Knowledge and Information Systems (2001) 3: 338–355


Knowledge and Information Systems

# Efficient Graph-Based Algorithms for Discovering and Maintaining Association Rules in Large Databases

Guanling Lee, K. L. Lee and Arbee L. P. Chen

Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan

**Abstract.** In this paper, we study the issues of mining and maintaining association rules in a large database of customer transactions. The problem of mining association rules can be mapped into the problems of finding *large itemsets* which are sets of items brought together in a sufficient number of transactions. We revise a graph-based algorithm to further speed up the process of itemset generation. In addition, we extend our revised algorithm to maintain discovered association rules when incremental or decremental updates are made to the databases. Experimental results show the efficiency of our algorithms. The revised algorithm is a significant improvement over the original one on mining association rules. The algorithms for maintaining association rules are more efficient than re-running the mining algorithms for the whole updated database and outperform previously proposed algorithms that need multiple passes over the database.

**Keywords:** Association rule; Bit vector; Graph-based approach; Rules maintenance

## 1. Introduction

Data mining, or knowledge discovery in databases (KDD), has been considered as a promising new area in database research (Agrawal et al, 1993a; Srikant and Agrawal, 1996a). When the amount of data is large, it is quite important to extract potentially useful knowledge embedded in it. To extract such previously unknown knowledge from large databases is the task of data mining. Various types of knowledge can be mined from large databases, such as mining characteristics and classification rules (Agrawal et al, 1992; Anwar et al, 1992; Han et al, 1992; Han et al, 1993; Cheung et al, 1994; Ng and Han, 1994; Hwang and Fu, 1995; Yen and Chen, 1995, 1996a), association rules (Agrawal et al, 1993b; Agrawal and

*Received 4 August 1999*
*Revised 18 March 2000*
*Accepted 18 October 2000*

Srikant, 1994; Klemettinen et al, 1994; Mannila et al, 1994; Housma and Swami, 1995; Park et al, 1995; Savasere et al, 1995; Meo et al, 1996; Toivonen, 1996; Yen and Chen, 1996b), and sequential patterns (Agrawal and Srikant, 1995; Srikant and Agrawal, 1996b; Yen and Chen, 1996b).

Data mining has been widely applied in retail industry to improve market strategies. Each customer transaction stored in the database typically consists of customer identifier, transaction time, and the set of items bought in this transaction. It is important to analyze the customer transactions to discover customer purchasing behaviors.

The problem of mining association rules over customer transactions was introduced in Agrawal et al (1993b). An association rule describes the association among items like '80% of customers who purchase items X and Y also buy item Z in the same transaction', where X, Y, Z are initially unknown. There have been many works exploring this problem and its variations, such as mining quantitative association rules (Srikant and Agrawal, 1996a), generalized association rules (Srikant and Agrawal, 1995; Meo et al, 1996), and multi-level association rules (Han and Fu, 1995).

While new transactions are added to a database or old ones are removed, rules or patterns previously discovered should be updated. The problem of maintaining discovered association rules was first studied in Cheung et al (1996a), which proposed the FUP algorithm to discover new association rules when incremental updates are made to the database. Other algorithm proposed in Cheung et al, (1996b, 1997) improve FUP by generating and counting fewer candidates.

The graph-based algorithm DLG proposed in Yen and Chen (1996b) can efficiently solve the problems of mining association rules. In Yen and Chen (1996b), DLG is shown to outperform other algorithms which need to make multiple passes over the database. In this paper, we first propose the revised algorithms DLG$^*$ to achieve higher performance. Then we develop algorithm DUP (DLG$^*$ for database updates), which are based on the framework of DLG$^*$, to handle the problem of maintaining discovered association rules in the cases of insertion and deletion of transaction data.

The remaining of this paper is organized as follows. Section 2 gives detailed descriptions of the above two problems. The algorithms DLG$^*$ are described in Section 3. The algorithm DUP for maintaining association rules is described in Section 4. Experimental results are discussed in Section 5, and Section 6 concludes our study.

## 2. Problem Descriptions

### 2.1. Mining Association Rules

The following definitions refer to Agrawal and Srikant (1994). Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called *items*. A set of items is called an *itemset*. The number of items in an itemset is called the *length* of an itemset. Itemsets of length $k$ are referred to as *k-itemsets*. Let $D$ be a database of transactions, a transaction $T$ *contains* itemset $X$ if and only if $X \subseteq T$. An association rule is an implication of the form $X \Rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$, and $X \cap Y = \phi$. The *support count* of itemset $X$, $sup_X$, is the number of transactions in $D$ containing $X$. The association rule $X \Rightarrow Y$ has *support* $s\%$ if $s\%$ of transactions in $D$ contain $X \cup Y$, i.e. $sup_{X \cup Y}/|D| = s\%$. The association rule $X \Rightarrow Y$ has a confidence $c\%$ if $c\%$ of transactions in $D$ that contain $X$ also contain $Y$, i.e. $sup_{X \cup Y}/sup_X = c\%$.

The problem of mining association rules is to generate all rules that have support and confidence greater than the user specified thresholds, *minimum support* and *minimum confidence*. As mentioned before, the problem of mining association rules can be divided into the following steps:

1. Find out all itemsets that have supports above the user-specified minimum support. Each such itemset is referred to as a *large* itemset. The set of all large itemsets in $D$ is $L$, and $L_k$ is the set of large $k$-itemsets.
2. Generate the association rules from the large itemsets with respect to another threshold, minimum confidence.

The second step is relatively straightforward. However, the first step is not trivial if the total number of items $|I|$ and the maximum number of items in each transaction $|MT|$ are large. For example, if $|I| = 1000$, $|MT| = 10$, there are $2^{1000}$ possible itemsets. We need to identify large itemsets among $\sum_{i=1}^{10} C_i^{1000} \cong 2.66 \times 10^{23}$ potentially large itemsets. Therefore, finding all large itemsets satisfying a minimum support is the main problem of mining association rules.

## 2.2. Update of Association Rules

The following definitions refer to Cheung et al (1997). Let $L$ be the set of large itemsets in the original database $D$, and $s\%$ be the minimum support. Assume the support count of each large itemset $X, sup_X$, is available. Let $d^+(d^-)$ be the set of added (deleted) transactions, and the support count of itemset $X$ in $d^+(d^-)$ is denoted as $sup_X^+(sup_X^-)$. With respect to the same minimum support $s\%$, an itemset $X$ is large in the updated database $D'$ if and only if the support count of $X$ in $D'$, $sup'_X$, is no less than $|D'| \times s\%$, i.e. $(sup_X - sup_X^- + sup_X^+) \geqslant (|D| - |d^-| + |d^+|) \times s\%$.

Thus the problem of updating associations rules is to find the set of new large itemsets $L'$ in $D'$. Note that a large itemset in $L$ may not appear in $L'$. On the other hand, an itemset not in $L$ may become a large itemset in $L'$.

## 2.3. Previous Work

The Apriori (Agrawal and Srikant, 1994) and DHP (Park et al, 1995) algorithms of the generate-and-test type are quite successful for mining association rules. The Apriori algorithm generates the candidate $k$-itemsets $C_k$ in the $k$-th iteration by applying the **apriori-gen** function (Agrawal and Srikant, 1994) on the set $L_{k-1}$, which is the set of large $(k-1)$-itemsets found in the previous iteration. After finding candidates $C_k$, Apriori scans the database to count the support of each itemset in $C_k$, and then determines $L_k$ which can be used to generate $C_{k+1}$ in the next iteration. The algorithm DHP improves over Apriori by reducing the number of candidates and trimming the database progressively using hashing techniques.

Dynamic itemset counting (DIC) is introduced in Brin et al (1997). Unlike Apriori, where large itemsets and candidate itemsets are generated at the end of each scan, DIC takes these actions while scanning the database. The place where the action is taken is called 'stop' and each scan contains a number of stops. In Tang (1998), an incremental pruning technique (DICIP) is used to improve the DIC algorithm. At each stop, the candidate itemsets generated at the previous stop, which cannot be large, will be pruned. The DICIP algorithm improves the

performance of the DIC algorithm. Moreover, these two algorithms require fewer database scans and never generate more candidate sets than Apriori does.

In Yen and Chen (1996b), we propose the graph-based algorithms DLG to efficiently solve the problems of mining association rules. The algorithm DLG constructs an *association graph* to indicate the associations between large items, and then traverses the graph to generate large itemsets. The DLG algorithm is very efficient because it need not scan the database to count candidates, and needs to scan the database only once. However, it can be improved further by reducing the number of candidates. We revise the DLG algorithm in Section 3 and show its improvement in Section 5.2.

The algorithm FUP (Cheung et al, 1996a) was first designed for maintaining association rules when an incremental update is made to the database. FUP makes use of the stored support counts of the original large itemsets to generate a much smaller number of candidates to be checked in the updated database. Other algorithms FUP* (Cheung et al, 1996b) and $FUP_2$ (Cheung et al, 1997) are faster versions of FUP. The $FUP_2$ is able to maintain the discovered association rules in the cases including insertion, deletion, and modification of transactions. As with Apriori and DHP, all of the FUP families have to repeatedly scan the database to count candidates. In Tsai et al (1999), we propose another approach to deal with the rule maintenance problem. The information of the itemsets which are not large at one moment but may become large after updating the database is stored. This information is then used to reduce the number of database scans. In this paper, we extend our previous work (Lee et al, 1999) to efficiently update the rules by scanning the database only once. We compare with $FUP_2$ in the cases of insertion and deletion. The experimental results reveal that our approaches significantly outperform $FUP_2$.

## 3. The Revised Algorithm DLG*

In this section, we first briefly describe the algorithm DLG (Yen and Chen, 1996b) for efficient large itemset generation. DLG is a three-phase algorithm. The *large 1-itemset generation phase* finds large items and records related information. The *graph construction phase* constructs an *association graph* between large items, and at the same time generates large 2-itemsets. The *large itemset generation phase* generates large $k$-itemsets ($k > 2$) based on this association graph. The DLG* algorithm reduces the execution time during the *large itemset generation phase* by recording additional information in the *graph construction phase.*

### 3.1. Large 1-Itemset Generation Phase

The DLG algorithm scans the database to count the support and builds a bit vector for each item. The length of a bit vector is the number of transactions in the database. The bit vector associated with item $i$ is denoted as $BV_i$. The $j$-th bit of $BV_i$ is set to 1 if item $i$ appears in the $j$-th transaction. Otherwise, the $j$-th bit of $BV_i$ is set to 0. The number of 1's in $BV_i$ is equal to the support count of the item $i$.

For example, Table 1 records a database of transactions, where **TID** is the transaction identifier, and **itemset** records the items purchased in the transaction.

**Table 1.** A database of transactions.

| TID | Itemset |
|---|---|
| 100 | 2 3 4 5 6 |
| 200 | 1 3 7 |
| 300 | 3 6 |
| 400 | 1 2 3 4 |
| 500 | 1 4 |
| 600 | 1 2 3 |
| 700 | 2 4 5 |

**Table 2.** The bit vectors of large items in Table 1.

| Large item | Bit vector |
|---|---|
| 1 | 0101110 |
| 2 | 1001011 |
| 3 | 1111010 |
| 4 | 1001101 |
| 5 | 1000001 |
| 6 | 1010000 |

Assume the minimum support count is 2 transactions ($|D| \times$ minimum support). The large items and the associated bit vectors are shown in Table 2.

## 3.2. Graph Construction Phase

The support count for the itemset $\{i_1, i_2, \ldots, i_k\}$ is the number of 1's in $BV_{i1} \wedge BV_{i2} \wedge \ldots \wedge BV_{ik}$, where the notation '$\wedge$' is a logical AND operation. Hence, the support count of the itemset $\{i_1, i_2, \ldots, i_k\}$ can be found directly by applying logical AND operations on the bit vectors of the $k$ itemsets instead of scanning the database. If the number of 1's in $BV_i \wedge BV_j (i < j)$ is no less than the minimum support count, a directed edge from item $i$ to item $j$ is constructed in the association graph. Also, $\{i, j\}$ is a large 2-itemset. Take the database in Table 1 for example; the association graph is shown in Fig. 1, and $L_2 = \{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{2,5\},\{3,4\},\{3,6\},\{4,5\}\}$.

## 3.3. Large Itemset Generation Phase

For each large $k$-itemset $\{i_1, i_2, \ldots, i_k\}$ in $L_k (k > 1)$, the last item $i_k$ is used to extend the itemset into $(k+1)$-itemsets. If there is a directed edge from $i_k$ to item $j$, the itemset $\{i_1, i_2, \ldots, i_k, j\}$ is a candidate $(k+1)$-itemset. If the number of 1's in $BV_{i1} \wedge BV_{i2} \wedge \ldots \wedge BV_{ik} \wedge BV_j$ is no less than the minimum support count, $\{i_1, i_2, \ldots, i_k, j\}$ is a large $(k+1)$-itemset in $L_{k+1}$. If no large $k$-itemset is generated in the $k$-th iteration, the algorithm terminates.

Consider the above example; candidate large 3-itemsets can be generated based on $L_2$. The candidate 3-itemsets are $\{\{1,2,3\}, \{1,2,4\}, \{1,2,5\}, \{1,3,4\}, \{1,3,6\}, \{1,4,5\}, \{2,3,4\}, \{2,3,6\}, \{2,4,5\}, \{3,4,5\}\}$. After applying the logical AND operations on the bit vectors of the three items in each candidate, the large 3-itemset $L_3 = \{\{1,2,3\},\{2,3,4\},\{2,4,5\}\}$ is generated. The candidate 4-itemsets are $\{\{1,2,3,4\}, \{1,2,3,6\}, \{2,3,4,5\}\}$. After applying the logical AND operations on the bit vectors of the four items in each candidate, no large 4-itemset is generated. Thus, the algorithm terminates.

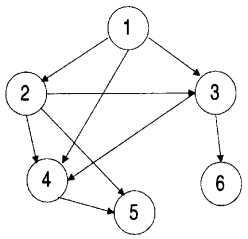



**Fig. 1.** The association graph for Table 1.

## 3.4. Improvements over DLG

In the $k$-th ($k > 2$) iteration, DLG generates candidate $k$-itemsets by extending each large $(k-1)$-itemset according to the association graph. Suppose, on the average, the out-degree of each node in the association graph is $q$. The number of candidate itemsets is $|L_{k-1}| \times q$, and DLG must perform $|L_{k-1}| \times q \times (k-1)$ logical AND operations on bit vectors to determine all large $k$-itemsets. The key issue of the DLG* algorithm is to reduce the number of candidate itemsets. The following properties are used by DLG* to reduce the number of candidates.

**Lemma 1.** If a $(k+1)$-itemset $X = \{i_1, \ldots, i_k, i_{k+1}\} \in L_{k+1} (k \geqslant 2)$, then $\{i_j, i_{k+1}\} \in L_2$ for $1 \leqslant j \leqslant k$, that is, item $i_{k+1}$ is contained in at least $k$ large 2-itemsets.

*Proof.* Any transaction that contains $X$ also contains the subsets of $X$. If the support of $X = i_1, \ldots, i_k, i_{k+1}\}$ meets the minimum support, all $k$ 2-itemsets $\{i_j, i_{k+1}\}$ $(1 \leqslant j \leqslant k)$ are also large. In addition, $i_{k+1}$ may be contained in other large 2-itemsets which are not subsets of $X$. Therefore, item $i_{k+1}$ is contained in at least $k$ large 2-itemsets. □

In the large itemset generation phase, DLG* extends each large $k$-itemset in $L_k (k \geqslant 2)$ into $(k+1)$-itemsets like the original DLG algorithm. Suppose $\{i_1, i_2, \ldots, i_k\}$ is a large $k$-itemset, and there is a directed edge from item $i_k$ to item $i$. From Lemma 1, if the $(k+1)$-itemset $\{i_1, i_2, \ldots, i_k, i\}$ is large, it must satisfy the following two conditions (otherwise, it cannot be large and is excluded from the set of candidate $(k+1)$-itemsets).

1. Any $\{i_j, i\}$ $(1 \leqslant j \leqslant k)$ must be large. In other words, the in-degree of the node associated with item $i$ must be at least $k$.
2. Moreover, a directed edge from $i_k$ to item $i$ means that $\{i_k, i\}$ is also a large 2-itemset. Therefore, we only need to check if all $\{i_j, i\}$ $(1 \leqslant j \leqslant k-1)$ are large.

These simple checks significantly reduce the number of candidate itemsets. In order to speed up these checks, we record some information during the graph construction phase. For the first condition, for each large item, we count the in-degrees of this item. For the second condition, a bitmap with $|L_1| \times |L_1|$ bits is built to record related information about the association graph. If there is a directed edge from item $i$ to item $j$, the bit associated with $\{i, j\}$ is set to 1. Otherwise, the bit is set to 0. DLG* requires extra memory space of size quadratic to $|I|$, but speeds up the performance significantly.

**Table 3.** The related information recorded by DLG* for Fig. 1.

| | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 |
| 2 | | 1 | 1 | 1 | 0 |
| 3 | | | 1 | 0 | 1 |
| 4 | | | | 1 | 0 |
| 5 | | | | | 0 |

| Item | Total |
|---|---|
| 1 | 0 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |
| 5 | 2 |
| 6 | 1 |

The DLG* algorithm is illustrated with the example in Table 1 in the following. The extra information recorded by DLG* is shown in Table 3. For large 2-itemset $\{1,2\}$, it can be extended into $\{1,2,3\}$, $\{1,2,4\}$, and $\{1,2,5\}$. Consider $\{1,2,3\}$: the in-degree of item 3 is 2 ($\geqslant 2$), and the bit associated with $\{1,3\}$ in the bitmap is 1. Therefore, $\{1,2,3\}$ is a candidate. Consider $\{1,2,5\}$: the in-degree of item 5 is 2 ($\geqslant 2$), but the bit associated with $\{1,5\}$ in the bitmap is 0. Therefore, $\{1,2,5\}$ is not a candidate. In the third iteration, DLG generates 10 candidates, but DLG* only generates 5 candidates ($\{1,2,3\}$, $\{1,2,4\}$, $\{1,3,4\}$, $\{2,3,4\}$, $\{2,4,5\}$). For large 3-itemset $\{1,2,3\}$, it can be extended into $\{1,2,3,4\}$, $\{1,2,3,6\}$. Consider $\{1,2,3,4\}$: the in-degree of item 4 is 3 ($\geqslant 3$), and the bits associated with $\{1,4\}$ and $\{2,4\}$ in the bitmap are both 1. Therefore, $\{1,2,3,4\}$ is a candidate. Consider $\{1,2,3,6\}$: the in-degree of item 6 is 1 ($< 3$). Therefore, $\{1,2,3,6\}$ is not a candidate. In the fourth iteration, DLG generates 3 candidates, but DLG* generates only 1 candidate $\{1,2,3,4\}$. In this example, DLG* has reduced the number of candidate $k$-itemsets ($k > 2$) by $((13-6)/13)*100\% \approx 54\%$. The DLG* algorithm, which is a revision of the DLG algorithm in Yen and Chen (1996b), is shown as follows.

```
/* Large 1-itemset generation phase */
forall itemset i do
  set all bits of BV_i to 0;
for (j = 1; j ⩽ N; j++) do begin /* N is the number of transactions */
  forall items i in the jth transaction do begin
    i.count++;
    set the jth bit of BV_i to 1;
  end
end
L_1 = φ;
forall items i in database D do
  if i.count ⩾ minsup then L_1 = L_1 ∪ {i};
  /* minsup is the minimum support count */

/* Graph construction phase */
if L_1 ≠ φ then begin
  initialize all |L_1| × |L_1| bits of Bitmap to 0;
  forall large 1-itemsets l ∈ L_1 do
    total[l] = 0; /* record the number of large 2-itemsets containing l */
  L_2 = φ;
  for every two large items i, j (i < j) do
    if (the number of 1's in BV_i ∧ BV_j) ⩾ minsup then begin
      CreateEdge (i, j); /* create a directed edge from i to j */
      L_2 = L_2 ∪ {{i, j}};
```

```
    Bitmap[i][j] = 1;
    total[i] + +;
    total[j] + +;
  end
  end

/* Large itemset generation phase */
k = 2;
while |L_k| ⩾ k + 1 do begin
  L_{k+1} = φ;
  forall itemsets {i_1, i_2, ..., i_k} ∈ L_k do begin
    if there is a directed edge from item i_k to item i then begin
      if total[i] ⩾ k then
        if (Bitmap[i_1][i] = Bitmap[i_2][i] = ··· = Bitmap[i_{k-1}][i]) = 1) then
          if (the number of 1's in BV_{i1} ∧ BV_{i2} ∧ ... ∧ BV_{ik} ⩾ BV_i) ⩾ minsup then
            L_{k+1} = L_{k+1} ∪ {{i_1, i_2, ..., i_k, i}};
    end
  end
  k = k + 1;
end
```

## 4. Efficient Update Algorithms based on DLG*

In this section, we introduce the update algorithm for transaction insertion and deletion. The algorithm DUP is based on the framework of DLG*, which can be split into three phases. As in Cheung et al (1996a), we assume the support counts of all large itemsets found in the previous mining operations are available. If a candidate itemset $X$ is large in the original database, we can directly get the support count $sup_x$ in the original database $D$. Otherwise, we must apply logical AND operations on the bit vectors associated with $D$ to find the support count $sup_x$. However, we can use the following properties to reduce the cost of performing logical AND operations. As defined in Section 2, $sup_x^+$ is the support count of itemset $x$ in the set of inserted transactions $d^+$ and $sup_x^-$ is the support count in the set of deleted transactions $d^-$. The following lemma is similar to Lemma 4 in Cheung et al (1997).

**Lemma 2.** If an itemset $X$ is not large in the original database, then $X$ is large in the updated database only if $sup_x^+ - sup_x^- > (|d^+| - |d^-|) \times s\%$.

*Proof.* Since $X$ is not large in the original database $D$, $sup_x < |D| \times s\%$. When $sup_x^+ - sup_x^- \leqslant (|d^+| - |d^-|) \times s\%$, $sup'_x = sup_x + sup_x^+ - sup_x^- < (|D| + |d^+| - |d^-|) \times s\%$, $X$ can not be large in the updated database. □

For each $k$-itemset $X$ not in $L_k$, we first apply logical AND operations on the bit vectors associated with the changed part of the database ($d^+$ and $d^-$). For the itemsets $X$ satisfying $sup_x^+ - sup_x^- \leqslant (|d^+| - d^-|) \times s\%$, we can determine they will not be large in $L'$ without applying logical AND operations on the bit vectors associated with the unchanged part of the database ($D - d^-$). In the following, we describe the DUP algorithm in detail.

**Table 4.** An example.

| TID | Itemset |
|---|---|
| 100 | 1 2 3 4 5 6 |
| 200 | 1 3 7 |
| 300 | 3 6 |
| 400 | 1 2 3 |
| 500 | 1 4 |
| 600 | 1 2 3 |
| 700 | 2 |
| 800 | 2 4 5 |
| 900 | 2 4 5 |

**Table 5.** The bit vectors and associated bits of large items in Table 4.

| Large item | $BV^-$ | $BV^{(D-d^-)}$ | $BV^+$ | $\delta^-$ $\Delta$ $\delta^+$ |
|---|---|---|---|---|
| 1 | 11 | 01110 | 00 | 1 1 0 |
| 2 | 10 | 01011 | 11 | 1 1 1 |
| 3 | 11 | 11010 | 00 | 1 1 0 |
| 4 | 10 | 00100 | 11 | 1 1 1 |
| 5 | 10 | 00000 | 11 | 1 0 1 |

## 4.1. Large 1-Itemset Generation Phase

The DUP algorithm scans the set of inserted and deleted database $d^+$ and $d^-$. For these two databases, the bit vector $BV_i^+$ and $BV_i^-$ are build for each item $i$. In order to determine which item is large in the updated database $D'$ more efficiently, DUP requires the support count of each item in the original database $D$ be stored in the previous mining operation. Hence, we can directly calculate the new support count $sup'_{\{i\}} = sup_{\{i\}} + sup^+_{[i]} - sup^-_{[i]}$. If an item $i$ is large in $D'$, DUP scans the remaining database (original database $-$ deleted database) and builds a bit vector $BV_i^{(D-d^-)}$. After completing the phase, the bit vectors $BV_i^{(D-d^-)}$, $BV_i^+$ and $BV_i^-$ for each large item $i$ are available. This requires extra storage space of size linear to $|I|$ to store $sup_{\{i\}}$ for each item $i$, but reduces the cost of building bit vectors and counting supports for those items not large in the updated database.

For each large item $i$, we allocate three bits $i.\Delta, i.\delta^+$ and $i.\delta^-$. We set the bit $i.\Delta$ $(i.\delta^+, i.\delta^-)$ to 1 if item $i$ is large in $D$ $(d^+, d^-)$. Otherwise, the bit is set to 0. The usage of these three bits is explained in the following two phases.

For example, consider the original, inserted and deleted databases in Table 4: TID 100 and 200 are deleted from the original database, and TID 800 and 900 are inserted into the original database. Assume the minimum support is 25%, that is, an itemset is in $L'$ if the support count is no less than $7 \times 0.25 = 1.75$. The bit vectors and three associated bits ($\Delta, \delta^+$ and $\delta^-$) for each large item is shown in Table 5. The large items in the updated database are items 1, 2, 3, 4, and 5. Since items 1, 2, 3 and 4 are large in $L_1$, their associated $\Delta$ bits are all set to 1. Item 5 is not large in the original database, and its associated $\Delta$ bit is set to 0. For the inserted (deleted) database, if the support count of any item is no less than $|d^+| \times 0.25 = 0.5$ ($|d^-| \times 0.25 = 0.5$), this item is large in $d^+(d^-)$. Since $sup^+_{\{2\}}, sup^+_{\{4\}}, sup^+_{\{5\}}$ are all greater than 0.5, $sup^+_{\{1\}} = sup^+_{\{3\}} = 0$, we set $2.\delta^+ = 4.\delta^+ = 5.\delta^+ = 1$, $1.\delta^+ = 3.\delta^+ = 0$. ($sup^-_{\{1\}}, sup^-_{\{2\}}, sup^-_{\{3\}}, sup^-_{\{4\}}, sup^-_{\{5\}}$ are all greater than 0.5, and their associated $\delta^-$ bits are all set to 1.)

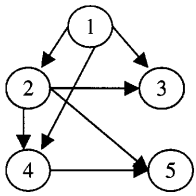



**Fig. 2.** The association graph constructed by DUP.

## 4.2. Graph Construction Phase

Each 2-itemset $X = \{i, j\}$ $(i < j)$, where $i, j$ are large items in $D'$, is a candidate 2-itemset. The $sup^+_{\{i,j\}}$ can be found by counting the number of 1's in $BV_i^+ \wedge BV_j^+$. Similarity, $sup^-_{\{i,j\}}$ can be found by counting the number of 1's in $BV_i^- \wedge BV_j^-$. For $X \in L_2$, $sup_x$ is available from the previous mining result, and we can calculate $sup'_x = sup_x + sup_x^+ - sup_x^-$. If $sup'_x$ is no less than $|D'| \times s\%$, then $X$ is added into $L'_2$. For $X \notin L_2$, according to Lemma 2, $X \in L'_2$ only if $sup_x^+ - sup_x^- > (|d^+| - |d^-|) \times s\%$. If $sup_x^+ - sup_x^- > (|d^+| - |d^-|) \times s\%$, we perform $BV_I^{(D-d^-)} \wedge BV_j^{(D-d^-)}$ and count the number of 1's to find $sup_x$, then add this count to $sup_x^+$ to get $sup'_x$. If $sup'_x$ is no less than $|D'| \times s\%$, $X$ is added into $L'_2$.

The three bits $i.\Delta$, $i.\delta^+$ and $i.\delta^-$ of each large item $i$ can be used to further improve the performance. Lemma 3 is used to illustrate the function of the bits.

**Lemma 3.** If an itemset $X$ is not large in the original database, then $X$ can not be large in the updated database if $sup_x^+ < |d^+| \times s\%$ and $sup_x^- > |d^-| \times s\%$.

*Proof.* Since $X$ is not large in the original database $D$, $sup_x < |D| \times s\%$. The support of itemset $X$ ($sup'_x$) after updating the database is equal to $sup_x + sup_x^+ - sup_x^-$. $sup'_x < |D| \times s\% + |d^+| \times s\% - |d^-| \times s\% = (|D| + |d^+| - |d^-|) \times s\%$. Therefore, $X$ can not be large in the updated database. □

There is also a property that should be mentioned here. Refer to Lemma 1: if any subset of itemset $X$ is not large, $X$ can not be a large itemset. However, if all subsets of itemset $X$ is large, $X$ may or may not be a large itemset, further checking is needed to make sure whether $X$ is a large itemset. According to these properties, before we check whether $\{i, j\} \in L_2$, we can check if both $i.\Delta$ and $j.\Delta$ are equal to 1. If either $i.\Delta$ or $j.\Delta$ is 0, $X$ can not be large in $L_2$. Thus, we save the cost of the membership check, which is costly when $|L_2|$ is large. For $X \notin L_2$, if either $i.\delta^+$ or $j.\delta^+$ is 0, we can make sure that $sup_x^+$ can not be greater than $|d^+| \times s\%$ without performing $BV_i^+ \wedge BV_j^+$, which is costly when the number of transactions in $d^+$ is large. For $X \notin L_2$ and $sup_x^+ < |d^+| \times s\%$, if either $i.\delta^-$ or $j.\delta^-$ is 1, $sup_x^-$ may be greater than $|d^-| \times s\%$. Therefore, $BV_i^- \wedge BV_j^-$ is performed to check whether $X$ is large in $d^-$. If $X$ is large in $d^-$, we know $X \notin L'_2$. For example, we know 2-itemset $\{3, 5\}$ can not be a large 2-itemset, because $3.\Delta \wedge 5.\Delta = 0$, $3.\delta^+ \wedge 5.\delta^+ = 0$, $3.\delta^- \wedge 5.\delta^- = 1$ and itemset $\{3, 5\}$ is really large in $d^-$.

For each large 2-itemset $\{i, j\}$, a directed edge from item $i$ to item $j$ is constructed in the association graph. We also set three bits $X.\Delta$, $X.\delta^+$ and $X.\delta^-$ for each large 2-itemset $X$.

Continuing the above example, there are 8 candidate 2-itemsets. These candidates are all in $L'_2$, except $\{3, 4\}$. $L'_2 = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{2, 5\},$

**Table 6.** The associated bits of the large 2-itemsets.

| $L'2$ | $\Delta$ | $\delta^+$ | $\delta^-$ |
|---|---|---|---|
| $\{1,2\}$ | 1 | 0 | 1 |
| $\{1,3\}$ | 1 | 0 | 1 |
| $\{1,4\}$ | 1 | 0 | 1 |
| $\{2,3\}$ | 1 | 0 | 1 |
| $\{2,4\}$ | 0 | 1 | 1 |
| $\{2,5\}$ | 0 | 1 | 1 |
| $\{4,5\}$ | 0 | 1 | 1 |

**Table 7.** Number of candidate itemsets.

| Iteration | DLG* | DUP | | |
|---|---|---|---|---|
| | $(D-d^-)\cup d^+$ | $D-d^-$ | $d^+$ | $d^-$ |
| 2 | 10 | 0 | 8 | 5 |
| 3 | 3 | 1 | 2 | 1 |

$\{4,5\}\}$. The association graph is shown in Fig. 2, and Table 6 shows the associated bits of the large 2-itemsets.

## 4.3. Large Itemset Generation Phase

Candidate itemset $C_k$ is extended from $L'_{k-1}$ by the method used in DLG*. Suppose $X=\{i_1,i_2,\ldots,i_k\}$ is a large $k$-itemset, and a candidate $Y=\{i_1,i_2,\ldots,i_k,i_{k+1}\}$ can be generated successfully based on $X$. Similar to the above phase, we get $sup_Y^+$ by performing $BV_1^+\wedge BV_2^+\wedge\ldots\wedge BV_{k+1}^+$, and $sup_Y^-$ by performing $BV_1^-\wedge BV_2^-\wedge\ldots\wedge BV_{k+1}^-$, and then checking whether $Y\in L_{k+1}$. The bits $X.\Delta$ and $i_{k+1}.\Delta$ are used to save the cost of the membership check. For $Y\notin L_{k+1}$, if either $X.\delta^+$ or $i_{k+1}.\delta^+$ is 0, and Y is large in $d^-$, we know $Y\notin L'_{k+1}$ without performing $BV_1^{(D-d^-)}\wedge BV_2^{(D-d^-)}\wedge\ldots\wedge BV_{k+1}^{(D-d^-)}$ and $BV_1^+\wedge BV_2^+\wedge\ldots\wedge BV_{k+1}^+$. Three bits $X.\Delta, X.\delta^+$ and $X.\delta^-$ are set for each large itemset $X$, which can be used in the next iteration.

Continue the above example. There are two candidate 3-itemsets $\{1,2,3\}$ and $\{2,4,5\}$, and their supports are all greater than 1.75. Therefore, $\{1,2,3\}$ and $\{2,4,5\}\in L'_3$. Table 7 compares the number of candidates of DLG* with that of DUP. DLG* performs logical AND operations on bit vectors associated with the whole updated database $((D-d^-)\cup d^+)$, with a total of 13 candidates. DUP performs logical AND operations on bit vectors associated with $D-d^-$, $d^+$ and $d^-$, with a total of 10 candidates.

In the $k$-th iteration, $N\times(k-1)$ logical AND operations are performed to determine all large $k$-itemsets, where $N$ is the candidate number. While DLG* performs $10\times1+3\times2=16$ operations on the bit vectors of length 7, DUP performs $(8+5)\times1+(2+1)\times2=19$ operations on the bit vectors of length 2 and $1\times2=2$ operations on the bit vectors of length 5. Therefore, DUP reduces the cost of performing logical AND operations by $1-(19*2+2*5)/(16*7)\approx57\%$.

The DUP algorithm also can deal with the case when there is no inserted database or deleted database. For the case of $|d^+|=0$, we only need to set $\delta_i^+=0$

**Table 8.** The parameters.

| | |
|---|---|
| $\|D\|$ | Number of transactions |
| $\|d\|$ | Number of inserted/deleted transactions |
| $\|I\|$ | Average size of the potentially large itemsets |
| $\|MI\|$ | Maximum size of potentially large itemsets |
| $\|L\|$ | Number of the potentially large itemsets |
| $\|T\|$ | Average size of the transactions |
| $\|MT\|$ | Maximum size of the transactions |
| $N$ | Number of items |

and $sup_i^+ = 0$. For the case of $|d^-| = 0$, we only need to set $\delta_i^- = 1$ and $sup_i^- =$ minsup.

## 5. Experimental Results

To assess the performance of our graph-based algorithms for discovering and maintaining large itemsets, the algorithms DLG (Yen and Chen, 1996b), DLG*, $FUP_2$ (Cheung et al, 1997), and DUP are implemented on a Sun SPARC/20 workstation. We first show the improvement of DLG* over DLG, and then demonstrate the performance of DUP by comparing with DLG* and $FUP_2$, which is the most efficient update algorithm recently.

### 5.1. Synthetic Data Generation

In each experiment, we use synthetic data as the input dataset to evaluate the performance of the algorithms. The method to generate synthetic transactions is the same as the one used in Yen and Chen (1996b). The parameters used are similar to those in Yen and Chen (1996b) except for the size of the changed part of the database ($d^+$ or $d^-$). The parameters used to generate our synthetic database are listed in Table 8. The reader can refer to Yen and Chen (1996b) for a detailed explanation of these parameters.

We generate datasets by setting $N = 1000$ and $|L| = 1000$. We choose two values for $|T| = 10$ and 20, and the corresponding $|MT| = 20$ and 40, respectively. We choose two values for $|I| = 3$ and 5, and the corresponding $|MI| = 5$ and 10, respectively. We use T$x$.I$y$.D$m$.d$n$, adopted from Cheung et al (1996a), to mean that $|T| = x$, $|I| = y$, $|D| = m$ thousands, and $|d| = n$ thousands. Notice that a positive value $n$ means the size of inserted transactions $|d^+|$, and a negative one means the size of deleted transactions $|d^-|$.

The way we create the updated database is straightforward. To generate the original database $D$ and the inserted transactions $d^+$, a database of size $|D| + |d^+|$ is first generated and then the first $|D|$ transactions are stored in $D$, and the remaining $|d^+|$ transactions are stored in $d^+$. Similarly, to generate the original database $D$ and the deleted transactions $d^-$, a database of size $|D|$ is first generated, and the first $|d^-|$ transactions are considered as deleted ones.

### 5.2. Comparison of DLG* with DLG

We perform an experiment on the dataset T10.I5.D100 with minimum support 0.75%. The experiment results are shown in Table 9. Let us examine the number

**Table 9.** Number of candidate itemsets.

| Iteration | DLG | DLG* | DLG*/DLG |
|---|---|---|---|
| 3 | 3449 | 1358 | 39% |
| 4 | 2283 | 873 | 38% |
| 5 | 1124 | 428 | 38% |
| 6 | 345 | 125 | 36% |
| 7 | 70 | 16 | 23% |
| 8 | 10 | 1 | 10% |
| 9 | 1 | 0 | 0% |
| Total | 7282 | 3801 | 38% |

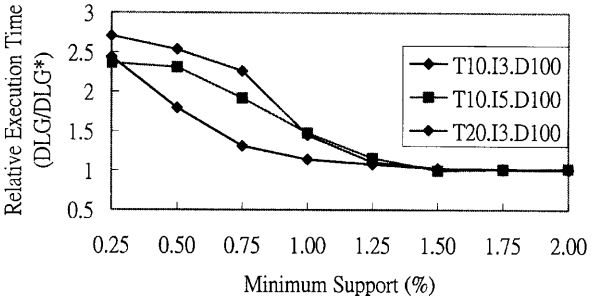



**Fig. 3.** Comparison of DLG* with DLG.

of candidate itemsets generated by each algorithm. DLG* reduces the number of candidate $k$-itemsets of DLG ($k > 2$) by $1 - 38\% = 62\%$.

Figure 3 shows the relative execution times for DLG* and DLG, using three synthetic datasets. The minimum support is varied between 0.25% and 2%. The result shows that DLG* can perform 2.7 times faster than DLG with a moderate size database of 100,000 transactions. The performance gain is achieved by the significant reduction in the number of candidates. As shown, the performance gap increases as the minimum support decreases because the increase of the large itemset number results in more redundant candidates generated by DLG. Moreover, for a larger $|I|$ or $|T|$, DLG* can perform much better than DLG. This is because DLG* prunes more candidates and the computation cost for candidates' support counting is increased as $|I|$ or $|T|$ becomes large.

## 5.3. Effects of the Minimum Support on the Update Algorithm

The next two experiments show the effects of the minimum support. The value of minimum support is varied between 0.25% and 2%. We set T10.I5.D100.d+1.d−1 to demonstrate the effect. Moreover, for a fair comparison, the dataset of $FUP_2$ is also put in main memory to ignore the time needed for scanning the disk.

As shown in Fig. 4, DUP is 1.9 to 3.6 times faster than DLG*, and 1.9 to 2.8 times faster than $FUP_2$. The result shows that DUP always has a better performance than re-running DLG* and $FUP_2$. The speed-up ratio over DLG* decreases as the minimum support increases, because the number of large itemsets becomes smaller and re-running DLG* is less costly. In general, the smaller the minimum support is, the larger the speed-up ratio over $FUP_2$; this is because

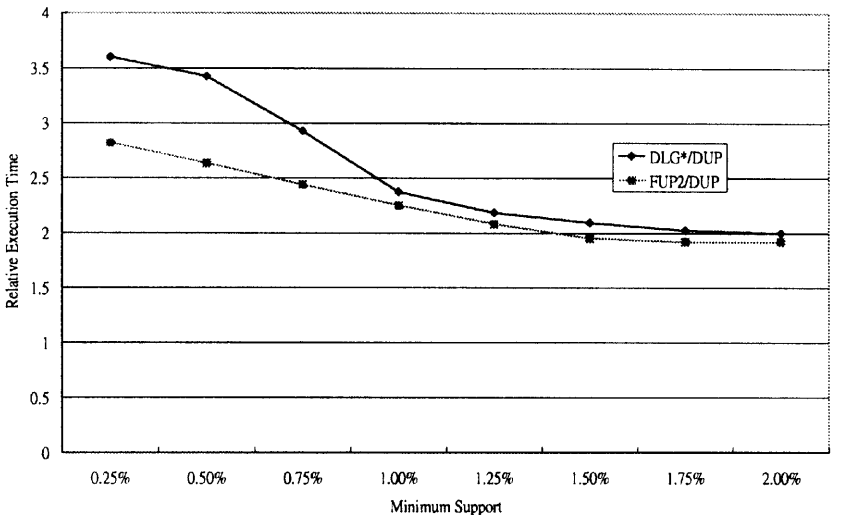



**Fig. 4.** Effects of minimum support.

a small minimum support will induce a large number of large itemsets and the computation cost of candidates' support counting is more expensive in $FUP_2$.

## 5.4. Effects of the Update Size on the Update Algorithms

Next, we examine how the size of the changed part of the database affects the performance of the algorithms. Compare DUP to re-running DLG*: when the amount of changes becomes large, the performance of update algorithms degrades. There are two major reasons for this. First, the previous mining results become less useful when the updated database is very different from the original one. Second, the number of transactions which need to be handled increases. Compare DUP to $FUP_2$: when the number of changes becomes large, the speed-up ratio increases. The reason is that the computation process of candidate support counting is more efficient in DUP, and when the inserted database increases $FUP_2$ spends more time computing the candidates' support. Two series of experiments are conducted to support the analysis. We set T10.I5.D100.d + x.d − 1 for the insertion case, and T10.I5.D100.d + 1.d − x for the deletion case. The minimum support is set to 1%.

In the insertion case, we increase the number of inserted transactions from 20 k to 120 k to evaluate the performance ratio. Figure 5 shows that DUP is 2.9 to 3.3 times faster than $FUP_2$, and 1.4 to 1.9 times faster than DLG*. When comparing DUP to $FUP_2$, the speed-up ratio increases as the inserted database increases. When comparing DUP to DLG*, although the execution time DLG* also increases as $|d^+|$, the speed-up ratio decreases. However, DUP still has a better performance even when $|d^+| = 120$ k, which is larger than the size of the original database.

In the deletion case shown in Fig. 6, DUP is 2.6 to 3.0 times faster than $FUP_2$, and 1.0 to 1.6 times faster than DLG* for $10 \leqslant |d^-| \leqslant 30$. The execution time of DLG* decreases as $|d^-|$ increases. DUP outperforms DLG* for $|d^-| \leqslant 30$, but

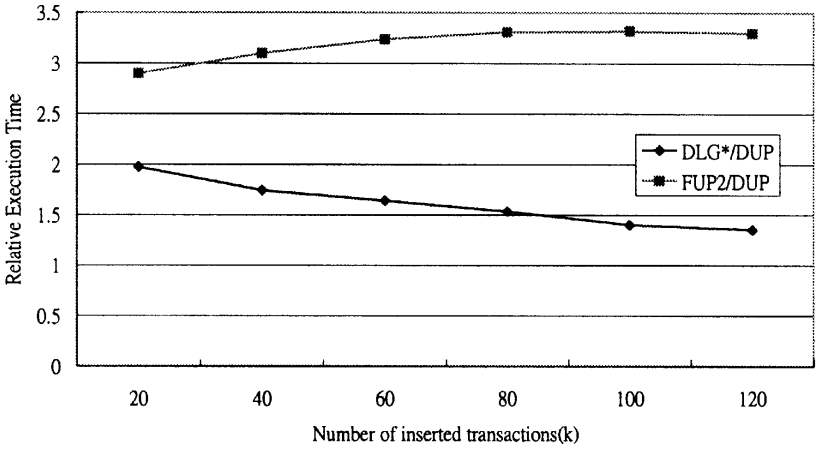



**Fig. 5.** Effects of the number of inserted transactions.

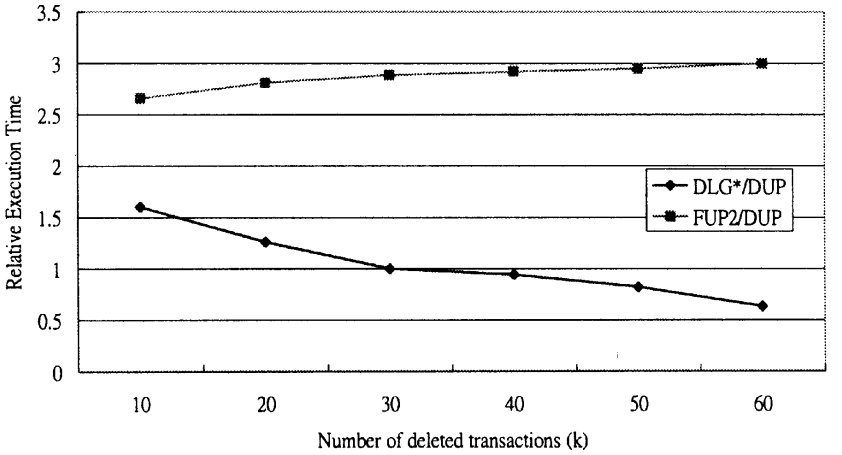



**Fig. 6.** Effects of the number of deleted transactions.

eventually drops down to DLG* for $|d^-| \geqslant 40$. However, DUP always performs significantly faster than $FUP_2$.

In Fig. 7, we examine how the size of the deleted transactions affects the performance of DUP, using three synthetic datasets. Compared with re-running DLG* in the remaining database $(D - d^-)$, DUP maintains a better performance when $|d^-|$ is limited to 30% of $|D|$.

## 6. Conclusion and Future Work

We study efficient graph-based algorithms for discovering and maintaining knowledge in the database. The revised algorithm DLG* is developed to efficiently solve the problem of mining association rules. DLG* improves DLG (Yen and Chen, 1996b) by reducing the number of candidates and the cost of performing

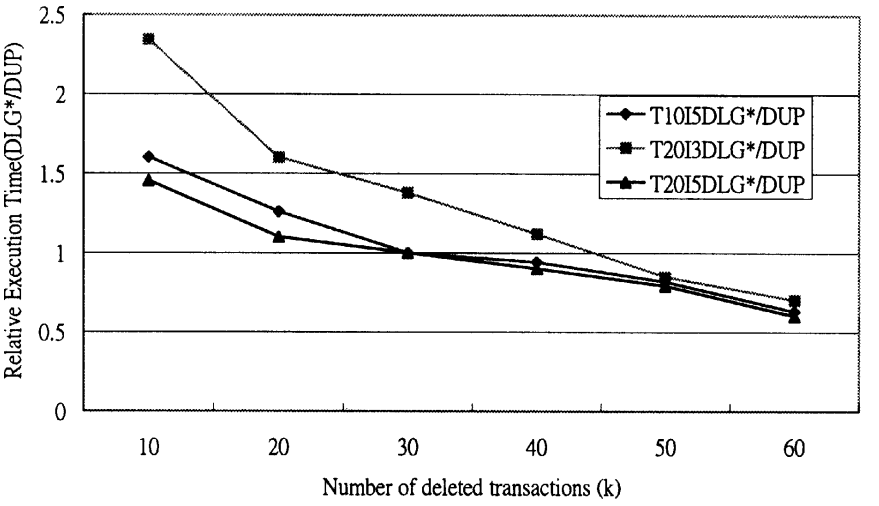



**Fig. 7.** Comparison of DUP with DLG*.

logical AND operations on bit vectors. The update algorithms DUP, which are based on the framework of DLG*, are further developed to solve the problem of maintaining association rules in the cases of insertion and deletion. The experimental results show that it significantly outperforms $FUP_2$, (Cheung et al, 1997), which is the most efficient update algorithm developed so far. DUP always performs faster than re-running DLG* even when $|d^+|$ is larger than the size of the original database $|D|$, and DUP keeps a better performance when $|d^-|$ is not greater than 30% of $|D|$.

To reduce the space for storing the bit vectors, some compression method has to be considered. The characteristic of the bit vectors is that they contain a large number of zeros. Taking advantage of this characteristic, we can substitute the sequence of zeros by recording the number of zeros in the sequence. To achieve a better compression rate, we should make the length of the sequence of zeros longer. We are currently investigating a method to rearrange the order of the transactions such that the bit vectors of most items can have a longer sequence of zeros.

## References

Agrawal R, Ghosh S, Imielinski T, Iyer B, Swami A (1992) An interval classifier for database mining applications. In Proceedings of the international conference on very large data bases, Vancouver, Canada, pp 560–573

Agrawal R et al (1993a) Database mining: a performance perspective. IEEE Transactions on Knowledge and Data Engineering 914–925

Agrawal R, Imielinski T, Swami A (1993b) Mining association rules between sets of items in large databases. In Proceedings of ACM SIGMOD, Washington DC, USA, pp 207–216

Agrawal R, Srikant R (1994) Fast algorithm for mining association rules. In Proceedings of the international conference on very large data bases, Santiago, Chile, pp 487–499

Agrawal R, Srikant R (1995) Mining sequential patterns. In Proceedings of the international conference on data engineering, Taipei, Taiwan, pp 3–14

Anwar TM, Beck HW, Navathe SB (1992) Knowledge mining by imprecise querying: a classification-

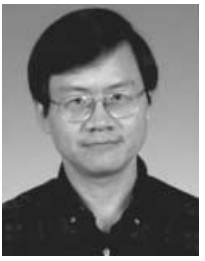
based approach. In Proceedings of the international conference on data engineering, February 1992, Tempe, Arizona, USA, pp 622–630
Brin S, Motwani R, Ullman J, Tsur S (1997) Dynamic itemset counting and implication rules for market basket data. In Proceedings of ACM SIGMOD, Tucson, Arizona, USA, pp 255–264
Cheung DW, Fu A W-C, Han J (1994) Knowledge discovery in databases: a rule-based attribute-oriented approach. In Proceedings of international symposium on methodologies for intelligent systems, Charlotte, NC, October 1994, pp 164–173
Cheung DW, Han J, Ng VT, Wong CY (1996a) Maintenance of discovered association rules in large databases: an incremental updating technique. In Proceedings of the international conference on data engineering, New Orleans, Louisiana, pp 106–114
Cheung DW, Ng VT, Tam BW (1996b) Maintenance of discovered knowledge: a case in multi-level association rules. In Proceedings of 2nd international conference on knowledge discovery and data mining, pp 307–310
Cheung DW, Lee SD, Kao B (1997) A general incremental technique for maintaining discovered association rules. In Proceedings of international conference on database systems for advanced applications, 1–4 April 1997, pp 185–194
Han J, Cai Y, Cerone N (1992) Knowledge discovery in databases: an attribute-oriented approach. In Proceedings of the international conference on very large data bases, Vancouver, Canada, pp 547–559
Han J, Cai Y, Cercone N (1993) Data-driven discovery of quantitative rules in relational databases. IEEE Transactions on Knowledge and Data Engineering 5:29–40
Han J, Fu Y (1995) Discovery of multiple-level association rules from large databases. In Proceedings of the international conference on very large data bases, Zurich, Switzerland, pp 420–431
Housma M, Swami A (1995) Set-oriented mining for association rules in relational databases. In Proceedings of the international conference on data engineering, Taipei, Taiwan, pp 25–33
Hwang H-Y, Fu W-C (1995) Efficient algorithm for attribute-oriented induction. In Proceedings of the first international conference of knowledge discovery and data mining, pp 168–173
Klemettinen M, Mannila H, Ronkainen P, Toivonen H, Verkamo AI (1994) Finding interesting rules from large sets of discovered association rules. In Proceedings of the 3rd international conference on information and knowledge management, Gaithersburg, MD, pp 401–408
Lee KL, Lee G, Chen ALP (1999) Efficient graph-based algorithm for discovering and maintaining knowledge in large databases. In Third Pacific-Asia conference, PAKDD-99 proceedings, Beijing, China, pp 409–419
Mannila H, Toivonen H, Verkamo AI (1994) Efficient algorithm for discovering association rules. In Proceedings of AAAI workshop on knowledge discovery in databases, pp 221–235
Meo R, Psaila G, Ceri S (1996) A new SQL-like operator for mining association rules. In Proceedings of the international conference on very large data bases, Mumbai, India, pp 122–133
Ng RT, Han J (1994) Efficient and effective clustering method for spatial data mining. In Proceedings of the international conference on very large data bases, Santiago, Chile, pp 144–155
Park JS, Chen MS, Yu PS (1995) An effective hash-based algorithm for mining association rules. In Proceedings of ACM SIGMOD, San Jose, California, pp 175–185
Savasere A, Omiecinski E, Navathe S (1995) An efficient algorithm for mining association rules in large databases. In Proceedings of the international conference on very large data bases, Zurich, Switzerland, pp 432–444
Simoudis E, Han J, Fayyad U (1996) Proceedings of the 2nd international conference on knowledge discovery and data mining (KDD96)
Srikant R, Agrawal R (1995) Mining generalized association rules. In Proceedings of the international conference on very large data bases, Zurich, Switzerland, pp 407–419
Srikant R, Agrawal R (1996a) Mining quantitative association rules in large relational tables. In Proceedings of ACM SIGMOD, Montreal, Quebec, Canada, pp 1–12
Srikant R, Agrawal R (1996b) Mining sequential patterns: generalizations and performance improvements. In Proceedings of the fifth international conference on extending database technology (EDBT), Avignon, France, March 1996
Tang J (1998) Using incremental pruning to increase the efficiency of dynamic itemset counting for mining association rules. In Proceedings of the sixth international conference on information and knowledge management (CIKM), Bethes da, Maryland, USA, pp 273–280
Toivonen H (1996) Sampling large databases for finding association rules. In Proceedings of the international conference on very large data bases, Mumbai, India, pp 134–145
Tsai PSM, Lee CC, Chen ALP (1999) An efficient approach for incremental association rule mining. In Third Pacific-Asia conference, PAKDD-99 proceedings, Beijing, China, pp 74–83
Wang JTL, Chirn G-W, Marr TG, Shapiro B, Shasha D, Zhang K (1994) Combinatorial pattern

discovery for scientific data: some preliminary results. In Proceedings of ACM SIGMOD, Minneapolis, Minnesota, pp 115–125
Yen SJ, Chen ALP (1995) An efficient algorithm for deriving compact rules from databases. In Proceedings of international conference on database systems for advanced applications, pp 364–371
Yen SJ, Chen ALP (1996a) The analysis of relationships in databases for rule derivation. Journal of Intelligent Information Systems, 7:1–24
Yen SJ, Chen ALP (1996b) An efficient approach to discovering knowledge from large databases. In Proceedings of the IEEE/ACM international conference on parallel and distributed information systems, pp 8–18

## Author Biographies

**Guanling Lee** received B.S. and M.S. degrees, both in computer science, from National Tsing Hua University, Taiwan, ROC, in 1995 and 1997, respectively. She is currently a Ph.D. candidate in the Department of Computer Science, National Tsing Hua University. Her research interests include location management in mobile environments and data scheduling on wireless channels.

**K. L. Lee** received B.S. and M.S. degrees, both in computer science, from National Tsing Hua University, Taiwan, ROC, in 1995 and 1997, respectively. She is currently a computer teacher at Chupei Senior High School.



**Arbee L. P. Chen** received a B.S. degree in computer science from National Chiao-Tung University, Taiwan, ROC, in 1977, and a Ph.D. degree in computer engineering from the University of Southern California in 1984. He joined National Tsing Hua University, Taiwan, as a National Science Council (NSC) sponsored Visiting Specialist in August 1990, and became a Professor in the Department of Computer Science in 1991. He was a Member of Technical Staff at Bell Communications Research, New Jersey, from 1987 to 1990, an Adjunct Associate Professor in the Department of Electrical Engineering and Computer Science, Polytechnic University, New York, and a Research Scientist at Unisys, California, from 1985 to 1986. His current research interests include multimedia databases, data mining and mobile computing. Dr Chen organized (and served as a Program Co-Chair) the 1995 IEEE Data Engineering Conference and 1999 International Conference on Database Systems for Advanced Applications (DASFAA) in Taiwan. He is a recipient of the NSC Distinguished Research Award.

*Correspondence and offprint requests to*: A. L. P. Chen, Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 300, ROC. Email: alpchen@cs.nthu.edu.tw