An Efficient Method For Finding Emerging Large Itemsets

Susan P. Imberman
College of Staten Island
The Graduate Center
City University of New York
2800 Victory Blvd.
Staten Island, NY 10314
Imberman@postbox.csi.cuny.edu

Abdullah Uz Tansel
Baruch College
The Graduate Center
151 East 25th Street
New York, NY 10010
tansel@baruch.cuny.edu

Eric Pacuit
The Graduate Center
City University of New York
365 Fifth Avenue
New York, NY 10016
e_pacuit@hotmail.com

ABSTRACT
The incremental mining of association rules has been shown to be more efficient than rerunning standard association rule algorithms such as Apriori. As each increment is processed, we see the emergence of some itemsets. An itemset that has emerged is one that was small and is large in the current increment. An emergent large itemset is a small itemset that has the potential to become large, and will do so with high probability. In this paper we modify an existing incremental algorithm, UWEP, so that it can identify emergent large itemsets. We show that, on average, 65% of the emergent large itemsets identified by the algorithm actually do emerge.

General Terms
Algorithms, Experimentation

Keywords
Incremental Algorithms, Association Rules, Negative Border, Temporal Association Rules, Emergent Large Itemsets.

1. INTRODUCTION
The use of association rules to find the co-occurrence between variables in large datasets is a well researched technique for data mining. Association rules have been used in many domains, most notably market basket data. Here, mined associations between store products have been used for product shelf placement and customer marketing. In the banking industry association rules are used to find the characteristics of high risk credit customers.

The problem of finding association rules can be decomposed into two steps. The first step involves finding all large itemsets. This step is the most difficult. Algorithms that accomplish this, such as Apriori [1,2], are exponential in the worst case. Once the large itemsets are found, generating association rules is straightforward. and can be accomplished in linear time.

TDM '04, August 22, 2004, Seattle, Washington.

When a database is expanded with an incremental increase in the number of transactions, we can either rerun an algorithm such as Apriori, or use the knowledge obtained from the "original" database to reduce the processing time for finding small itemsets that have become large or any large itemsets that have become small. Several incremental algorithms have been proposed that can efficiently find large itemsets. [4,5,6,7,9,11]. UWEP (Update With Early Pruning) has been shown to be an efficient algorithm that addresses the incremental association rule problem. UWEP gains its efficiency by pruning from consideration, those large itemsets in *DB* (the original database), that are not present in *db* (the increment) and become small due to the change in the overall number of transactions in the combined "new" database, *DB+db*. UWEP scans DB at most once and db exactly once. It has been shown to generate and count the minimum number of candidates in order to determine the new set of association rules. [4,5]

The negative border has been defined as the set of minimal small itemsets, closed with respect to set inclusion, and whose subsets are all large. [12] The negative border is a subset closed collection of itemsets. Thomas, Bodagala, Alsabti, and Ranka [11] used the negative border set to see which itemsets have changed status from large to small and vice versa. Dong and Li [8] use interval closed borders to find emergent patterns. They define an emergent pattern as an itemset whose support increases significantly between databases.

If we look at emergent patterns with respect to the negative border we can see whether a small itemset is becoming large. In this paper we use this property of the negative border, in a modified version of UWEP called NUWEP (Negative border Update With Early Pruning), to find these emergent large itemsets. We view emergent large itemsets as a special case of the emergent patterns described by Dong and Li. An emergent large itemset (ELI) can be thought of as an itemset that is currently small in the combined database but has increased its support such that it will eventually become large. ELI are particularly interesting emergent patterns. For example, in the market basket domain, if we define an interval as the time between wholesale purchases, recognizing that a set of items will emerge or become large in the next time period, can allow the storekeeper to order these items much earlier than usual. Thus the storekeeper can avoid being short these items and losing the income their sales could have generated. Finding emergent large itemsets does not add a lot of increased overhead to the standard incremental algorithm. Our contribution in this paper is to show how an incremental approach allows us to find ELIs efficiently. We can identify ELIs that have the potential to

emerge with results that are better than random. Moreover, our incremental approach allows us to identify the time an itemset becomes large before waiting for the next batch cycle.

The organization of this paper is as follows: Section 2 gives the definition of the emergent large itemset problem. Section 3 outlines our approach for solving the emergent large itemset problem. In section 4 we show experimental results derived from synthetic data. Section 5 is our discussion and section 6 our conclusions.

2. STATEMENT OF PROBLEM

Define the support of an itemset *I,* as the number of transactions containing that itemset, divided by the number of transactions in the dataset. Itemsets with support above a user defined threshold called *minsup* are called large itemsets [1,2,3]. Given a set of items *I*, such as products in a store, an association rule is an implication $X \Rightarrow Y$ where X and Y are subsets of I and $X \cap Y = \phi$. Interesting association rules are defined by two metrics, support and confidence. Support is a measure of a rule's significance with respect to the database, and confidence is a measure or the rule's strength.

More formally, let N be the number of transactions in a database *DB*. Given an association rule $X \Rightarrow Y$, let S_1 be the number of transactions in *DB* that satisfy *X*. *X* satisfies a transaction *t* in *DB* if the items contained in *X* are also in *t*. Let S_2 be the number of observations in *T* that satisfy the vector $X \cup Y$. The support *s* of $X \Rightarrow Y$ is S_2 / N and the confidence *c* of $X \Rightarrow Y$ is S_2 / S_1 . An interesting association rule has support above or equal to a user defined parameter called *minsup*, and confidence equal to or above a user defined parameter called *minconf*. Large itemsets can generate interesting association rules. Finding large itemsets is difficult and costly, most times exponential. Finding rules from a large itemset is linear. Therefore most methods for finding association rules concentrate on efficiently finding large itemsets.

Many association rule algorithms have been formulated to address this problem. [1,2,3]. The problem increases in cost when one wishes to find large itemsets in a database that is increasing in size. Rerunning the association rule algorithm is costly, especially since much of the cost in processing large datasets lies in parsing the data. Incremental algorithms take advantage of the knowledge gained from the previous increment, to reduce iterations over the entire database. UWEP (Update With Early Pruning) is an efficient incremental algorithm, that counts the original database at most once, and the increment exactly once. In addition the number of candidates generated and counted is minimum. [4,5].

As time progresses, we see many interesting patterns with regard to the change in status of individual itemsets. An itemset that was small can become large, large itemsets can become small, or an itemsets may remain large or small. We define small itemsets that are moving toward large as emerging. Conversely, large itemsets moving toward small are submerging. A small (large) itemset that becomes large, i.e. support is above (below) *minsup*, is said to have emerged (submerged). The problem we address in this paper is, can we identify itemsets that are currently emerging (submerging). Next, which of these itemsets have the potential to emerge (submerge) within the next increment. Or, more generally, can this happen within *n* intervals. For this paper we talk about emergence, but the solution for the submergence problem is analogous.

For the remainder of this paper we use the following notation:

DB is the set of old transactions (where transactions denote the records in the original database)
db is the set of new coming transactions (the increment)
DB+db is the set of old and new coming transactions
$support_{DB}$ (X) is the support of itemset X in *DB*
$support_{db}$ (X) is the support of X in *db*,
$support_{DB+db}$ (X) is the support of X in *DB+ db*
tidlist *DB* (X) is the transaction list of X in *DB*
tidlist *db* (X) is the transaction list of X in *db*
tidlist *DB+db* (X) is the transaction list of X in *DB + db*
C_{db}^{k} is the set of candidate k-itemsets in *db,* where *k* is the number of items in that itemset.
L_{db}^{k} is the set of large k-itemsets in *db*
L_{DB}^{k} is the set of large k-itemsets in *Db*
L_{DB+db} is the set of large itemsets in *DB + db*.

2.1 Emergent Large Itemsets

In *Figure1* we have modified the formalizations given by Dong and Li in [8] for the special case of ELI. *Figure1* partitions the space of itemsets. It can also be visualized as all the possible transitions for an itemset X from *DB* to *DB + db*.

Definition 2.1.1: The support count (*SC*) of an itemset is the number of transactions that an itemset satisfies.

Assume that *db* is of constant size and is smaller than *DB*. Even though we place constraints on the problem, the following easily generalizes to the case where *db* varies. *Figure1* plots the support count in *DB* (denoted as SC_{DB}) against the support count in *db* (denoted as SC_{db}). Each point in the graph depicts an ordered pair (SC_{db}, SC_{DB}) where the sum of SC_{db} and SC_{DB} is an itemset's support count in *DB+db* at some increment interval.

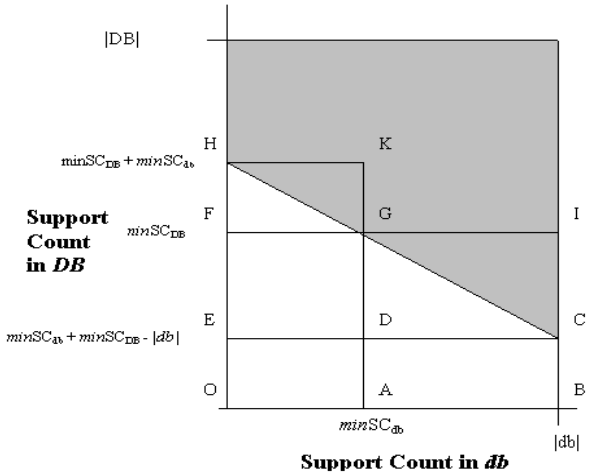

Figure 1 Emergent Itemsets

Definition 2.1.2 $minSC_{Db+db}$ is the minimum number of transactions needed to be large in *DB+db*.

Definition 2.1.3 $minSC_{DB}$ is the minimum number of transactions needed to have to have been large in *DB*.

Definition 2.1.4 $minSC_{db}$ is the minimum number of transactions needs to have been large in *db*.

If the increment adds no transactions to an itemset's support count, then in order to achieve $minSC_{Db+db}$ it's support count in *DB* has to be equal to $minSC_{DB} + minSC_{db}$. This is point H in *Figure1*. Alternately, if an itemset's *SC* is equal to $|db|$ in *db,* then to be large it's support in *DB* has to be some $SC = n$, where $n > 0$, and $n = minSC_{DB+db} - |db|$. This is point C in *Figure 1*.

Lemma 2.1.1: All points G = (SC_{db}, SC_{DB}), where $SC_{db} + SC_{DB} = minSC_{Db+db}$ lies on line HC.

Proof: : Proof directly follows from the definition of the line HGC. At point H, SC_{DB} of an itemset X has sufficient support at *DB* to become large at *DB+db* without any contribution from *db*, i.e., $SC_{DB} = minSC_{Db+db}$. From point H to point G, SC_{DB} declines, however the decline is compensated by SC_{db} and their sum ($SC_{DB} + SC_{db}$) equals to $minSC_{Db+db}$. At point G, an itemset X has the exact support count at both *DB* and *db* and hence G is on the line HC. For any point on GC, the increase in SC_{db} compensates the decrease in SC_{DB} and their sum still equals, $SC_{DB} + SC_{db} = minSC_{Db+db}$. □

Lemma 2.1.2 For all points G described in Lemma 2.1.1on the line HC, (SC_{db}, SC_{DB}) where $SC_{db} + SC_{DB} = minSC_{Db+db}$

Proof: Proof directly follows from the Lemma 2.1.1. (SC_{db}, SC_{DB}) = $minSC_{Db+db}$ is true by the definition of line HC and $minSC_{DB} + minSC_{db} = minSC_{Db+db}$ holds by the definition of minsup over *DB + db*. □

Line HC partitions the space of all itemsets in *DB+db* into large and small. The shaded area in *Figure1* represents all the large itemsets and it includes Line HC. In fact, using defined values for $minSC_{Db+db}$,, $minSC_{DB}$, ,and $minSC_{db}$ we can further partition the itemset space as shown in *Figure1*. Each partition exhibits some interesting properties with respect to itemsets in *DB*, *db,* and *DB+db*. Specific partitions under HC contain itemsets that are emerging in the current increment. For example, the area defined by ΔHFG represents those itemsets that were large itemsets in *DB*, small itemsets in *db*, and now are small in *DB+db.* These itemsets have therefore submerged. ΔGIC represents itemsets that were small in *DB* and large in *db*. These itemsets have emerged.

Lemma 2.1.3: An itemset that is small over *DB* with support *n,* and large in *db* with support >= *minsup,* has support in *DB+db* >*n*.

Proof: : Let n = SC_{DB} / |DB|. SC_{db} / $|db|$ >= *minsup* is given. Since the itemset is small in *DB*, $n < minsup$. We can break SC_{db} into j + k such that j / SC_{db} = n. It is shown that (SC_{DB} + j) / ($|DB|$ + $|db|$) = *n* [11]. Clearly adding a positive value to the numerator increases the value of the fraction, i.e., (SC_{DB} + j + k) / ($|DB|$ + $|db|$) > *n*. □

Lemma 2.1.4: An itemset that is small *DB* with support *n,* and small in *db* with support < *n* < *minsup,* has support in *DB+db* < *n*.

Proof: Proof is analogous to the proof of Lemma 2.1.3. □

Therefore, according to Lemma 2.1.3 and 2.1.4, all itemsets in area ABCG are emerging in the current interval and all itemsets in area OAGH are submerging. To find <u>all</u> ELI's in the current interval, we need to identify all itemsets in ABCG.

2.2 Interesting Emerging Itemsets

Finding interesting ELI's can be both a subjective or objective problem. For example, if we look at an itemset, that contains gardening tools such as hoe, rake, and spade, an ELI containing these items wouldn't be interesting if it were emerging during the spring/summer seasons. It would be interesting if it were emerging in the winter season. This would be a subjectively interesting ELI. Subjectively interesting ELI are domain and user dependent.

Definition 2.2.1 The growth rate of an ELI is the average gain in support over *n* increments.

By definition, ELI's that continue to remain ELI's will eventually become large. If the ELI is growing at a very small rate, then it will become large at some time far into the future. ELI's with faster growth rates will emerge sooner. Therefore, one type of objectively interesting ELI is one that will emerge within the next increment, or within the next *N* increments.

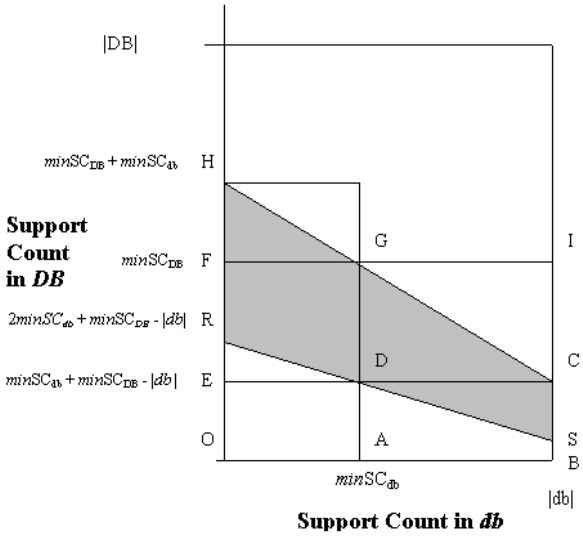

Figure 2 Potentially Emergent Large Itemsets

Maintaining our assumptions about a constant increment size and |db| < |DB|, we can establish a lower bound on the support count needed by an itemset in the current increment interval to achieve emergence in the next increment. The largest possible gain in support count in the next increment is equal to $|db|$. This occurs when an itemset is contained in every transaction in the increment. To have the *potential* to emerge in the next increment, the support count of the itemset in *DB+db* needs to be greater than or equal to $2minSC_{db} + minSC_{DB} - |db|$ in the current increment. All points with this value are represented by line RS in *Figure 2*. For example, if we have a $|DB| = 10{,}000$, $|db| = 1{,}000$ and *minsup = .20,* then the minimum support count for the current

increment is 2,200 (2,000 from *DB* + 200 from *db)*. If an itemset can add the maximum support incremental support count, a total of 1,000 from *db*, in the next increment, it would need a support count of at least 1,400 in the current increment to be able to attain the minimum support count of 2,400 needed to become large. The band of itemsets between line RS and line HC are all itemsets that have the potential to become large in the next increment, by this formula. If we use only information from the current increment to determine which ELI's have the potential to become large in the next increment, then all itemsets that are in this band and are emerging have the highest probability of emerging in the next increment. Therefore, itemsets in GDSC are most likely to emerge in the next increment.

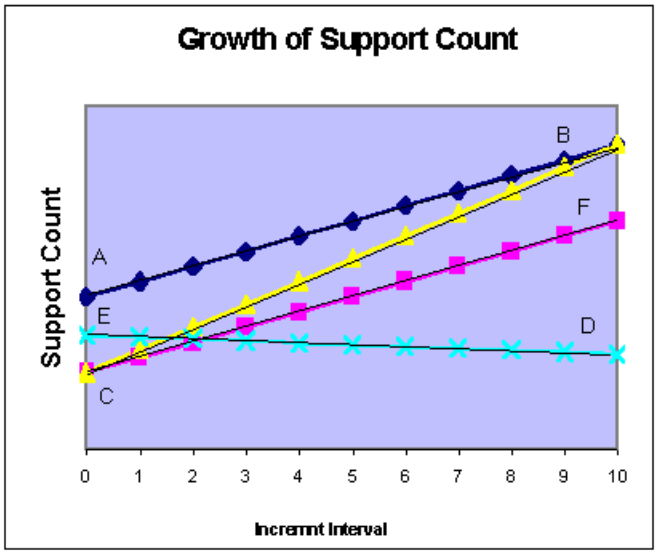

Figure 3 Growth in Support of Itemsets

In large datasets this can be a large group. We can bound this set even further by looking at an itemset's growth rate. If we were to plot the support count of an itemset over time, we see that in order to maintain minimum support, an itemset has to have a constant growth rate. Line AB in *Figure 3* represents the minimum support count, in the cumulative database (*DB*+*db*), in order for that itemset to remain at exactly *minsup*. If a small itemset's support count increases at the same rate as AB, i.e. its support in *db* is *minsup* for each interval, then that itemset will never become large. This is shown by line CF. Note the slope of CF in the graph is the same as AB. Those itemsets whose growth decreases with time, as shown by ED have slope less that of line AB. Emergent large itemsets, such as CB have slope greater than AB. Thus, even though an itemset is emerging, it may not emerge in the next increment or within the next *n* increments. Those itemsets whose growth rates allow them to become large in the next interval are potentially emergent large itemsets. Therefore these itemsets are those itemsets whose rate of growth is larger than that of an itemset maintaining *minsup*.

3. NUWEP, AN ALGORITHM FOR FINDING INTERESTING EMERGING LARGE ITEMSETS

Keeping track of all small itemsets' growth rates can be exponentially large in both time cost and memory cost. The negative border is a concise way to represent larger sets of small itemsets. The negative border is the set of small itemsets whose supersets are large [12]. Thomas, Bodagala, Alsabti, and Ranka [11] proved that an itemset *X* that was previously small in the original dataset and large in the *"new"* or original plus incremental dataset, has either moved from the negative border set to the set of large itemsets in the new dataset, or some subset of *X* has moved from the negative border set to the set of large itemsets in new dataset.

Lemma 3.1: An itemset in the negative border will emerge at the same time or before any of its supersets.

Proof: Let *X*, *Y* be itemsets, *X* is in the negative border and *Y* be an immediate superset of *X*. Lets assume that *Y* becomes large and *X* is not large. Apparently, $support_{DB}(X) < minsup$ and $support_{DB+db}(X)$ but $support_{DB}(Y) > minsup$. This is a contradiction since all the subsets (i.e., *X*) of a large itemset (*Y*) are also large [8]. □

Lemma 3.2 The growth rate of an itemset in the negative border will be equal to or greater than any of its supersets.

Proof: Let *X*, *Y* be itemsets, X is in the negative border and *Y* be an immediate super set of *X*. Lets assume that growth rate of *Y* is larger that the growth rate of *X*, i.e. $support_{DB+db}(Y) > support_{DB+db}(X)$. From the definition of support, this implies $SC_{Db+db}(Y) > SC_{Db+db}(X)$ and it is clearly a contradiction. □

If we keep track of the growth rate of an itemset in the negative border, and that itemset has the potential to emerge in the next increment, then we can easily find the supersets that will also emerge.

3.1 UWEP to NUWEP

UWEP is an incremental association rule algorithm that has been shown to efficiently solve the incremental association rule problem. It has been shown to generate and count the minimum number of candidates in order to determine the new set of association rules. [4,5]. To find ELI, we can modify UWEP so that in addition to counting and keeping track of large itemsets, it keeps track of the negative border.

Appendix 1 contains UWEP with the additional steps necessary to maintain the negative border. We call the modified algorithm, NUWEP (Negative border Update With Early Pruning).

Lemma 3.1.1**:** The number of candidates generated and counted by NUWEP algorithm is minimum.

Proof**:** Candidate generation is driven by the increment database, *db*. Therefore it would suffice to show that we generate and count minimum number of candidates in *db* at each level since NUWEP is a level wise algorithm. C^1_{db} contains only the itemsets whose support is greater than zero and it is a minimum bound. At level k only the itemsets that are large in db and DB + db are placed into L^k_{db}. Itemsets in L^k_{db} are used in generating C^{k+1}_{db}. Clearly this is the minimum possible number of candidate itemsets. The

candidates counted in *db* are minimum, since itemsets that are large in *DB* or *db* are only considered.

The number of itemsets counted over *DB* is also a minimum since only itemsets that are small in *DB* and large in *db* need to be counted over *DB*. If the itemset is in NB_{db}^{k} we do not need to count it. However, if NB_{DB}^{k} expands, itemsets need to be counted. There are two possibilities:

1) Any small itemset whose subsets are large over *DB+ db* is in NB_{DB+db}^{k+1}, and it needs to be counted if it is small in DB.
2) 2) Consider an itemset X that is large in *DB+db* and large in *db,* but small in *DB*. Consider an itemset Y that is small over *db* but large over *DB+db* and *DB*. An immediate superset of *X* made up of the items in *XY* is in the NB_{DB+db}^{k+1}. An algorithm that considers only counting these itemsets when generating the negative border is minimal. Since NUWEP generates the negative border and counts only itemsets that are of the two possibilities mentioned above, NUWEP is minimal. □

Of course maintaining the negative border adds cost with respect to time and memory. The memory cost of the negative border is much smaller than maintaining the same information on all small itemsets since the memory required to maintain both large and small itemsets is exponential with respect to the dimensionality of the dataset. This is not tractable for most large datasets.

As for time costs, we ran several experiments comparing execution time of NUWEP to UWEP with synthetic data generated as per [3]. We generated a 10,000 transaction dataset where the average transaction was 5 items, the average large itemset was 4 items, and the total number of items was 10 (T5.I4.D10). The results of these experiments on this dataset are in *Figure 4*. We see that that as we increase the size of *db* the difference between UWEP and NUWEP's execution time is constant. In fact for low levels of support the difference is negligible. In addition overall execution time increases at a logarithmic rate. As we increase the size of *DB*, holding *db* constant, we see that the difference between NUWEP and UWEP is also constant. Hence the cost in maintaining the negative border is related to increases in size of *db* rather than *DB*. We repeated these experiments with a dataset T5.I2.D10 with similar results. The largest negative border equals C(n, n/2), which is the combination of n items taken n/2 at a time, and n is equal to the total number of items. Therefore there is a bound on how large the constant difference between the two algorithms can be.

3.2 Finding potentially Emergent Large Itemsets.

Modifications were made to NUWEP that allow the calculation of growth rate for itemsets in the negative border. These modifications were minor. Itemsets were always stored in data structures that include their support count. Therefore, because of this, we know the support count of the negative border itemset in DB for the current interval. The algorithm update finds the updated value of the support count in *DB+db.* From these two values we calculate the growth rate, which is the slope of the line that connects these two points. Note that the additional memory required for this approach is a numeric variable to record the growth rate. Additional memory is needed only for those itemsets in the negative border.

More formally, given an itemset whose support count in *DB* is SC_{DB} and its support count in *DB+db* is SC_{DB+db}, then the growth rate of that itemset is $SC_{DB+db} - SC_{DB}$. The growth rate of an itemset that maintains minimal support is, $minSC_{Db+db} - minSC_{DB}$. An itemset where $\frac{SC_{DB+db} - SC_{DB}}{\min SC_{DB+db} - \min SC_{DB}} > 1$ is an emerging itemset. An itemset needs a support count of at least $minSC_{Db+2db}$ to emerge in the next increment. A potentially emerging large itemset is one that is emerging and $SC_{DB+db} + (SC_{DB+db} - SC_{DB}) > minSC_{Db+2db}$. Once an itemset in the negative border is identified as a potential ELI, then we can easily find all supersets that are also potentially emerging large itemsets.

4. EXPERIMENTAL RESULTS

We used the two synthetic market basket datasets that was created in for the UWEP/NUWEP comparison in section 3. NUWEP was run in turn on each of the datasets. In addition to the negative border, we kept track of the set of potential ELI, and those itemsets that actually emerged in the next increment. *Table 1* shows the results of these experiments.

Table 1 Potentially Emergent Large Itemsets

INCREMENT	#Potentially Emergent Large Itemsets	# That Emerged In Next Increment	Percent Of Actual Large Itemsets That Were Potentia
1	62	51	82
2	44	33	75
3	56	49	87
4	48	37	77
5	76	70	92
6	43	39	90
7	51	48	94
8	38	37	99

As can be seen from *Table 1,* for each increment, a large percentage of the itemsets that did emerge were correctly identified. The significance is that these itemsets were identified in the period prior to the one in which they actually did become large.

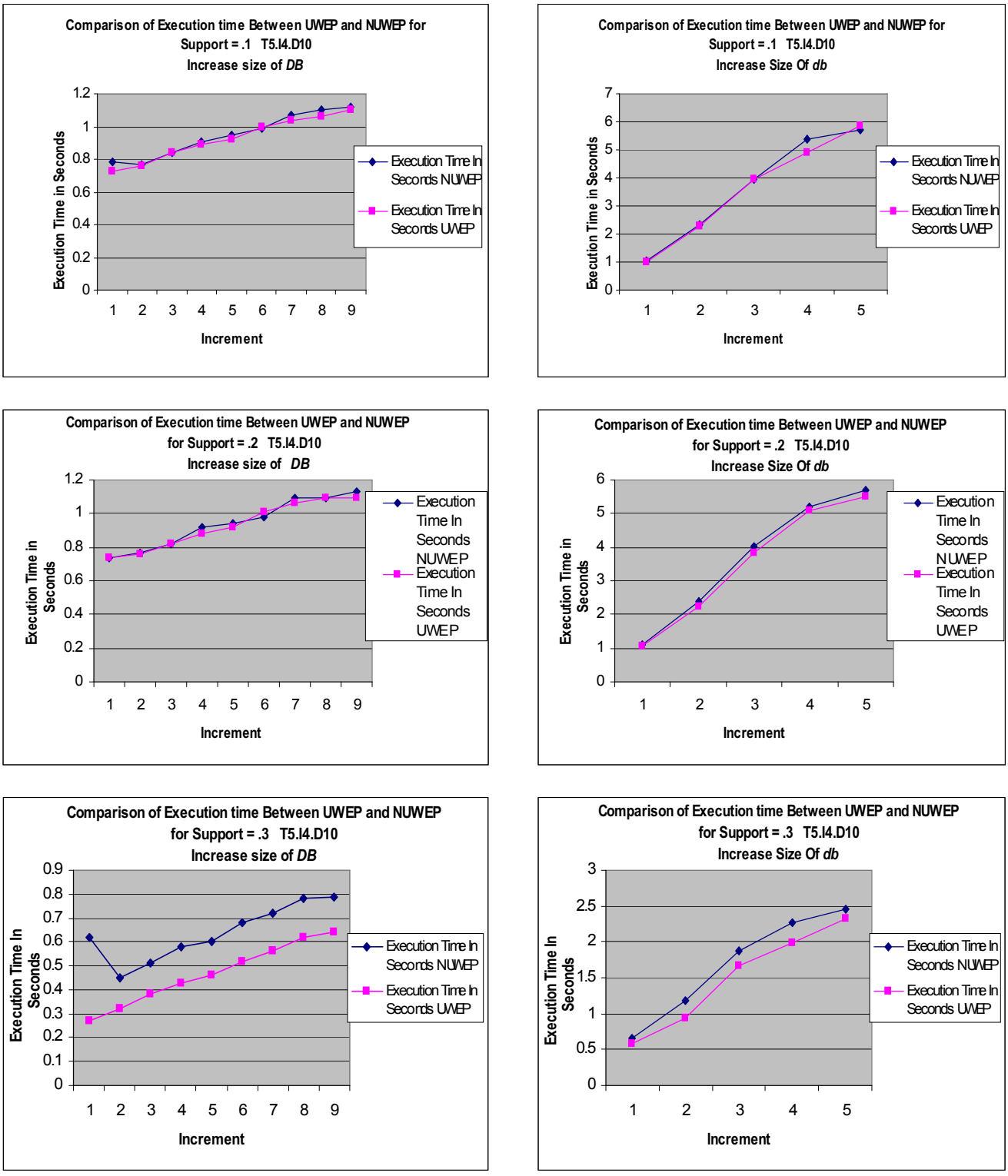

Figure 4 Comparison of UWEP to NUWEP

5. DISCUSSION

In the past, the incremental association rule algorithms were created to gain efficiency over standard association rule algorithms when finding large itemsets. Our approach shows that at the risk of losing some efficiency, we can gain more information about itemset behavior. By watching an itemset and its supersets move from the negative border to large, and possibly back indicate that there are patterns that can yield interesting information. For example, it is quite possible that a group of itemsets will oscillate between being large and ΔHFG in *Figure 2*. If this happens often, to a significant number of itemsets, one could conclude that support may be set either too high or too low. Since support is a user defined parameter, this oscillation behavior may give a heuristic for fine tuning this parameter.

Although we have shown how we can predict, with reasonable accuracy, which itemsets will emerge in the next increment, it is only natural to ask how one might predict if an itemset will emerge within the next n increments. If we use only the information available to us in the current increment, then this can be easily determined by adjusting the calculation of an emerging itemset. An itemset that will potentially emerge within n increments, is an itemset that is currently emerging and $SC_{DB+db} + n(SC_{DB+db} - SC_{DB}) > minSC_{Db+ndb}$. Of course, the larger n is, the less accurate our predictions.

Besides extending our view from one increment to n increments, we can also utilize the knowledge obtained about itemset behavior from past increments, including the current increment, to predict emergence. The problem here is that processing large databases requires substantial amounts of memory to store just the information on large itemsets and the small itemsets in the negative border. Maintaining too much information from past increments can be very memory intensive. If memory is not an issue, then we can remember the support counts of each itemset in the negative border, for each increment. These points may contain a rich set of interesting patterns. This allows us to apply standard statistical prediction techniques to these points to find the growth rate of an itemset. By doing this, we can avoid labeling itemsets that have spikes in support count as being potentially emergent. Thus, we may see an increase in the accuracy of prediction. Since memory is usually an issue, we can get a "dirtier" picture of an itemset's potential by time stamping the itemset with the increment number within which it enters the negative border, remembering the support count of the itemset when this happens, and use this information with the information in the current interval to determine the growth rate. The experimental results showed that we could predict, within the current period, a significant number of itemsets that would become large in the next period. The number of itemsets identified as being potential, and did emerge was less significant. We feel that if we were to use more past history, as outlined above, we could reduce the number of potentially large itemsets while still predicting well for the next increment.

6. CONCLUSIONS

In this paper we discussed a method that identifies small itemsets that have the potential to become large in the next increment. Our method makes use of an incremental approach, and by doing this decreases the time needed for processing if standard association rule algorithms did the same. Our results indicate that a good percentage of the itemsets that we predict to emerge actually do emerge. Our methodology easily extends to predicting emergence within a set number of increments. For the future, we would like to investigate using more history to identify potentially emergent large itemsets and compare them to the approach used in this paper.

7. REFERENCES

[1] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In Proceedings of 1993 ACM SIGMOD Intl. Conf. on Management of Data, pages 207--216, Washington, D. C., May 1993.

[2] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In Usama Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramaswamy Uthurusamy, editors, Advances in Knowledge Discovery and Data Mining, pages 307--328. AAAI/MIT Press, 1996.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Proceedings of 20 th Intl Conf. on Very Large Databases (VLDB'94), pages 487--499, Santiago, Chile, 1994.

[4] N.F. Ayn, A.U. Tansel, and E. Arun. An efficient algorithm to update large itemsets with early pruning. Technical Report BU-CEIS-9908 Dept of CEIS Bilkent Uniiversity , June 1999.

[5] N.F. Ayn, A.U. Tansel, and E. Arun. An efficient algorithm to update large itemsets with early pruning. Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, August 1999.

[6] David Wai-Lok Cheung, Jiawei Han, Vincent T. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental update technique. In Proceedings of Intl. Conf. on Data Engineering (ICDE'96), New Orleans, Louisiana, February 1996.

[7] David Wai-Lok Cheung, Sau Dan Lee, and Benjamin Kao. A general incremental technique for maintaining discovered association rules. In Proceedings of the 5 th Intl. Conf. on Database Systems for Advanced Applications (DASFAA'97), Melbourne, Australia, April 1997.

[8] Guzhu Dong, Jinyan Li. Efficient Mining of Emerging Patterns: Discovering Trends and Differences. Proceedings of the Fifth ACM SIGKDD International Conference on

Knowledge Discovery and Data Mining, San Diego, August 1999.

[9] N. L. Sarda and N. V. Srinivas. An adaptive algorithm for incremental mining of association rules. In Proceedings of DEXA Workshop'98, pages 240--245, 1998.

[10] A. Savasere, Edward Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In Proceedings of 21 st Intl. Conf. on Very Large Databases (VLDB'95), Zurich, Switzerland, September 1995.

[11] Shiby Thomas, Sreenath Bodagala, Khaled Alsabti, and Sanjay Ranka. An efficient algorithm for the incremental updation of association rules in large databases. In Proceedings of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining (KDD'97), New Port Beach, California, 1997.

[12] Hannu Toivonen. Sampling large databases for association rules. In Proceedings of 22nd Intl. Conf. on Very Large Databases (VLDB'96), Mumbay, India, September 1996.

Appendix I - NUWEP algorithm

DB is the set of old transactions (where transactions denote the records in the original database)
db is the set of new coming transactions (the increment)
DB+*db* is the set of old and new coming transactions
$support_{DB}$ (X) is the support of itemset X in *DB*
$support_{db}$ (X) is the support of X in *db*,
$support_{DB+db}$ (X) is the support of X in *DB*+ *db*
tidlist *DB* (X) is the transaction list of X in *DB*
tidlist *db* (X) is the transaction list of X in *db*
tidlist *DB*+*db* (X) is the transaction list of X in *DB* + *db*
C_{db}^{k} is the set of candidate k-itemsets in *db,* where *k* is the number of items in that itemset.
L_{db}^{k} is the set of large k-itemsets in *db*
L_{DB}^{k} is the set of large k-itemsets in *Db*
L_{DB+db} is the set of large itemsets in *DB* + *db*.
N_{DB}, is the negative border of *DB*
NB_{DB+db} is the negative border of *DB*+*db*
NC_{DB+db}^{k} are the candidate *k- itemsets* for the negative border

NUWEP(*DB*, *db, N_{DB}, L_{DB},* |*DB*|, |*db*|, *minsup*);
1 C_{db}^{1} = all 1-itemsets in *db* whose support is greater than 0
1a N_{DB+db} = Ø; *BorderSet* = Ø // Initialize N_{DB+db}
2 *PruneSet* = L_{DB}^{1} - C_{db}^{1}
3 **while** *PruneSet* ≠ Ø **do begin**
4 X = first element of *PruneSet*
5 **if** $support_{DB}$ (X) < *minsup* * | *DB* + *db* | **then begin**
6 remove X and all supersets from L_{DB} *and PruneSet*
6a remove all of X's supersets from N_{DB}
6b add X to N_{DB+db}
6c **end**
7 **else**
8 **begin**
9 add the supersets of X in L_{DB} to the *PruneSet*
10 add X to L_{DB+db}
11 remove X from L_{DB}
12 **end**
13 remove X from *PruneSet*
14 **end**
15 k = 1

16 **while** $C^k_{db} \neq \emptyset$ or $L^k_{DB} \neq \emptyset$ or *BorderSet* $\neq \emptyset$ **do begin**

17 *Unchecked* = L^k_{DB}

18 **for** all X $\in$ C^k_{db} **do**

19 **if** $support_{db}$ (X) < *minsup* * | db| **then** // X is small in *db*

20 **if** X $\in$ L^k_{DB} **then begin** // X is large in *DB*

21 remove X from *Unchecked*

22 **if** $support_{DB + db}$ (X) < *minsup* * |DB + db| **then begin**
// X is small in *DB* + *db*

23 remove all supersets of X from L_{DB}

23a remove all supersets of X from NB_{DB}

23b add X to N_{DB+db}

23c **end**

24 **else** // X is large in *DB* + *db*

25 add X to L_{DB+db}

26 **end**

26b **else** add X to NB_{DB+db}

27 **else** // X is large in *db*

28 **if** X $\in$ L^k_{DB} **then begin** // X is large in *DB*

29 remove X from *Unchecked*

30 add X to L_{DB+db}

31 add X to L^k_{db}

32 **end**

33 **else begin** // X is small in *DB*

34 find $support_{DB}$ (X) using *tidlists* if not in N_{DB}

35 // X is large in *DB* + *db*

36 if $support_{DB + db}$ (X) >= *minsup* * | DB + db| **then begin**

37 add X to L_{DB+db}

38a add X to *BorderSet* // X became large and the negative
// border expands

38c remove X from N_{DB}

38d **end**

38e **else** add X to NB_{DB+db}

40 **end**

41 **for** all X $\in$ *Unchecked* **do begin** // X is large in DB but not counted in *db*

42 find $support_{db}$ (X) using tidlists

43 if $support_{DB + db}$ (X) < *minsup* * |DB + db| **then begin** // X is small in *DB* + *db*

44 remove all supersets of X from L_{DB}

44a add X to N_{DB+db}

44b remove all ~~subsets~~ supersets of X from N_{DB}

44c **end**

45 **else** // X is large in *DB* + *db*

46 add X to L_{DB+db}

47 **end**

47a **for** all X $\in$ *BorderSet* **do begin** // X is large in db but not combined with small itemsets in
DB that are large in DB+db

47b NC^k_{DB+db} = Generate all k + 1 supersets of X //Candidate itemsets for Negative
//Border.

47b2 remove X from *BorderSet*

47b3 **end**

47c **for** all X ∈ NC^{k}_{DB+db}

47d **if** X has subsets in N_{DB+db} remove X from NC^{k}_{DB+db}

47e **else if** $support_{DB+db}$ (X) < *minsup* * |DB + db| do begin

47f add X to N_{DB+db}

47g remove X from NC^{k}_{DB+db}

47h **end**

47i **else do begin**

47j add X to L_{DB+db}

47k add X to *BorderSet*

47k2 remove X from NC^{k}_{DB+db}

47l **end**

47m **end**

48 k = k + 1

49 $C^{k}_{db} = generate\_candidate\ (L^{k-1}_{db})$ //generate candidate k - itemsets

50 **end**

50a **for** all X ∈ N_{DB} **do begin**

50b // X is small in DB and db since large itemsets of db are already considered

50c add X to N_{DB+db}

50d **end**