# Efficient Algorithms for Incremental Update of Frequent Sequences*

Minghua Zhang, Ben Kao, David Cheung, and Chi-Lap Yip

Department of Computer Science and Information Systems
The University of Hong Kong
{mhzhang, kao, dcheung, clyip}@csis.hku.hk

**Abstract.** Most of the works proposed so far on mining frequent sequences assume that the underlying database is static. However, in real life, the database is modified from time to time. This paper studies the problem of incremental update of frequent sequences when the database changes. We propose two efficient *incremental* algorithms `GSP+` and `MFS+`. Throught experimetns, we compare the performance of `GSP+` and `MFS+` with `GSP` and `MFS` — two efficient algorithms for mining frequent sequences. We show that `GSP+` and `MFS+` effectively reduce the CPU costs of their counterparts with only a small or even negative additional expense on I/O cost.
**keywords**: data mining, sequence, incremental update

## 1 Introduction

One of the many data mining problems is mining frequent sequences from transactional databases. The goal is to discover frequent sequences of events. The problem was first introduced by Agrawal and Srikant [1]. In their model, a database is a collection of transactions. Each transaction is a set of items (or an itemset) and is associated with a customer ID and a time ID. If one groups the transactions by their customer IDs, and then sorts the transactions of each group by their time IDs in increasing value, the database is transformed into a number of customer sequences. Each customer sequence shows the order of transactions a customer has conducted. Roughly speaking, the problem of mining frequent sequences is to discover "subsequences" (of itemsets) that occur frequently enough among all the customer sequences.

A few efficient algorithms for mining frequent sequences have been proposed, notably, `AprioriAll` [1], `GSP` [13], `SPADE` [15], `MFS` [16] and `PrefixSpan` [10]. The above studies assume the database is static, and even a small change in the database will require the algorithms to run again to get the updated frequent sequences. In practice, the content of a database changes continuously, and data mining has to be performed repeatedly. If each time we have to rerun the mining algorithms from scratch, it will be very inefficient.

* This research is supported by Hong Kong Research Grants Council grant HKU 7035/99E

In this paper we study the problem of incremental maintenance of frequent sequences. We assume that a mining exercise has been performed on an *old* database to obtain a set of frequent sequences. The old database is then updated by inserting new sequences and/or by deleting some old sequences. Our objective is to discover the set of frequent sequences in the *new* database efficiently, taking advantage of information that was obtained in a previous mining exercise. We propose two new algorithms `GSP+` and `MFS+` to solve the problem. The two algorithms are modified versions of `GSP` [13] and `MFS` [16]. The modification is made based on the following observations:

– If one knows about the support of a *frequent* sequence in the old database, then the sequence's support w.r.t. the new database can be deduced by scanning the inserted customer sequences and the deleted customer sequences. The portion (typically the majority) of the database that has not been changed needs not be processed.
– Given that a sequence is infrequent w.r.t. the old database, the sequence cannot become frequent unless its support in the inserted customer sequences is *large enough* and that its support in the deleted customer sequences is *small enough*. This observation allows us to determine whether a candidate sequence should be considered by "looking" at the *small* portion of the database that has been changed.
– If the old database and the new database share a non-trivial set of common sequences, then the set of frequent sequences found in the old database gives a good indication of which sequences in the new database are *likely* to be frequent. As we will see later, this allows us to generate long candidate sequences fairly early on in the mining algorithm. The effect is that fewer passes over the data are required compared with the mine-from-scratch approach. We can thus effectively reduce the I/O cost.

The rest of this paper is organized as follows. Section 2 gives a formal definition of the maintenance problem. In Section 3, we review some related works, in particular, `GSP` and `MFS`. The two new algorithms `GSP+` and `MFS+` are presented in Section 4. Experiment results comparing the performance of the algorithms are shown in Section 5. Finally, we conclude the paper in Section 6.

## 2 Model

In this section, we give a formal statement of the maintenance problem.

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of literals called items. An itemset $X$ is a set of items (hence, $X \subseteq I$). A sequence $s = \langle t_1, t_2, \ldots, t_n \rangle$ is an ordered set of transactions, where each transaction $t_i$ $(i = 1, 2, \ldots, n)$ is an itemset.

The length of a sequence $s$ is defined as the number of items contained in $s$. (If an item occurs several times in different itemsets of a sequence, the item is counted for each occurrence.) We use $|s|$ to represent the length of $s$.

Given two sequences $s_1 = \langle a_1, a_2, \ldots, a_n \rangle$ and $s_2 = \langle b_1, b_2, \ldots, b_l \rangle$, we say $s_1$ contains $s_2$ (or equivalently $s_2$ is a subsequence of $s_1$) if there exist integers $j_1, j_2, \ldots, j_l$, such that $1 \leq j_1 < j_2 < \ldots < j_l \leq n$ and $b_1 \subseteq a_{j_1}, b_2 \subseteq a_{j_2}, \ldots, b_l \subseteq a_{j_l}$. We represent this relationship by $s_2 \sqsubseteq s_1$.

In a sequence set $V$, a sequence $s \in V$ is *maximal* if $s$ is not a subsequence of any other sequence in $V$.

Given a sequence set $V$ and a sequence $s$, if there exists a sequence $s' \in V$ such that $s \sqsubseteq s'$, we write $s \vdash V$. Given a database $D$ of sequences, the *support count* of a sequence $s$, denoted by $\delta_D^s$, is defined as the number of sequences in $D$ that contain $s$. The *fraction* of sequences in $D$ that contain $s$ is called the *support* of $s$. If we use the symbol $|D|$ to denote the number of sequences in $D$ (or the size of $D$), we have: support of $s = \delta_D^s / |D|$.

If the support of $s$ is no less than a user specified support threshold $\rho_s$, $s$ is a frequent sequence. The problem of mining frequent sequences is to find all *maximal* frequent sequences in a database $D$. We use symbol $L_i$ to denote the set of all length-$i$ frequent sequences, and $L$ to denote the set of all frequent sequences.

Given a database $D$, we assume that a previous mining exercise has been executed to obtain the supports of the frequent sequences. The database $D$ is then updated by deleting a set of sequences $\Delta^-$ followed by inserting a set of sequences $\Delta^+$. Let us denote the updated database $D'$. Note that $D' = (D - \Delta^-) \cup \Delta^+$. We denote the set of unchanged sequences by $D^- = D - \Delta^- = D' - \Delta^+$. Since the relative order of the sequences within a database does not affect the mining results, we may assume (without loss of generality) that all of the deleted sequences are located at the beginning of the database and all of the new sequences are appended at the end, as illustrated in Figure 1.

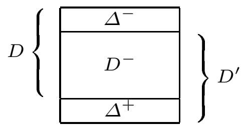


**Fig. 1.** Definitions of $D$, $D'$, $\Delta^-$, $D^-$ and $\Delta^+$

Our objective is to find all maximal frequent sequences in the database $D'$ given $\Delta^-$, $D^-$, $\Delta^+$, and the result of mining $D$.

## 3 Related Works

Agrawal and Srikant [1] first studied the problem of mining frequent sequences, and proposed 3 algorithms. Later, the same authors proposed a more efficient algorithm `GSP` [13]. Similar to the structure of the Apriori algorithm [11] for mining association rules, `GSP` starts by finding all frequent 1-sequences from the database. A set of candidate 2-sequences are then generated. The support counts of the candidate sequences are then counted by scanning the database once. Those frequent 2-sequences are then used to generate candidate 3-sequences, and so on. In general, `GSP` uses a function `GGen` to generate candidate ($k$+1)-sequences given the set of all frequent $k$-sequences. The algorithm terminates when no more frequent sequences are discovered during a database scan. Figure 2 shows the GSP algorithm (In the algorithm, $C_i$ means the set of candidate sequences of length-$i$). Details of the candidate generation function GGen is omitted due to space limitation. Readers are referred to [13] for more information. We note that if the database is huge and if it contains very long frequent sequences, the I/O cost of GSP is high.

```
1  Algorithm GSP($D$, $\rho_s$, $I$)
2    $C_1 := \{\langle\{i\}\rangle | i \in I\}$
3    Scan $D$ to compute $\delta_D^s$ for every sequence $s$ in $C_1$
4    $L_1 := \{s | s \in C_1, \delta_D^s \geq \rho_s \times |D|\}$
5    $i := 1$
6    while ($L_i \neq \emptyset$)
7      $C_{i+1} :=$ GGen($L_i$)
8      Scan $D$ to compute $\delta_D^s$ for every sequence $s$ in $C_{i+1}$
9      $L_{i+1} := \{s | s \in C_{i+1}, \delta_D^s \geq \rho_s \times |D|\}$
10     $i := i + 1$
11   Return $L_1 \cup L_2 \cup \ldots \cup L_{i-1}$
```

**Fig. 2.** Algorithm GSP

An interesting I/O-efficient algorithm, SPADE, was proposed by Zaki [15]. The idea is to first transform a sequence database from a “horizontal” representation to a “vertical” representation. SPADE works on the transformed database, and requires three database scans to find frequent sequences. While SPADE is an efficient algorithm, it requires the availability of the “vertical” database. Such a transformation requires either a high I/O cost or a lot of memory.

Based on SPADE, ISM [9] was put forward to address the maintenance problem of frequent sequences. By making use of a *sequence lattice*, which contains information about the old database, ISM can determine frequent sequences in the new database efficiently. Our algorithms differ from ISM in the following aspects. First, we consider both sequence insertions and deletions, while ISM handles insertion only. Second, similar to SPADE, ISM works on the “vertical” representation of the database. Our algorithms do not require the transformation.

PrefixSpan [10] is a newly devised efficient algorithm for mining frequent sequences. It needs to generate a number of intermediate databases during the process. If main memory is large enough, PrefixSpan is very efficient; otherwise it will require a high cost.

Another I/O-efficient algorithm MFS was proposed in [16]. MFS achieves I/O efficiency by making use of a candidate generation function MGen to generate sequences of various lengths given a set of frequent sequences of various lengths. This allows long sequences to be generated early, reducing the number of iterations and hence the I/O cost. For MFS to be effective, it requires an initial estimate ($S_{est}$) of the set of frequent sequences be available. $S_{est}$ could be obtained by mining a small sample of the database. For the maintenance problem, MFS can use the frequent sequences in the old database as $S_{est}$ directly. Hence, MFS is potentially efficient for the maintenance problem. Figure 3 shows the MFS algorithm. Details of the candidate generation function MGen is omitted due to space limitation. Readers are referred to [16] for more information.

```
1  Algorithm MFS($D$, $\rho_s$, $I$, $S_{est}$)
2    $MFSS := \emptyset$
3    $CandidateSet := \{\langle\{i\}\rangle | i \in I\} \cup \{s | s \vdash S_{est}, |s| > 1\}$
4    Scan $D$ to get $\delta_D^s$ for every sequence $s$ in $CandidateSet$
5    $NewFrequentSequences := \{s | s \in CandidateSet, \delta_D^s \geq \rho_s \times |D|\}$
6    $AlreadyCounted := \{s | s \vdash S_{est}, |s| > 1\}$
7    $Iteration := 2$
8    while ($NewFrequentSequences \neq \emptyset$)
9      //$Max(S)$ returns the set of all maximal sequences is $S$
10     $MFSS := Max(MFSS \cup NewFrequentSequences)$
11     $CandidateSet :=$ MGen($MFSS$, $Iteration$, $AlreadyCounted$)
12     Scan $D$ to get $\delta_D^s$ for every sequence $s$ in $CandidateSet$
13     $NewFrequentSequences := \{s | s \in CandidateSet, \delta_D^s \geq \rho_s \times |D|\}$
14     $Iteration := Iteration+1$
15   Return $MFSS$
```

**Fig. 3.** Algorithm MFS

Finally, a number of studies have been done on the problem of maintaining discovered association rules including [2, 3, 4, 5, 7, 8, 12, 14].

## 4 Algorithms

In this section we describe our incremental update algorithms GSP+ and MFS+. The idea is that, given a sequence $s$, we use the support count of $s$ in $D$ (if available) to deduce whether $s$ could have enough support in $D'$. In the deduction process, the portion of the database that has been changed, namely, $\Delta^-$ and $\Delta^+$, might have to be scanned. If we deduce that $s$ cannot be frequent in $D'$, $s$'s support in $D^-$ (the portion of the database that has not been changed) is not counted. If $D^-$ is large comparing with $\Delta^-$ and $\Delta^+$, the pruning technique saves much CPU cost.

Before we present the algorithms, let us consider a few mathematical equations that allow us to perform the pruning deductions.

First of all, since $D' = D - \Delta^- \cup \Delta^+ = D^- \cup \Delta^+$, we have, $\forall s$,

$$\delta_D^s = \delta_{D^-}^s + \delta_{\Delta^-}^s, \tag{1}$$

$$\delta_{D'}^s = \delta_{D^-}^s + \delta_{\Delta^+}^s, \tag{2}$$

$$\delta_{D'}^s = \delta_D^s + \delta_{\Delta^+}^s - \delta_{\Delta^-}^s. \tag{3}$$

Let us define $b_X^s = \min_{s'} \delta_X^{s'}$ for any sequence $s$ and database $X$, where $(s' \sqsubseteq s) \wedge (|s'| = |s| - 1)$. That is to say, if $s$ is a $k$-sequence, $b_X^s$ is the smallest support count of the $(k-1)$-subsequences of $s$ in the database $X$. Since the support count of a sequence $s$ must not be larger than the support count of any subsequence of $s$, $b_X^s$ is an *upper bound* of $\delta_X^s$.

The reason for considering $b_X^s$ is to allow us to estimate $\delta_X^s$ without counting it. As we will see later, under both GSP+ and MFS+, a candidate sequence $s$ is considered (and may have its support counted) only if all of $s$'s subsequences are frequent. Since frequent sequences would already have their supports counted (in order to conclude that they are frequent), we would have the necessary information to deduce $b^s_X$ when we consider $s$.

To illustrate how the bound is used in the deduction, let us consider the following simple Lemma:

**Lemma 1** *If a sequence $s$ is frequent in $D'$, then $\delta^s_D + b^s_{\Delta^+} \geq \delta^s_D + b^s_{\Delta^+} - \delta^s_{\Delta^-} \geq |D'| \times \rho_s$.*

Proof: If $s$ is frequent in $D'$, we have,

$$\begin{aligned} |D'| \times \rho_s &\leq \delta^s_{D'} && \text{(by definition)} \\ &= \delta^s_D + \delta^s_{\Delta^+} - \delta^s_{\Delta^-} && \text{(by Equation 3)} \\ &\leq \delta^s_D + b^s_{\Delta^+} - \delta^s_{\Delta^-} \\ &\leq \delta^s_D + b^s_{\Delta^+}. \end{aligned}$$

Given a sequence $s$, if $s$ is frequent in $D$, we know $\delta^s_D$. If $b^s_{\Delta^+}$ is available, we can compute $\delta^s_D + b^s_{\Delta^+}$ and conclude that $s$ is infrequent in $D'$ if the quantity is less than $|D'| \times \rho_s$. Otherwise, we scan $\Delta^-$ to find $\delta^s_{\Delta^-}$. We conclude that $s$ is infrequent in $D'$ if $\delta^s_D + b^s_{\Delta^+} - \delta^s_{\Delta^-}$ is less than the required support count ($|D'| \times \rho_s$). Note that in the above cases, the deduction is made without processing $D^-$ or $\Delta^+$.

If a sequence $s$ is not frequent in $D$, $\delta^s_D$ is unavailable. The pruning tricks derived from Lemma 1 are thus not applicable. However, being infrequent in $D$ means that the support of $s$ (in $D$) is *small*. The following Lemma allows us to prune those sequences.

**Lemma 2** *If a sequence $s$ is frequent in $D'$ but not in $D$, then $b^s_{\Delta^+} \geq b^s_{\Delta^+} - \delta^s_{\Delta^-} \geq \delta^s_{\Delta^+} - \delta^s_{\Delta^-} > (|\Delta^+| - |\Delta^-|) \times \rho_s$.*

Proof: If $s$ is frequent in $D'$ but not in $D$, we have, by definition:

$$\begin{aligned} \delta^s_{D'} &\geq |D'| \times \rho_s = (|D^-| + |\Delta^+|) \times \rho_s, \\ \delta^s_D &< |D| \times \rho_s = (|D^-| + |\Delta^-|) \times \rho_s. \end{aligned}$$

Hence,

$$\begin{aligned} \delta^s_{D'} - \delta^s_D &> (|D^-| + |\Delta^+|) \times \rho_s - (|D^-| + |\Delta^-|) \times \rho_s \\ \delta^s_{\Delta^+} - \delta^s_{\Delta^-} &> (|\Delta^+| - |\Delta^-|) \times \rho_s \quad \text{(by Equation 3).} \end{aligned}$$

Also,

$$b^s_{\Delta^+} \geq b^s_{\Delta^+} - \delta^s_{\Delta^-} \geq \delta^s_{\Delta^+} - \delta^s_{\Delta^-}.$$

Lemma 2 thus follows.

Given a candidate sequence $s$ that is not frequent in the old database $D$, we first compare $b^s_{\Delta^+}$ against $(|\Delta^+| - |\Delta^-|) \times \rho_s$. If $b^s_{\Delta^+}$ is not large enough, $s$ cannot be frequent in $D'$, and hence $s$ can be pruned. Otherwise, we scan $\Delta^-$ to find $b^s_{\Delta^+} - \delta^s_{\Delta^-}$ and see if $s$ can be pruned. If not, we scan $\Delta^+$ and consider $\delta^s_{\Delta^+} - \delta^s_{\Delta^-}$.

Similar to Lemma 1, Lemma 2 allows us to prune some candidate sequences without completely counting their supports in the new database.

## 4.1 GSP+

Based on the Lemmas, we modify GSP to incorporate the pruning techniques mentioned. The new algorithm, GSP+, is shown in Figure 4.

```
Algorithm GSP+($\Delta^-$, $D^-$, $\Delta^+$, $\rho_s$, $I$)
  $C_1 := \{\langle\{i\}\rangle | i \in I\}$
  Scan $\Delta^-$, $D^-$, $\Delta^+$ to get $\delta^s_{\Delta^-}$, $\delta^s_{D^-}$, $\delta^s_{\Delta^+}$ for each $s \in C_1$
  $L_1 := \{s | s \in C_1, \delta^s_{D^-} + \delta^s_{\Delta^+} \geq |D'| \times \rho_s\}$
  $i := 2$
  $C_i :=$ GGen($L_{i-1}$)
  while ($C_i \neq \emptyset$)
    calculate $b^s_{\Delta^+}$ for each $s \in C_i$
    $candp = \{s | s \in C_i, s\ is\ frequent\ in\ D\}$, $candq = C_i - candp$
    $\forall s \in candp$, if $\delta^s_D + b^s_{\Delta^+} < |D'| \times \rho_s$, delete $s$ from $candp$
    $\forall s \in candq$, if $b^s_{\Delta^+} \leq (|\Delta^+| - |\Delta^-|) \times \rho_s$, delete $s$ from $candq$
    scan $\Delta^-$ to count $\delta^s_{\Delta^-}$ for each $s$ in $candp$ and $candq$
    $\forall s \in candp$, if $\delta^s_D + b^s_{\Delta^+} - \delta^s_{\Delta^-} < |D'| \times \rho_s$, delete $s$ from $candp$
    $\forall s \in candq$, if $b^s_{\Delta^+} - \delta^s_{\Delta^-} \leq (|\Delta^+| - |\Delta^-|) \times \rho_s$, delete $s$ from $candq$
    scan $\Delta^+$ to count $\delta^s_{\Delta^+}$ for each $s$ in $candp$ and $candq$
    $\forall s \in candq$, if $\delta^s_{\Delta^+} - \delta^s_{\Delta^-} \leq (|\Delta^+| - |\Delta^-|) \times \rho_s$, delete $s$ from $candq$
    scan $D^-$ to count $\delta^s_{D^-}$ for each $s$ in $candq$
    $L_i = \emptyset$
    $\forall s \in candp$, if $\delta^s_D + \delta^s_{\Delta^+} - \delta^s_{\Delta^-} \geq |D'| \times \rho_s$, insert $s$ into $L_i$
    $\forall s \in candq$, if $\delta^s_{D^-} + \delta^s_{\Delta^+} \geq |D'| \times \rho_s$, insert $s$ into $L_i$
    if ($|L_i| > i$), $C_{i+1} :=$ GGen($L_i$)
    $i := i + 1$
  Return $L_1 \cup L_2 \cup \ldots \cup L_{i-1}$
```

**Fig. 4.** Algorithm GSP+

GSP+ shares the same structure with GSP. GSP+ is an iterative algorithm. During each iteration, GSP+ also uses GGen to generate $C_i$ (set of candidate sequences of length-$i$) based on $L_{i-1}$. Before the database is scanned to count the support of the candidate sequences, the pruning tests derived from Lemmas 1 and 2 are applied. Depending on the test results, the datasets $\Delta^-$ and/or $\Delta^+$ may have to be processed to count the support of a candidate sequence. GSP+ carefully controls when such countings are necessary. If all pruning tests fail on a candidate sequence $s$, GSP+ checks whether $s$ is frequent in $D$. If so, $\delta^s_D$ is available. Hence, $\delta^s_{D'}$ can be computed by $\delta^s_D + \delta^s_{\Delta^+} - \delta^s_{\Delta^-}$. Finally, if $s$ is not frequent in $D$, the unchanged part of the database, $D^-$, is scanned to find out the actual support of $s$. Since $D^-$ is typically much larger than $\Delta^+$ and $\Delta^-$, saving is achieved by avoiding processing $D^-$ for certain candidate sequences. As we will see later in Section 5, the pruning tests can prune up to 60% of the candidate sequences in our experiment setting. The tests are thus quite effective.

## 4.2 MFS+

The pruning tests can also be applied to MFS. We call the resulting algorithm MFS+ (see Figure 5). The interesting thing about MFS+ is that it uses the set of frequent sequences ($L_{old}$) of the old database $D$ as an initial estimate of the set of frequent sequences of the new database $D'$. These sequences together with all possible 1-sequences are put into a candidate set *CandidateSet*. It then scans $\Delta^-$, $D^-$, and $\Delta^+$ to obtain $\delta^s_{\Delta^-}$, $\delta^s_{D^-}$, and $\delta^s_{\Delta^+}$ for each sequence $s$ in *CandidateSet*. From these counts, we can deduce which sequences in *CandidateSet* are frequent in $D'$. The *maximals* of such frequent sequences are put into the set *MFSS*. We can consider the set *MFSS* as the set of maximal frequent sequences that `MFS+` knows given the information that `MFS+` has obtained so far. `MFS+` then executes a loop, trying to refine *MFSS*. During each iteration, `MFS+` uses `MGen` to generate a set of candidate sequences *CandidateSet* from *MFSS*. `MFS+` then deduces which candidate sequences must not be frequent by applying the pruning tests. In the process, the datasets $\Delta^-$ and $\Delta^+$ may have to be scanned. For those sequences that are not pruned by the tests, $D^-$ is scanned to obtain their exact support counts. Since sequences that are originally frequent in the old database $D$ have already had their supports (w.r.t $D'$) counted in the initial part of `MFS+`, all candidate sequences considered by `MFS+` in the loop section are infrequent in $D$. Hence, only those pruning tests resulting from Lemma 2 are used. `MFS+` terminates when no refinement is made to *MFSS* during an iteration.

```
1  Algorithm MFS+($\Delta^-$, $D^-$, $\Delta^+$, $\rho_s$, $I$, $L_{old}$)
2    $MFSS := \emptyset$
3    $CandidateSet := \{\langle\{i\}\rangle | i \in I\} \cup \{s | s \vdash L_{old}, |s| > 1\}$
4    Scan $\Delta^-$, $D^-$, $\Delta^+$ to get $\delta^s_{\Delta^-}$, $\delta^s_{D^-}$, $\delta^s_{\Delta^+}$ for each $s \in CandidateSet$
5    $NewFrequentSequences := \{s | s \in CandidateSet, \delta^s_{D^-} + \delta^s_{\Delta^+} \geq |D'| \times \rho_s\}$
6    $AlreadyCounted := \{s | s \vdash L_{old}, |s| > 1\}$
7    $Iteration := 2$
8    while ($NewFrequentSequences \neq \emptyset$)
9      //$Max(S)$ returns the set of all maximal sequences is $S$
10     $MFSS := Max(MFSS \cup NewFrequentSequences)$
11     $CandidateSet :=$ MGen($MFSS$, $Iteration$, $AlreadyCounted$)
12     calculate $b^s_{\Delta^+}$ for each $s \in CandidateSet$
13     $\forall s \in CandidateSet$, if $b^s_{\Delta^+} \leq (|\Delta^+| - |\Delta^-|) \times \rho_s$, delete $s$ from $CandidateSet$
14     scan $\Delta^-$ to count $\delta^s_{\Delta^-}$ for each $s \in CandidateSet$
15     $\forall s \in CandidateSet$,
         if $b^s_{\Delta^+} - \delta^s_{\Delta^-} \leq (|\Delta^+| - |\Delta^-|) \times \rho_s$, delete $s$ from $CandidateSet$
16     scan $\Delta^+$ to count $\delta^s_{\Delta^+}$ for each $s \in CandidateSet$
17     $\forall s \in CandidateSet$,
         if $\delta^s_{\Delta^+} - \delta^s_{\Delta^-} \leq (|\Delta^+| - |\Delta^-|) \times \rho_s$, delete $s$ from $CandidateSet$
18     scan $D^-$ to count $\delta^s_{D^-}$ for each $s \in CandidateSet$
19     $NewFrequentSequences := \{s | s \in CandidateSet, \delta^s_{D^-} + \delta^s_{\Delta^+} \geq |D'| \times \rho_s\}$
20     $Iteration := Iteration + 1$
21   Return $MFSS$
```

**Fig. 5.** Algorithm `MFS+`

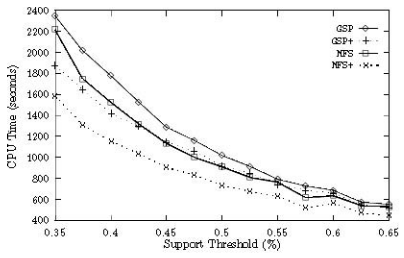
## 5 Results

We performed a number of experiments comparing the performance of `GSP+` and `MFS+` with `GSP` and `MFS`. In this section we present some representative results from our experiments. The test databases are synthetic data generated by a generator provided in the IBM Quest data mining project. Readers are referred to [6] for the details of the data generator. The experiments were performed on a 700MHz PIII Xeon machine with 4GB main memory running Solaris 8.

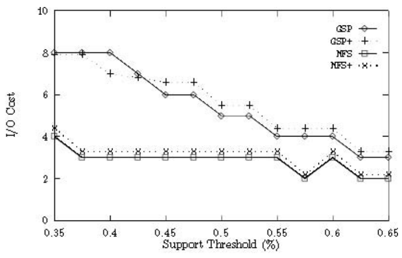
## 5.1 Performance

In our first experiment, we used a database $D$ of 1,500,000 sequences. We first executed a sequence mining program on $D$ to obtain all frequent sequences and their support counts. Then, 10% (150,000) of the sequences in $D$ are deleted. These sequences form the dataset $\Delta^-$. Another 10% (150,000) sequences, which form the set $\Delta^+$, are added into $D$ to form the updated database $D'$. After that, four algorithms `GSP`, `MFS`, `GSP+`, and `MFS+` were executed to mine $D'$. We did the experiment using different values of support thresholds ($0.35\% \leq \rho_s \leq 0.65\%$).

We compared the CPU costs and I/O costs of the four algorithms. I/O cost is measured in terms of database scans normalized by the size of $D'$. For example, if `GSP` scans $D'$ 8 times, then the I/O cost of `GSP` is 8. For an incremental algorithm, if it reads $\Delta^-$ $n_1$ times, $D^-$ $n_2$ times, and $\Delta^+$ $n_3$ times, then the I/O cost is $(n_1|\Delta^-| + n_2|D^-| + n_3|\Delta^+|)/|D'|$. We notice that while the I/O costs of `GSP` and `MFS` are integral numbers (because $D'$ is the only dataset they read), those of `GSP+` and `MFS+` could be fractional. Figure 6 shows the experiment results.



(a) CPU Costs



(b) I/O Costs

**Fig. 6.** Comparison of four algorithms under different support thresholds

Figure 6(a) shows that as $\rho_s$ increases, the CPU costs of all four algorithms decrease. This is because a larger $\rho_s$ means fewer frequent sequences, and thus fewer candidate sequences whose supports need to be counted.

Among the four algorithms, `GSP` has the highest CPU cost. Using the set of frequent sequences in $D$ as an initial estimate allows `MFS` to disover the long frequent sequences early. As we will see shortly, this results in fewer iterations for `MFS`. During each iteration, each sequence $s$ in the database is matched against a set of candidate sequences to see which candidates are contained in $s$, and to increase their support counts. Since `MFS` performs fewer database scans compared with `GSP`, the database sequences are matched against candidate sequences fewer times. The overall effect is a lower CPU cost for `MFS` compared with `GSP`.

From the figure, we also see that GSP+ and MFS+ need less CPU time than their counterparts — GSP and MFS. The saving is obtained by the pruning effect of the incremental algorithms. Some candidate sequences are pruned by processing only $\Delta^-$ and/or $\Delta^+$; the set $D^-$ is avoided. In our experiment setting, the size of $D^-$ is 9 times that of $\Delta^+$ and $\Delta^-$. Avoid counting the supports of the pruned candidate sequences in $D^-$ results in a significant CPU cost reduction.

Table 1 shows the effectiveness of the pruning tests of GSP+. The total number of candidate sequences processed by pruning tests is shown in the second row, and the number of them requiring the scanning of $D^-$ is shown in the third row. We can see that GSP+ needs to process $D^-$ to obtain the support counts of only about 40% of all candidate sequences. This accounts for the saving in CPU cost achieved by GSP+ over GSP. Similarly, MFS+ outperforms MFS, mainly due to candidate pruning.

| $\rho_s$ | 0.35% | 0.4% | 0.45% | 0.5% | 0.55% | 0.6% | 0.65% |
|---|---|---|---|---|---|---|---|
| Total # of candidates | 34,065 | 18,356 | 10,024 | 5,812 | 3,365 | 2,053 | 1,160 |
| Those requiring $\delta^s_{D^-}$ | 13,042 | 7,161 | 3,966 | 2,313 | 1,353 | 867 | 509 |
| Percentage (row3/row2) | 38% | 39% | 40% | 40% | 40% | 42% | 44% |

**Table 1.** Effectiveness of prunning tests

Finally, when $\rho_s$ becomes large, the CPU times of the algorithms are roughly the same. This is because a large support threshold implies short and few frequent sequences. In such a case, GSP and MFS take similar number of iterations, and hence the CPU saving of MFS over GSP is diminished. Moreover, there are much fewer candidate sequences when $\rho_s$ is large, and there are much fewer opportunities for the incremental algorithms to prune candidate sequences.

Figure 6(b) shows the I/O costs of the four algorithms. We see that as $\rho_s$ increases, the I/O costs of the algorithms decrease. This is due to the fact that a larger $\rho_s$ leads to shorter frequent sequences. Hence, the number of iterations (and database scans) needed is small.

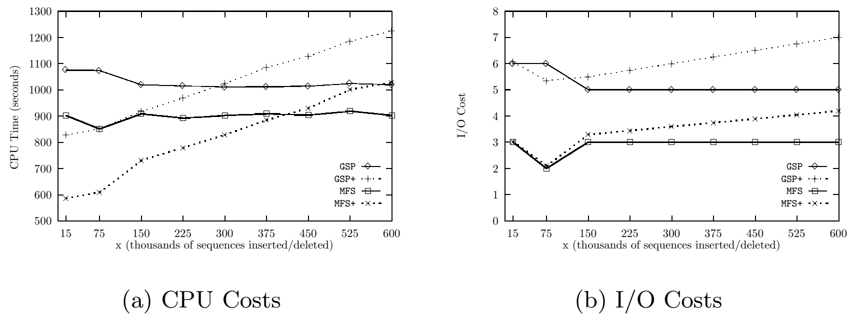
Comparing GSP and MFS, we see that MFS is a very I/O-efficient algorithm. Because MFS uses the frequent sequences in $D$ as an estimate of those in $D'$, which gives MFS a head start and allows MFS to discover all maximal frequent sequences in much fewer database scans.

The incremental algorithms generally require a slightly higher I/O cost than their non-incremental counterparts. The reason is that the incremental algorithms scan and process $\Delta^-$ to make pruning deductions, which is not needed by GSP and MFS. However, in some cases, the pruning tests remove all candidate sequences during an iteration of the algorithm. In such cases, incremental algorithms save some database passes. For example, when $\rho_s < 0.45\%$, GSP+ has a smaller I/O cost than GSP (see Figure 6(b)).

From Figure 6 we can conclude that GSP has a high CPU cost and a high I/O cost. GSP+ reduces the CPU requirement but does not help in terms of I/O. MFS is very I/O-efficient and it also performs better than GSP in terms of CPU cost. MFS+ is the overall winner. It requires the least amount of CPU time and its I/O cost is comparable to that of MFS.

### 5.2 Varying $|\Delta^{+}|$ and $|\Delta^{-}|$

As we have discussed, GSP+ and MFS+ achieve efficiency by processing $\Delta^{+}$ and $\Delta^{-}$ to prune candidate sequences. The performance of GSP+ and MFS+ is thus dependent on the sizes of $\Delta^{+}$ and $\Delta^{-}$. we ran an experiment to study how $|\Delta^{+}|$ and $|\Delta^{-}|$ affect the performance of GSP+ and MFS+. In the experiment, we set $|D| = |D'| = 1,500,000$ sequences, $\rho_s = 0.5\%$, and varied $|\Delta^{+}|$ and $|\Delta^{-}|$ from 15,000 to 600,000 (1% – 40% of $|D|$). Figure 7 shows the experiment results.



(a) CPU Costs (b) I/O Costs

**Fig. 7.** Comparison of four algorithms under different insertion and deletion sizes

Figure 7(a) shows that the CPU costs of GSP and MFS stay relatively steady. This is expected since $|D'|$ does not change. The CPU costs of GSP+ and MFS+, on the other hand, increase linearly with the size of $\Delta^{+}$ and $\Delta^{-}$. This increase is due to a longer processing time taken by GSP+ and MFS+ to deal with $\Delta^{+}$ and $\Delta^{-}$. From the figure, we see that MFS+ outperforms others even when the database is changed substantially. In particular, if $|\Delta^{+}|$ and $|\Delta^{-}|$ are less than 375,000 sequences (or 25% of $|D|$), MFS+ is the most CPU-efficient algorithm in our experiment. For cases in which only a small fraction of the database is changed, the incremental algorithms can achieve significant performance gains.

As we have discussed previously, GSP+ and MFS+ usually have slightly higher I/O costs than their non-incremental counterparts, since they have to read $\Delta^{-}$. Therefore the gap between GSP and GSP+ (and also that between MFS and MFS+) widens as $|\Delta^{-}|$ increases (see Figure 7(b)).

## 6 Conclusions

This paper studies the maintenance problem of frequent sequences. We proposed two incremental algorithms, GSP+ and MFS+. We analyzed their performance in terms of CPU efficiency and I/O efficiency, and compared them with GSP and MFS. Our experiment results show that the incremental algorithms outperform their non-incremental counterparts in CPU cost at the expense of a small penalty in I/O cost. The performance gain is obtained by pruning candidate sequences to avoid processing the bulk of the database. The gain is thus most prominent when the database contains long and numerous frequent sequences, since in such cases there are many candidate sequences to consider and the impact of the pruning technique is most significant.

## References

[1] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proc. of the 11th Int'l Conference on Data Engineering*, Taipei, Taiwan, March 1995.

[2] Necip Fazil Ayan, Abdullah Uz Tansel, and Erol Arkun. An efficient algorithm to update large itemsets with early pruning. In *Proc. 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Diego, CA USA, August 1999.

[3] D. W. Cheung, J. Han, V. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating techniques. In *Proc. 12th IEEE International Conference on Data Engineering (ICDE)*, New Orleans, Louisiana, U.S.A, March 1996.

[4] D. W. Cheung, S. D. Lee, and B. Kao. A general incremental technique for maintaining discovered association rules. In *Proc. International Conference On Database Systems For Advanced Applications (DASFAA)*, Melbourne, Australia, April 1997.

[5] D. W. Cheung, V. Ng, and B. W. Tam. Maintenance of discovered knowledge: A case in multi-level association rules. In *Proc. Second International Conference on Knowledge Discovery and Data Mining (KDD)*, Portland, Oregon, August 1996.

[6] http://www.almaden.ibm.com/cs/quest/.

[7] S. D. Lee and D. W. Cheung. Maintenance of discovered association rules: When to update? In *Proc. 1997 ACM-SIGMOD Workshop on Data Mining and Knowledge Discovery (DMKD)*, Tucson, Arizona, May 1997.

[8] E. Omiecinski and A. Savasere. Efficient mining of association rules in large dynamic databases. In *Proc. BNCOD'98*, 1998.

[9] S. Parthasarathy, M. J. Zaki, M. Ogihara, and S. Dwarkadas. Incremental and interactive sequence mining. In *Proceedings of the 1999 ACM 8th International Conference on Information and Knowledge Management (CIKM'99)*, Kansas City, MO USA, November 1999.

[10] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proc. 17th IEEE International Conference on Data Engineering (ICDE)*, Heidelberg, Germany, April 2001.

[11] T. Imielinski R. Agrawal and A. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD International Conference on Management of Data*, page 207, Washington, D.C., May 1993.

[12] N. L. Sarda and N. V. Srinivas. An adaptive algorithm for incremental mining of association rules. In *Proc. DEXA Workshop'98*, 1998.

[13] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. of the 5th Conference on Extending Database Technology (EDBT)*, Avignion, France, March 1996.

[14] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. An efficient algorithm for the incremental updation of association rules in large databases. In *Proc. KDD'97*, 1997.

[15] Mohammed J. Zaki. Efficient enumeration of frequent sequences. In *Proceedings of the 1998 ACM 7th International Conference on Information and Knowledge Management(CIKM'98)*, Washington, United States, November 1998.

[16] Minghua Zhang, Ben Kao, C.L. Yip, and David Cheung. A GSP-based efficient algorithm for mining frequent sequences. In *Proc. of IC-AI'2001*, Las Vegas, Nevada, USA, June 2001.