***DWORM*: A Novel Algorithm To Maintain Large Itemsets in Deleted Items and/or Transactions Databases Without Re-Mining**

*Dr. Hussein K. AL-Khafaji

**Noora A. Al-Saidi

* AL-Rafidain University College.
** AL-Rafidain University College.

Abstract

Transactional databases can be updated by three cases; the addition of new transactions, the deletion of set of transactions, and\or increasing or decreasing of the support of the itemsets. The update process affects the previously mined itemsets, some of the large items will be small and vice versa. Therefore the updated database should be re-mined to discover the changes in the hidden itemsets. There are many algorithms to avoid the re-mining process in the case of updating a database by addition and there is one algorithm in case of changing the value of the support. But there is no actual algorithm to avoid the re-mining in the case of deletion. This research presents a novel algorithm to manipulate this case. The proposed algorithm manipulates the three possibilities of the deletion that are deletion of one or more items from a transaction or set of transactions, deletion of a set of transactions, and deletion of items from transactions and deletion of set of transactions at the same time. The experimental result shows that the updating algorithm outperforms the re-mining process in considerable amount of execution time.

1. Introduction

This section defines the problem of large itemsets mining and the problem of maintaining of these itemsets after updating the transactions database by increasing or decreasing the number of items, number of transactions, and the value of minimum support.

1.1 Large Itemset Description

Let I= {i1,i2, • • • ,i_m} be a set of literals, called items. Let DB be a database of transactions, where each transaction T is a set of items such that T⊆I. For a given itemset X ⊆ I and a given transaction T, we say that T contains X if and only if X ⊆T. The support count of an itemset X is defined to be X_{sup} = the number of transactions in DB that contain X. We say that an itemset X is large, with respect to a support threshold of s%, if $X_{sup} \geq |DB| \times s\%$ where |DB| is the number of transactions in the database DB. An association rule is an implication

of the form "X => Y", where X $\subseteq$I, Y $\subseteq$ I and X$\cap$Y=Ø. The association rule X =>Y is said to hold in the database DB with confidence c% if no less than c% of the transactions in DB that contain X also contain Y. The rule X => Y has support s% in DB if $X_{sup} \cup Y_{sup}$ = |DB| ×s%. For a given pair of confidence and support thresholds, the problem of mining association rules is to find out all the association rules that have confidence and support greater than the corresponding thresholds. This problem can be reduced to the problem of finding all large itemsets for the same support threshold. Thus, if s% is the given support threshold, the mining problem is reduced to the problem of finding the set L = $\{X \backslash X \subseteq I \wedge X_{sup} > |DB| \times s\%\}$ [1,2,3]. For the convenience of subsequent discussions, we call an itemset that contains exactly k items a k-itemset. We use the symbol L_k to denote the set of all k-itemsets in L.

1.2 Updating Database

Maintenance of large itemsets is a great challenge for designer of data mining algorithms. What is the situation of the large itemsets in the case of adding or deleting transactions from the database that is prepared to DM? What are the changes occurred on the large itemsets when the minimum support threshold is increased or decreased? Naturally, when new transactions are added to or deleted from database, some of the existing large patterns may disappear whereas new large patterns that did not exist before may also emerge. This say is the same for the second question. *The straightforward solution to discover the changes in the large itemsets is re-mining of the database depending on a data mining algorithm*. The re-mining process [3,4] will negligent all the previously discovered itemsets and all the mining process will be repeated. This means new efforts and execution time. To avoid the re-mining process, (the second solution), some algorithms for maintenance of frequent itemsets in updated databases was produced, like **F**ast **Up**date, **FUP2**[4], **U**pdate **W**ith **E**arly **P**runing, **UWEP**[5], and **A**lgorithm to **M**aintain **F**requent **I**temsets **W**ithout **R**e-mining **AMFIR** [6]. Actually, FUP2 and UWEP algorithms handled the case of adding new transactions in spite of the researcher's allegation that they are handle the case of deletion of transactions.

AMFIR algorithm is the unique algorithm which processes the case of increasing or decreasing the minimum support threshold. Also it manipulates the cases of adding new transaction(s), and new item(s) in a transaction or set of transactions. Furthermore it manipulates the two cases in the same time. In this research many unstudied cases are studied and manipulated such as *deletion of one or more items from a transaction or set of transactions, deletion of a set of transactions, and deletion of items from transactions and deletion of set of transactions at the same time*. The proposed algorithm excludes the need for applying the straightforward solution which is re-running the mining algorithm, i.e. re-mining the database. The following example depicts the inefficiency and complexity of re-mining process.

Example (1)

Table (1) shows a hypothesis transaction database, and one can find the support of each mined large k-itemset in table (2), depending on minsup=5.

Table (1) Hypothesis Database

TIDS	Items
1	A,C,D,E,F
2	B,D,F,G
3	A,D,E,G
4	A,B,D,E,F
5	A,B,C,E,F
6	B,F,G
7	A,D,E,F,K
8	A,B,D,F,K
9	A,D,F
10	A,F
11	B,C,F
12	A,C
13	B,F,K
14	A,B,C,K
15	A,C,D

Table (2) Large Itemsets With minsup=5

Support	Items	NO.
11	A,F	2
8	B,D	2
7	AD,AF,BF	3
6	C,DF	2
5	E,AC,AE,ADF	4

The mining of all the large itemsets together with the values of their supports is a major problem if the cardinality of the set of items is very high. Thus for the huge

database, the re-mining process makes multiple scans over the database[1,3]. This is the major drawback of the mining large itemsets algorithms in general. Figure (1) shows the generation of candidate and large itemsets where L^1, L^2,…, L^n and C^1, C^2, …, C^n represents large k-itemsets and candidate k-itemsets respectively. For more explanation to mining process, see figure(1)

C^1		L^1	C^2		L^2	C^3		L^3
Itemsets	support		Itemsets	support		Itemsets	support	
A	11	A	AB	4	AC	ACD	2	ADF
B	8	B	AC	5	AD	ACE	2	
C	6	C	AD	7	AE	ACF	2	
D	8	D	AE	5	AF	ADE	4	
E	5	E	AF	7	BF	ADF	5	
F	11	F	BC	3	DF	AEF	4	
G	3		BD	3				
K	4		BE	2				
			BF	7				
			CD	2				
			CE	2				
			CF	3				
			DE	4				
			DF	6				

Figure(1) Mining Process

Suppose that the database of table (1) is updated by deleting a number of items, A, B, C, D, F, and K; note that these items will represent the super set of some transactions. Table (3) represents affected transactions.

***Table (3) deleted transaction*s**

ITEM	TIDS
A	10,12,14,15
B	13,14
C	11,12,14,15
D	15
F	13
K	13,14

Table (4) depicts the new database after the deletion of these transactions. The mining algorithm will ignore the previous results that are presented in Table (2), and will return from initial point to discover the large itemsets in the same manner which is presented in Figure (1). The value of *minsup* will be 3. The next step is related to discovering all the large itemsets in the new database depending on

the new value of the *minsup*. Table (5) depicts the large itemsets in the updated database.

Table (4) Database

TIDS	Items
1	A,C,D,E,F
2	B,D,F,G
3	A,D,E,G
4	A,B,D,E,F
5	A,B,C,E,F
6	B,F,G
7	A,D,E,F,K
8	A,B,D,F,K
9	A,D,F
10	F
11	B,F

Table (5) Large itemsets With minsup=3

Support	Items	NO.
10	F	1
7	A,D	2
6	B,BF,AD,AF,DF	5
5	E,AE,ADF	3
4	DE,EF,ADE,AEF	4
3	G,AB,BD,ABF,BDF,DEF,ADEF	7

The mining algorithm makes multiple scans over the new database; Figure (2) represents solution steps to find new large itemsets after deleting the transactions from the old database.

L_1	C_2		L_2	C_3		L_3	C_4		L_4
	Itemsets	support		itemsets	support		itemsets	support	
A	AB	3	AB	ABD	2	ABF	ADEF	3	ADEF
B	AD	6	AD	ABE	2	ADE			
D	AE	5	AE	ABF	3	ADF			
E	AF	6	AF	ADE	4	AEF			
F	AG	1	BD	ADF	5	BDF			
G	BD	3	BF	AEF	4	DEF			
	BE	2	DE	BDF	3				
	BF	6	DF	DEF	3				
	BG	2	EF						
	DE	4							
	DF	6							
	DG	2							
	EF	4							
	EG	1							
	FG	2							

Figure(2) Re-mining process

It is obvious that there are *emerging* itemsets, i.e. some of the small itemsets in previous database become large after deletion of the transactions, and there are

withered itemsets that are large previously and become small after the deletion operation.

This process is not efficient because it ignores the discovered itemsets, and repeats all the work done previously. The re-mining process is costly particularly, when the size of the database is large. To avoid the re-mining process after each updating, this research presents an algorithm to maintain the discovered itemsets by utilizing the previous mining process.

2. Proposed Algorithm

This section focuses on a novel algorithm to maintain frequent itemsets in updated database. This algorithm concentrates on the deletion case. Really, there is no algorithm that deals with this problem including FUP2, UWEP, and AMFIR algorithms. This algorithm manipulates three cases two of these are not studied that are:

1. Deletion of one or more items from a transaction or set of transactions.
2. Deletion of a set of transactions.
3. Deletion of item(s) from transaction(s) and deletion of set of transactions at the same time.

These cases will be explained bellow. They are manipulated in the algorithms presented in figure (5) and figure (7). Table (6) elucidates the notation used in the proposed algorithm,

Table (6) Notation Used In The Proposed Algorithm

Notation	Description
DB	Original database before the updating
-Δ	The set of deleted K-itemsets and transactions from DB
-Δitems	The set of deleted K-itemsets selected from -Δ
DB-Δ	Updated DB by deletion the set of transaction of –Δ
minsup	User defined minimum support threshold
|A|	The number of transactions in the transaction database A
X	A set of items (i.e one itemset)
X_{sup}	Support of X in the set of transactions A
X_{TIDS}	Transaction list of X in the set of transactions A
C^k_A	Set of candidate k-itemsets in a set of transactions A
L^k_A	Set of large k-itemsets in a set of transactions A

In this section, we introduce the "*Delete WithOut Re-Mining*", ***DWORM***, algorithm step by step. Firstly, we'll focus on the special case of transactions and itemset deletion only. To discover the large itemsets in the updated database DB-Δ, DWORM is executed iteratively. In the k-th iteration, all the large K-itemsets in DB-Δ are found as follows.

Algorithm DWORM (input: DB, -Δ, L_{DB}, |DB|, |-Δ|, minsup);(output: $L_{DB-\Delta}$)

1. -Δitems = all 1-itemsets in -Δ .
2. check= all 1-itemsets in DB.
3. D=|DB|. (NOTE: D contains the number of transactions in old DB)
4. check=check- (-Δitems).
5. SI=Union of_Selected_itemsets(-Δitems) .
6. Candidates=Pruning_of_Selected_Itemsets(SI).
7. Updating_DB(Candidates, D,| DB- Δ |).
8. Updating_LDB(Candidates, -Δitems, |DB- Δ |).
9. if D≠| DB- Δ | then Checking_DB(check, | DB- Δ |).

Figure (3) DWORM algorithm

In the **first step,** DWORM selects all the 1-itemsets of -Δ which required to be deleted from the old database (DB). These 1-itemsets are stored in -Δitems. The **second step** puts all the 1-itemsets of old database in check list. In the **fourth step** the pruning of all 1-itemsets in the check is done by deleting all 1-itemsets which are common between check and -Δitems. The **fifth step** calls the procedure Union_of_Selected_Itemset, figure(4), to generate K-itemsets from the 1-itemsets in -Δitems and put the result in SI. It is self-documented procedure.

```
Procedure Union_of_Selected_itemsets(-Δitems)
5.1   k=1.
5.2   While  -Δitems^k ≠Ø  do
5.3     begin
5.4       for all itemsets  X and Y ∈ -Δitems^k  do begin
5.5         if X1 =Y1 Λ…..ΛXk-2 =Yk-2 Λ Xk-1 < Yk-1 then
5.6          -Δitems^k+1 = union(X,Y).
5.7        end. {for}
5.8      k=k+1
5.9     end.
```

Figure (4) Union_of_Selected_itemsets Algorithm

The **sixth step** calls the procedure Pruning_of _Selected_Itemset to prune the itemsets in SI. Figure (5) explains in details this procedure.

The steps(6.3-6.5) degenerate the generated (k+1)-itemset to its k-itemsets subsets. Step (6.6) intersects the tidlists of k-itemset X and Y to find the tidlist of the (k+1)-itemset(M), it finds the join of the tidlists.

```
Procedure Pruning of_Selected itemsets(SI)
6.1   k=2.
6.2   While SI^K≠Ø do begin
6.3     For all M ∈ SI^K  do
6.4       Begin
6.5           Subset M to X and Y .
6.6           M_TIDS= intersect(X_TIDS,Y_TIDS).
6.7           If M_TIDS=Ø then
6.8            Remove M and all supersets from SI
6.9           Else begin
6.10               M_sup=| M_TIDS|
6.11               Remove M_TIDS from X_TIDS and Y_TIDS.
6.12               X_sup =X_sup -M_sup.
6.13               Y_sup =Y_sup - M_sup.
6.14               If X_sup=0 then remove X from SI^K
6.15               If Y_sup=0 then remove Y from SI^K.
6.16          end; {else}
6.17       end;   {for}
6.18     k=k+1
6.19    end.
```

Figure (5) Pruning of selected itemsets algorithm

```
Procedure Updating_DB(input: Candidates,D),(output: | DB- Δ |)
7.1   k=1.
7.2   While Candidates^k ≠Ø  do  begin
7.3    for all X ∈ Candidates ^k do begin
7.4      Z=|X_TIDS|
7.5      for i=1 to Z do begin
7.6        Y=element(i) from  X_TIDS
7.7        remove X from transaction(Y) in DB.
7.8        if transaction(Y)= Ø then D=D-1     // transaction.no=  transaction.no-1//
7.9       end;   {for}
7.10     end;   {for}
7.11    k=k+1
7.12   end; {while}
7.13  | DB- Δ |=D
7.13    end.
```

Figure (6) Updating_DB Algorithm

The Steps(6.7-6.8) prune the SI from the (k+1)-itemset(M) and all its supersets if its tidlist is empty. If the tidlist of (k+1)-itemset(M) is not empty, the steps(6.10-6.16) are responsible for finding the support of (k+1)-itemset(M), removing the tidlists, which are common in k-itemsets X and Y, and updating the supports of X and Y and check it, if the supports equal zero remove X and Y from SI. These steps continue iteratively until SI^K is empty.

The **seventh step** invokes procedure Updating_DB to remove all the k-itemsets and transactions found in Candidates from old database. Figure (6) shows this procedure. The **eighth** step updates the k-itemsets in L_{DB} and removes all the k-itemsets which have supports smaller than new *minsup*. Figure (7) shows this procedure.

```
Procedure Updating_LDB(candidate, -Δitems, | DB- Δ |)
8.1   k=1.
8.2  C=-Δitems +candidate^(k+1)
8.3   While C^k ≠Ø  do  begin
8.4     for all X ∈ C^k do begin
8.5       if X large in DB then begin
8.6          X_sup(LDB) = X_sup(LDB)- X_sup(C)
8.7           if X_sup(LDB)<S% *| DB- Δ | then
8.8          remove X and all supersets from L_DB.
8.9       end;   {if}
8.10    end;{for}
8.11  end;  {while}
8.12  k=k+1
8.13  end.
```

Figure (7) Updating_LDB Algorithm

```
Procedure Checking_DB(check, | DB- Δ |)
9.1   k=1.
9.2   While check ^k ≠Ø  do  begin
9.3     for all X ∈ check^k do
9.4       if X large in DB then
9.5          add X to L_DB and P^k
9.6          k=k+1
9.7    end;     {while}
9.8    generate_candidates (P,L_DB, | DB- Δ |)
9.9     end.
```

Figure (8) Checking_DB Algorithm

The **ninth step** checks the number of transactions in new database. If the number of transactions is changed, then it calls the procedure Check DB to discover the supports of k-itemsets that was small before the updating and did not change after the updating and checks the supports of emerging k-itemsets which become large after the updating to add it to L_{DB}.

Procedure Generate -candidates (P,L_{DB}, | DB- Δ |)
1- k=1
2- while $P^k \neq \emptyset$ **do begin**
3- **for** all itemsets $X \in P^k$ and $Y \in L_{DB}{}^k$ **do**
4- **if** $X_1 = Y_1 \wedge \ldots \wedge X_{k-2} = Y_{k-2} \wedge X_{k-1} < Y_{k-1}$ **then begin**
5- $Z = X_1 X_2 \ldots X_{k-1} Y_{k-1}$
6- Z_{TIDS} = intersect (X,Y)
7- $Z_{sup} = |Z_{TIDS}|$
8- **if** Z_{sup} >= S%* | DB- Δ | **then**
9- add Z to L_{DB} and P^{k+1}
10- **end.**{if}
11- k=k+1
12- **end.**

Figure (9) Generate-Candidates Algorithm

EXAMPLE 2.

This example is presented to elucidate the behavior of the DWORM in all the previously mentioned cases of the updating by the deletion, which are not covered by example (1).

Table(7) A set of transactions DB

TIDS	Items
1	A,F
2	B,C,F
3	A,C
4	B,E,F
5	A,B,C
6	A,C,D

Table(8) A set of Large Itemsets in DB with the minsup=2

NO	Items	SUP
2	A,C	4
3	B,F,AC	3

Table(9) A set of transactions of - Δ

Items	TIDS	No. of Update
A	1,6	2
B	2	1
C	2,6	2
D	6	1

Table (7) shows a transactions database, DB, table(8) depicts the large itemsets of DB according to minsup=0.3, i.e |DB|*0.3= 1.8≈ 2, where |DB|, the number of transactions in DB, equals 6. Table(9) indicates the deleted items from the transactions of DB. For examples the first raw of table(9) means that item A will

be deleted from transaction#1 and transaction# 6. The database presented in table(9) is identified by (– Δ).

The first step of DWORM finds all the 1-itemsets of (–Δ), i.e., (-Δitems={A,B,C,D}), while step#2 fetches all 1-itemsets of the old DB from the previously mined frequent itemsets database, Hence check=(A, B, C, D, E, F). Step#4 removes the common items in (-Δitems) and check the lists and stores the uncommon items in check, therefore, check becomes equal to {E, F}.

Step#5 of the algorithm performs a union operation between the items of (-Δitems). Therefore, SI^i becomes:

SI^1={A,B,C,D}

SI^2={AB,AC,AD,BC,BD,CD}

SI^3={ABC,ABD,ACD,BCD}

SI^4={ABCD}

$SI= SI^1 + SI^2 + \ldots + SI^n$

In the next step, determining the zero_support itemsets is done by the algorithm shown in figure(5), by selecting one itemsets from SI and degenerates it to its (k-1)-itemsets. For example, say the selected itemset is AB, its subset is A and B. The intersection of these two elements are Ø which indicates that the tidlist AB, AB_{TIDS}= Ø . Therefore AB will be removed and all its supersets {ABC,ABD, ABCD} from SI .The content of SI after pruning is

SI^1={A,B,C,D}

SI^2={AC,AD,BC,CD}

SI^3={ACD}.

Another case is when the tidlist of the selected itemset is not equal to Ø. For example AC is the selected itemset and its subsets are A and C. The intersection of their tidlists is {6} which indicates that the AC_{SUP}=1. Therefore, the TID#6 will be removed from the tidlist of A and C. If the supports of the subsets become equal zero, then they will be removed from SI

SI^1={A}

SI^2={BC}

SI^3={ACD}.

Table (10) shows the result of these steps.

Table (10) The Candidate itemsets

Items	TIDS	SUP
A	1	1
BC	2	1
ACD	6	1

The next step selects k-itemset from Candidate like A, by using the A_{TIDS} which equal to 1; therefore A will be deleted from transaction#1. Then this transaction will be checked, if it become empty then update the number of transactions in DB. After that, check k-itemset(A), if it belongs to L_{DB} then update the support of A from L_{DB} and check the new support, if it becomes small then remove A from L_{DB}. Select another k-itemset (BC) and delete it from transaction(2). The last iteration selects the k-itemset ACD, deletes it from transaction(6) in DB, and updates the number of transactions in DB because the transaction(6) becomes empty.

After updating DB and L_{DB}, if the number of transactions in DB is changed; the supports of itemsets may be changed. For example, E item, the E_{sup} =1, the support of this item is stay small in (DB-Δ). Table(11) presents the (DB- Δ), i.e., the databases after deletion of some items from its transactions. Note, that transaction#6 is deleted completely after the elimination of the items A, C, and D. Table(12) shows $L_{DB-\Delta}$.

Table (11) A set of Transactions DB- Δ

TIDS	Items
1	F
2	F
3	A,C
4	B,E,F
5	A,B,C

Table(12) A set of Large Itemsets in DB- Δ with the minsup=2

NO	Items	SUP
1	F	3
4	A,B,C	2

3. Comparison with the Re-mining Algorithm

In this section a comparison between the complexity of the re-mining process and DWORM is illustrated. Recall table(1) and table(2) of example(1). The

database is updated according to table (3). The first row means delete the item A from transactions 10,12,14,15. It will not update the database transaction by transaction, but it finds the itemsets and therefore Tidlist as shown in figure(10) depending on the union of the items and intersection of the Tidlist. Table(13) shows the resultant itemsets. *The updater will access transactions 10, 11, 12, 13, 14, and 15 only once for each transaction instead of fourteen times.* The updated database is shown in table (14).

C^1		C^2		C^3		C^4	
Itemsets	Tids	Itemsets	Tids	Itemsets	support	Itemsets	support
A	10,12,14,15	AB	14	ABC	14	ABCK	14
B	13,14	AC	12,14,15	ABK	14		
C	11,12,14,15	AD	15	ACD	15		
D	15	AK	14	ACK	14		
F	13	BC	14	BCK	14		
K	13,14	BF	13	BFK	13		
		BK	13,14				
		CD	15				
		CK	14				
		FK	13				

Figure (10) The Candidates generation before pruning operation

Table (13) The Candidates after pruning operation

NO.	ITEMS	Cases of deletion
1	A	Delete one item from transaction 10
2	C	Delete one item from transaction 11
3	AC	Delete transaction 12
4	BFK	Delete transaction 13
5	ACD	Delete transaction 15
6	ABCK	Delete transaction 14

Table (14) A New Database

TIDS	Items
1	A,C,D,E,F
2	B,D,F,G
3	A,D,E,G
4	A,B,D,E,F
5	A,B,C,E,F
6	B,F,G
7	A,D,E,F,K
8	A,B,D,F,K
9	A,D,F
10	F
11	B,F

Table (15) Large itemsets With minsup=3

Support	Items	NO.
10	F	1
7	A,D	2
6	B,BF,AD,AF,DF	5
5	E,AE,ADF	3
4	DE,EF,ADE,AEF	4
3	G,AB,BD,ABF,BDF,DEF,ADEF	7

Table (16) shows the number of scanning over the DB by the re-mining algorithm and DWORM over the example database given in table (1). It is worth noting that the Apriori algorithm re-runs over whole set of transactions and therefore counting candidates over DB and $-\Delta$ is irrelevant. As table (16) shows, the proposed algorithm is a much smaller number of scanning operation over the DB than the re-mining algorithm.

Table(16)Number of scanning in the example(1) DB

Iteration No.		Re-mining	DWORM
Iteration1	NO. of scanning operation over the DB	6	2
Iteration2	NO. of scanning operation over the DB	10	1
Iteration3	NO. of scanning operation over the DB	8	2
Iteration4	NO. of scanning operation over the DB	1	1

4. Discussion and Conclusion

This paper presents a novel algorithm to manipulate many updating cases that frequently accomplished on transaction database, such as deleting a transaction or set of transactions form the database, increasing or decreasing the minsup, and deleting one item or more from a transaction or set of transactions, or all these operations at the same time. The examples elucidate that the comparison results show the efficiency of DWORM algorithm over the mining algorithms. It excludes

the generating and degenerating operations of the candidate itemsets. Furthermore, it excludes the needing for the hash tree, FP-tree, etc. [1,2,3], which most the mining algorithms are depending on them.

It is well-known that the mining process is started after many preparation steps which convert the database to static database or compel the miner to construct a static temporary copy of the database. DWORM is regarded as a miner for dynamic databases especially when it is accompanied with one of the algorithms FUP2, UWEP, and AMFIR to manipulate the cases of transactions updating and the case of increasing or decreasing the minimum support threshold.

References

[1] Ramez Elmasri, Shamkant B. Navathe, "*Fundamentals of **Database** Systems*", (5th Edition), Addison-Wesley, 2007.

[2] Usama M. Fayyad,"***Advances In Knowledge Discovery and Data Mining***", MIT Press, 2006.

[3] Jiawei Han, Micheline Kamber, "***Data Mining Concepts and Techniques***", Morgan Kaufmann, 3rd ed., 2006.

[4] David W. Cheung, S.D. Lee, Benjamin Kao, "***A General Incremental Technique for Maintaining Discovered Association Rules",*** Proc. Incremental conference on Database systems for advanced Application (DASF AA-97), Melbourne, Australia, April, 1997.

[5] Necip Ayan, "***Updating Large Itemsets With Early Pruning",*** M. Sc. thesis, Bilkent University, 2000.

[6] Hussein K. AL-Khafaji, Noora Ahmed, ***"A new Algorithm to Updating Large Itemsets in Dynamic Databases",*** Journal of AL-Rafidain University College, 2007**.**

<u>ادامة-بلا-تعدين</u>: خوارزمية مبتكرة لادامة المجاميع الكبيرة في قواعد البيانات المُحَدّثة بحذف الصفقات او العناصر او كلاهما معا

أ.م.د.حسين كيطان الخفاجي كلية الرافدين الجامعة
م.م.نورا احمد مولى الساعدي كلية الرافدين الجامعة

الخلاصة

تُحَدَّثُ قواعد البيانات الحركية بثلاثة حالات؛ اضافة صفقة او صفقات جديدة، حذف صفقة او صفقات، وتغيير مستوى قوة الدعم بالزيادة او النقصان. ان التحديث يُغَيّر حالة المجاميع الكبيرة فبعض الكبيرات منها تصبح صغيرة والعكس بالعكس ومنها ما تحافظ على حالاتها وان كانت تتغير قوة دعمها بكل الاحوال. ولاكتشاف هذا التغيير يجب اعادة عملية التعدين وبذلك نفقد ما استُكْشف سابقا فضلا عن هدر الوقت والجهد لتعقيد عملية التعدين. الباحثون قد اقترحوا العديد من الخوارزميات لتجنب عملية اعادة التعدين، فهنالك خوارزميات تعالج حالات التحديث عندما تضاف صفقة او صفقات، وهنالك خوارزمية وحيدة لمعالجة حالة التحديث عند تغيير مستوى قوة الدعم. ولا توجد خوارزمية لمعالجة حالة التحديث بحذف صفقة او صفقات من قاعدة البيانات. ان هذا البحث يقدم خوارزمية جديدة لمعالجة صيانة المجاميع الكبيرة بعد تحديث قاعدة البيانات بالحذف. كما ان البحث يعالج حالات لم يتم دراستها سابقا مثل حذف عنصر او عناصر من صفقة او صفقات، حذف صفقة او صفقات، وحذف عناصر من صفقات وحذف صفقات بوقت واحد. النتائج اوضحت تفوق الخوارزمية على عملية اعادة التعدين بشكل خطي ملحوظ.