	R1	R2	R3	R4	R5	R6
25%	10	10	10	11	21	117
50%	5	7	7	8	21	94
75%	4	4	7	7	15	87
100%	0.4	1	4	5	9	53

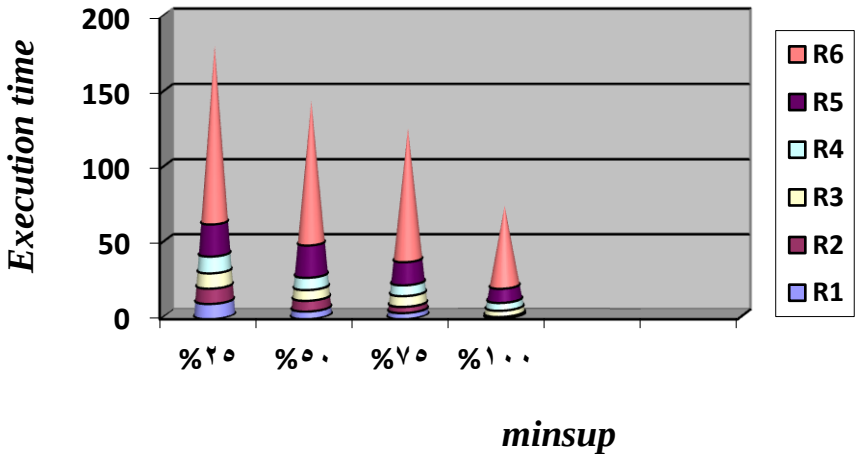


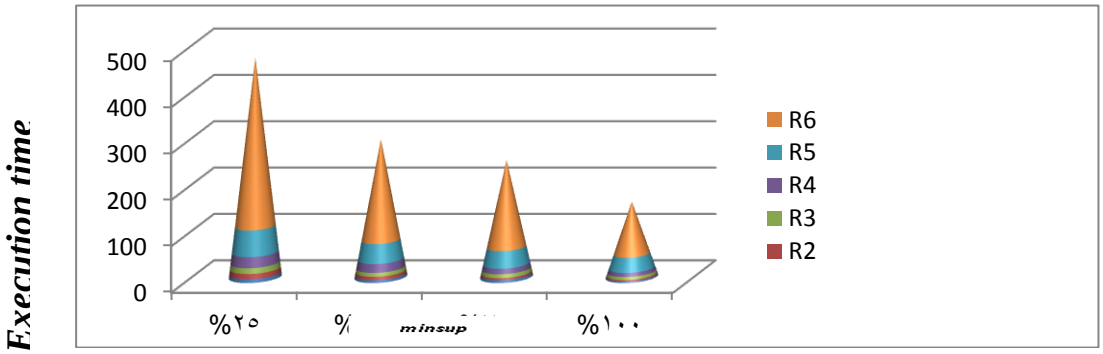
Figure (8) Execution time according to different minsup's values

The second phase of OFGIM was compared with apriori because they have same goal i.e., mining of frequent itemsets, but the first phase, extending phase, was lonely tested because it is related to GAR algorithms.Figure(10) presents the execution time of extending operation according to the number of transactions and 20 items as an average number of items in a transaction.

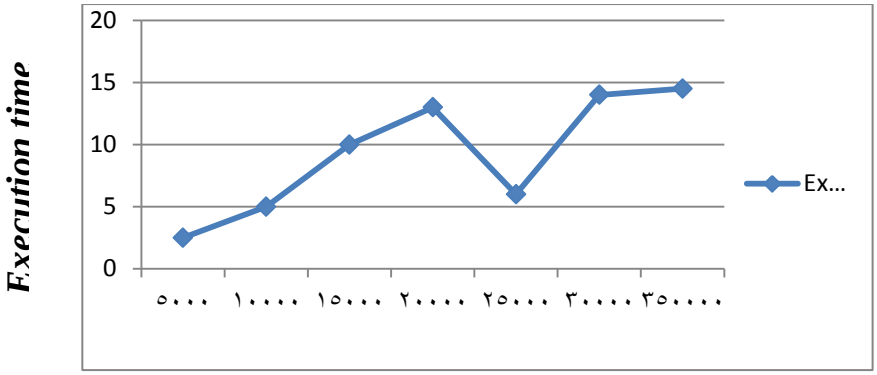
Table(9) Execution time according to minsup=25,50, 75, and 100% using

IBM/Almaden implementation of apriori algorithm

	R1	R2	R3	R4	R5	R6
25%	8	11	12	23	56	363
50%	6	7	8	18	41	217
75%	6	4	8	11	36	187
100%	3	3	6	8	30	112



Figure(9)) Execution time according to different minsup's values using IBM/Almaden implementation of apriori algorithm



Figure(10) Execution time according to the number of transactions and average number of items in a transaction.

7. Conclusions and suggestions for Future work

There are many conclusions can be derived from this work, and many recommendations to develop it in future.

7.1 Conclusions

Some of the conclusions are:

According to the previous sections, OFGIM consists of two processes; the first one is for extending the transactions of a database under mining according to the taxonomy database.The second process accomplishesthemining of generalized frequent itemsets, and produced, as an intentional by-product, a new algorithm to mine frequent crisp itemsets which depends on lattice theory. Indeed, it is compared with apriori algorithm and it overcomes it in all the aspects such as execution time, number of database scans, and excluding of pruning steps. Hence OFGIM can be used for crisp and generalized itemsets mining.

OFGIM overcomes apriori_based algorithms due to many reasons such as:

- It requires only one pruning step, i.e., it depends on the support counting only and does not need a pruning

step according to the apriori fact that said "All subsets of a frequent itemset are frequents too".

- No hash tree data structure is required.
- Support counting requires no database scans, it just counting of the tids continuing the itemset.

7.2 Suggestions for Future Work

1. We believe that OFGIM is very suitable for fuzzy itemsetsmining, an important future work is applying OFGIM to process fuzzy databases.
2. Developing OFGIM to mine sequential pattern and accomplish a comparison study with apriori based algorithms of this field.

References

[1] Farah Hanna AL-ZawaidahandYosefH. Jbara,"**An Improved Algorithm for Mining Association Rules in Large Databases**", World of Computer Science and Information Technology Journal (WCSIT),Vol. 1, No. 7, 311-316, 2011.

[2] KanhaiyaLal andN.C.Mahanti," **Mining Association Rules in Large Databaseby Implementing Pipelining Technique in Partition Algorithm",**International Journal of Computer Applications (0975 – 8887), June 2010.

[3] PratimaGautam and K. R. Pardasani, "**Efficient Method for Multiple-Level Association Rulesin Large Databases**", Journal of Emerging Trends inComputing and Information Sciences**,** December 2011.

[4] R. Agrawal T. Imielinski, and, A. Swami, **"MiningAssociation RulesBetween Sets of Item LargeDatabases"**, in Proc. Of the ACM SIGMODConference onmanagement of Data, Washington, 1993.

[5] R. Agrawal and R. Skrint, **"Fast Algorithms for Mining Association Rules"**, in proc. Of the VLDB conference,Santiago, Chile, 1994.

[6] M. Houtsma and A. Swami.**"Set-Oriented Mining ofAssociation Rules"**.InInt'l Conf. On Data Engineering, Taipei, Taiwan, March 1995.

[7] H. Mannila, H. Toivonen, and A. I. Verkamo. **"Efficient Algorithms for Discovering AssociationRules"**. In KDD-94: AAAI Workshop onKnowledgeDiscovery in Databases, pages 181-192, Seattle,Washington, July1994.

[8] J. S. Park, M.-S. Chen, and P. S. Yu. **"An Affective HashBased Algorithm for Mining Association Rules"** In

Proc. of the ACM-SIGMOD Conf. OnManagement of Data, San Jose, California, May 1995.

[9] Marcos Aur_elioDomingues and Solange Oliveira Rezende, **"UsingTaxonomies to Facilitate the Analysis of the Association Rules"**,arXiv:1112.1734v1 [cs.DB] 7 Dec 2011, Dec 2011.

[10] S.Prakash, M.Vijayakumar andR.M.S.Parvathi, "**A Novel Method of MiningAssociation Rule with Multilevel Concept Hierarchy"** Proceedingspublished by International Journal of Computer Applications® (IJCA),2011

[11] Gavin Shaw," **Discovery and Effective Use of Quality Association Rules in Multi-Level Datasets**", Ph.D Dissertation, Queensland university of Technology, 2010.

[12] Hussien K. Al-Khafajy, **"VKDP: Visual Knowledge Discovery Process a New Approach in Knowledge Discovery in Database"**, IJCSCE, 2011.

[13]B. A. Davey and H. A. Priestly, **"Introduction to Lattices and Order"**, 2nd ed,Cambridge University Press, 2002.

خوارزمية جديدة لتعدين مجاميع العناصر المعممة

أ.م.د. حسين كيطان الخفاجي
كلية الرافدين الجامعة
قسم هندسة اتصالات الحاسوب
Email: dr.hkm1811@yahoo.com

م. نورا احمد مولى الساعدي
كلية الرافدين الجامعة
قسم علوم الحاسوب
Email: noora_ahmed9@yahoo.com

المستخلص:

من المعتاد ان تكون قواعد البيانات التي تنقب لاستخراج قواعد الارتباط الثنائية هرمية او معممة. مع ذلك هنالك القليل من الخوارزميات التى تعنى بتعدين مجاميع العناصر الكبيرة المعممة للحصول منها على قواعد الارتباط المعممة والتي تقود الى معارف حقيقية ومحددة ذات النفع لمتخذي القرار.

هذا البحث يقدم خوارزمية جديدة لاكتشاف المجاميع المعممة، الخوارزمية تعتمد ببساطة على توسيع صفقات قاعدة البيانات باضافة اباء عناصرها اليها. بعد عملية التوسيع تاتي عملية التعدين والتي تعتمد على اتحاد مجاميع العناصر وتقاطع قوائم الصفقات. الخوارزمية تحتاج الى عمليتي مسح لقاعدة البيانات، الاولى للتوسيع والثانية للتعدين. ولا تتطلب هيكل بيانات محدد مثل اشجار الاختزال وانها تشذب عمليات التشذيب التي تحتاجها خوارزمية الابريوري او الخوارزميات المعتمدة عليها. اختبرت الخوارزمية بأستخدام ستة قواعد بيانات وقد تفوقت بجميع ألأختبارات بمعدل 1/4 من زمن التنفيذ ولكنها اظهرت ازدياداً بأستخدام الذاكرة الرئيسية.