An efficient algorithm for mining temporal high utility itemsets from data streams

Chun-Jung Chu a, Vincent S. Tseng b,*, Tyne Liang a

a Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan, ROC
b Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, Taiwan, ROC

Received 10 October 2006; received in revised form 19 July 2007; accepted 21 July 2007
Available online 3 August 2007

Abstract

Utility of an itemset is considered as the value of this itemset, and utility mining aims at identifying the itemsets with high utilities. The temporal high utility itemsets are the itemsets whose support is larger than a pre-specified threshold in current time window of the data stream. Discovery of temporal high utility itemsets is an important process for mining interesting patterns like association rules from data streams. In this paper, we propose a novel method, namely THUI (Temporal High Utility Itemsets-Mine), for mining temporal high utility itemsets from data streams efficiently and effectively. To the best of our knowledge, this is the first work on mining temporal high utility itemsets from data streams. The novel contribution of THUI-Mine is that it can effectively identify the temporal high utility itemsets by generating fewer candidate itemsets such that the execution time can be reduced substantially in mining all high utility itemsets in data streams. In this way, the process of discovering all temporal high utility itemsets under all time windows of data streams can be achieved effectively with less memory space and execution time. This meets the critical requirements on time and space efficiency for mining data streams. Through experimental evaluation, THUI-Mine is shown to significantly outperform other existing methods like Two-Phase algorithm under various experimental conditions.

© 2007 Elsevier Inc. All rights reserved.

Keywords: Utility mining; Temporal high utility itemsets; Data stream mining; Association rules

1. Introduction

The mining of association rules for finding the relationship between data items in large databases is a well studied technique in the data mining field with representative methods like Apriori (Agrawal et al., 1993, 1996). The problem of mining association rules can be decomposed into two steps. The first step involves finding of all frequent itemsets (or say large itemsets) in databases. Once the frequent itemsets are found, generating association rules is straightforward and can be accomplished in linear time.

An important research issue extended from the mining of association rules is the discovery of temporal association patterns in data streams due to the wide applications on various domains. Temporal data mining can be defined as the activity of discovering interesting correlations or patterns in large sets of temporal data accumulated for other purposes (Bettini et al., 1996). For a database with a specified transaction window size, we may use an algorithm like Apriori to obtain frequent itemsets from the database. For time-variant data streams, there is a strong demand to develop an efficient and effective method to mine various temporal patterns (Das et al., 1998). However, most methods designed for traditional databases cannot be directly applied to the mining of temporal patterns in data streams because of their high complexity.

In many applications, we would like to mine temporal association patterns from the most recent data in data
streams. That is, in temporal data mining, one should not only include new data (i.e., data in the new hour) but also remove the old data (i.e., data in the most obsolete hour) from the mining process. Without loss of generality, consider a typical market-basket application as illustrated in Fig. 1, where the transactional data of customer purchases are shown as time advances.

In Fig. 1, for example, data was accumulated as a function of time. Data obtained prior to some specified time interval in the past becomes useless for reference. People might be most interested in the temporal association patterns in the latest three hours (i.e., db^{3,5}) as shown in Fig. 1. It can be seen that in such a data stream environment it is intrinsically difficult to conduct the frequent pattern identification due to the constraints of limited time and memory space. Furthermore, it takes considerable time to find temporal frequent itemsets in different time windows. However, the frequency of an itemset may not be a sufficient indicator of interestingness, because it only reflects the number of transactions in the database that contain the itemset. It does not reveal the utility of an itemset, which can be measured in terms of cost, profit or other expressions of user preferences. On the other hand, frequent itemsets may only contribute a small portion of the overall profit, whereas non-frequent itemsets may contribute a large portion of the profit. In reality, a retail business may be interested in identifying its most valuable customers (customers who contribute a major fraction of the profits to the company). Hence, frequency is not sufficient to answer questions such as whether an itemset is highly profitable, or whether an itemset has a strong impact. Utility mining is thus useful in a wide range of practical applications and was recently studied in Chan et al. (2003), Liu et al. (2005) and Yao et al. (2004). This also motivates our research in developing a new scheme for finding temporal high utility itemsets (THUI) from data streams.

Recently, a utility mining model was defined in Yao et al. (2004). Utility is considered as a measure of how “useful” (e.g., “profitable”) an itemset is. The definition of utility \( u(X) \) of an itemset \( X \) is the sum of the utilities of \( X \) in all transactions containing \( X \). The goal of utility mining is to identify high utility itemsets which drive a large portion of the total utility. Traditional association rules mining models assume that the utility of each item is always 1 and the sales quantity is either 0 or 1, thus it is only a special case of utility mining, where the utility or the sales quantity of each item could be any number. If \( u(X) \) is greater than a utility threshold, \( X \) is a high utility itemset. Otherwise, it is a low utility itemset. 1 is an example of utility mining in a transaction database. The number in each transaction in Table 1 (panel a) is the sales volume of each item, and the utility of each item is listed in Table 2 (panel b). For example, \( u\{B, D\} = (6 \times 10 + 1 \times 6) + (1 \times 10 + 7 \times 6) + (3 \times 10 + 2 \times 6) = 160 \). \{B, D\} is considered as a high utility itemset if the utility threshold is set at 120.

However, a high utility itemset may consist of some low utility items. Another attempt is to adopt the level-wise searching schema that exists in fast algorithms such as Apriori (Agrawal and Srikant, 1995). However, this algorithm does not apply to the utility mining model. For example, consider \( u(D) = 84 < 120 \) as shown in 1. \( D \) is a low utility item. However, its superset \( \{B, D\} \) is a high utility itemset. If Apriori is used to find high utility itemsets, all combinations of the items must be generated. Moreover, the number of candidates is prohibitively large in discovering a long pattern. The cost of either computation time or memory will be intolerable, regardless of what implementation is applied. The challenge of utility mining is not only in restricting the size of the candidate set but also in simplifying the computation for calculating the utility. Another challenge of utility mining is how to find temporal high utility itemsets from data streams as time advances.

In this paper, we explore the issue of efficiently mining high utility itemsets in temporal databases like data streams. We propose an algorithm named \( THUI\)-Mine that can discover temporal high utility itemsets from data streams efficiently and effectively. The underlying idea of \( THUI\)-Mine algorithm is to integrate the advantages of Two-Phase algorithm (Liu et al., 2005) and SWF algorithm (Lee et al., 2001) with augmentation of the incremental mining techniques for mining temporal high utility itemsets efficiently. The novel contribution of \( THUI\)-Mine is that it can efficiently identify the utility itemsets in data streams so that the execution time for mining high utility itemsets can be substantially reduced. That is, \( THUI\)-Mine can discover the temporal high utility itemsets in current time window and also discover the temporal high utility itemsets in the next time window with limited memory space and less computation time by sliding window filter method. In this way, the process of discovering all temporal high utility itemsets under all time windows of data streams can be achieved effectively under less memory space and execution time. This meets the critical requirements of time and space efficiency for mining data streams. Through experimental evaluation, \( THUI\)-Mine is shown to produce fewer candidate itemsets in finding the temporal high utility itemsets, so it outperforms other methods in terms of execution efficiency. It is observed that the average improvement of \( THUI\)-Mine over Two-Phase algorithm reaches to about 67%. Moreover, it also achieves high scalability in dealing with large databases. To our best knowledge, this is the...
first work on mining temporal high utility itemsets from data streams.

The rest of this paper is organized as follows: Section 2 provides an overview of the related work. Section 3 describes the proposed approach, THUI-Mine, for finding the temporal high utility itemsets. In Section 4, we describe the experimental results for evaluating the proposed method. The conclusion of the paper is provided in Section 5.

2. Related work

In association with rules mining, Apriori (Agrawal and Srikant, 1995), DHP (Park et al., 1997) and partition-based ones (Lin and Dunham, 1998; Savasere et al., 1995) were proposed to find frequent itemsets. Many important applications have called for the need of incremental mining due to the increasing use of record-based databases to which data are being added continuously. Many algorithms like FUP (Cheung et al., 1996), FUP2 (Cheung et al., 1997) and UWEP (Ayn et al., 1999; Ayn et al., 1999) have been proposed to find frequent itemsets in incremental databases. The FUP algorithm updates the association rules in a database when new transactions are added to the database. Algorithm FUP is based on the framework of Apriori and is designed to discover the new frequent itemsets iteratively. The idea is to store the counts of all the frequent itemsets found in a previous mining operation. Using these stored counts and examining the newly added transactions, the overall count of these candidate itemsets are then obtained by scanning the original database. An extension to the work in Cheung et al. (1996) was reported in Cheung et al. (1997) where the authors propose an algorithm FUP2 for updating the existing association rules when transactions are added to and deleted from the 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 minimized.

In recent years, processing data from data streams becomes a popular topic in data mining. A number of algorithms like Lossy Counting (Manku and Motwani, 2002), FTP-DS (Teng et al., 2003) and RAM-DS (Teng et al., 2004) have been proposed to process data in data streams. Lossy Counting divided incoming stream conceptually into buckets. It uses bucket boundaries and maximal possible error to update or delete the itemsets with frequency for mining frequent itemsets. FTP-DS is a regression-based algorithm for mining frequent temporal patterns from data streams. A wavelet-based algorithm, RAM-DS, performs
pattern mining tasks for data streams by exploring both temporal and support count granularities.

Some algorithms like SWF (Lee et al., 2001) and Moment (Chi et al., 2004) were proposed to find frequent itemsets over a stream sliding window. By partitioning a transaction database into several partitions, algorithm SWF employs a filtering threshold in each partition to deal with the candidate itemset generation. The Moment algorithm uses a closed enumeration tree (CET) to maintain a dynamically selected set of itemsets over a sliding window.

A formal definition of utility mining and theoretical model was proposed in Yao et al. (2004), namely MEU, dynamically selected set of itemsets over a sliding window. The Moment algorithm is pro-

Liu et al. (2005) proposed the Two-Phase algorithm for solving this specific problem. Although this algorithm is not efficient or scalable, it is by far the best of all the combinations is impractical, especially at the beginning stages, where the number of candidates approaches the number of items large or the utility threshold is low. Although this algorithm is not efficient or scalable, it is by far the best one to solve this specific problem.

Another algorithm named Two-Phase was proposed in Liu et al. (2005), which is based on the definition in Yao et al. (2004) and achieves the finding of high utility itemsets. The Two-Phase algorithm is used to prune down the number of candidates and can obtain the complete set of high utility itemsets. In the first phase, a model that applies the “transaction-weighted downward closure property” on the search space is used to expedite the identification of candidates. In the second phase, one extra database scan is performed to identify the high utility itemsets. However, this algorithm must rescan the whole database when new transactions are added from data streams. It incurs more cost on I/O and CPU time for finding high utility itemsets. Hence, the Two-Phase algorithm is focused on traditional databases and is not suited for mining data streams.

Although there existed numerous studies on high utility itemsets mining and data stream analysis as described above, there is no algorithm proposed for finding temporal high utility itemsets in data streams. This motivates our exploration of the issue of efficiently mining high utility itemsets in temporal databases like data streams in this research.

3. Proposed method: THUI-Mine

In this section, we present the THUI-Mine method. Section 3.1 describes the basic concept of THUI-Mine. Section 3.2 gives an example for mining temporal high utility itemsets. The procedure of the THUI-Mine algorithm is provided in Section 3.3.

3.1. Basic concept of THUI-Mine

The goal of utility mining is to discover all the itemsets whose utility values are beyond a user specified threshold in a transaction database. In Yao et al. (2004), the goal of utility mining is defined as the discovery of all high utility itemsets. An itemset X is a high utility itemset if \( u(X) > \varepsilon \), where \( X \subseteq I \) and \( \varepsilon \) is the minimum utility threshold; otherwise, it is a low utility itemset. For example, in 1, \( u(A,T_8) = 1 \times 3 = 3 \), \( u(C,T_8) = u(A,T_8) + u(C,T_8) = 1 \times 3 \times 1 = 4 \), and \( u(A,C,T_8) = 4 + 30 = 34 \). If \( \varepsilon = 120 \), \( \{A,C\} \) is a low utility itemset. However, if an item is a low utility item, its superset may be a high utility itemset. For example, consider \( u(D) = 84 < 120 \), D is a low utility item, but its superset \( \{B,D\} \) is a high utility itemset since \( u(\{B,D\}) = 160 > 120 \). Intuitively, all combinations of items should be processed so that it never loses any high utility itemset. However, this will incur intolerable cost on computation time and memory space. A set of terms leading to the formal definition of utility mining problem can be generally defined as follows by referring to (Yao et al., 2004):

- **I** = \( \{i_1,i_2,...,i_m\} \) is a set of items.
- **D** = \( \{T_1,T_2,...,T_n\} \) is a transaction database where each transaction \( T_i \in \text{Dis} \) a subset of **I**.
- **o(i_p,T_q)**, local transaction utility value, represents the quantity of item \( i_p \) in transaction \( T_q \). For example, \( o(A,T_3) = 12 \), as shown in Table 1 (panel a).
- **s(i_p)**, external utility, is the value associated with item \( i_p \) in the Utility Table. This value reflects the importance of an item, which is independent of transactions. For example, in Table 1 (panel b), the external utility of item \( A \), \( s(A) \), is 3.
- \( u(i_p,T_q) \), utility, the quantitative measure of utility for item \( i_p \) in transaction \( T_q \) is defined as \( o(i_p,T_q) \times s(i_p) \). For example, \( u(A,T_3) = 12 \times 3 \), in 1.
- \( u(X,T_q) \), utility of an itemset \( X \) in transaction \( T_q \), is defined as \( \sum_{i \in X} u(i_p,T_q) \), where \( X = \{i_1,i_2,...,i_m\} \) is a **k**-itemset \( T_q \in T_q \) and \( 1 \leq k \leq m \).
- **u(X)**, utility of an itemset \( X \), is defined as \( \sum_{T_q \in T_q \in T_q \in T_q} u(X,T_q) \).

Liu et al. (2005) proposed the Two-Phase algorithm for pruning candidate itemsets and simplifying the calculation of utility. First, the Phase I overestimates some low utility itemsets, but it never underestimates any itemsets. For example in 1, the transaction utility of transaction \( T_q \), denoted as \( tu(T_q) \), is the sum of the utilities of all items in \( T_q \): \( tu(T_q) = \sum_{i \in T_q} u(i_p,T_q) \). Moreover, the transaction-weighted utilization of an itemset \( X \), denoted as \( twu(X) \), is the sum of the transaction utilities of all the transactions containing \( X \): \( twu(X) = \sum_{X \subseteq D^*} tu(u_X) \). For example, \( twu(A) = tu(T_9) + tu(T_8) + tu(T_9) + tu(T_9) = 42 + 48 + 27 + 40 = 157 \) and \( twu(D,E) = tu(T_2) + tu(T_3) = 71 + 27 = 98 \). In fact, \( u(A) = u(A), \) \( u(A), \) \( u(A), \) \( tu(T_9) = 36 + 3 + 3 = 45 \) and \( u(D,E) = u(D,E), T_3 = u(D,E), T_9 = T_3 + u(D,E), T_9 = 11 + 23 = 34 \). Table 2 gives the transaction...
utility for each transaction in I. Second, one extra database scan is performed to filter the overestimated itemsets in phase II. For example, by observing that $twu(A) = 157 > 120$ and $u(A) = 45 < 120$, the item $\{A\}$ is pruned. Otherwise, it is a high utility itemset. Finally, all high utility itemsets are discovered in this way.

We illustrate the detail process of Two-Phase algorithm by the following example in db1,3 of I. Suppose the utility threshold is set as 120 with nine transactions in db1,3. In Phase I, the high transaction-weighted utilization 1-itemsets $\{A,B,C,D,E\}$ are generated since $twu(A) = tu(T_3) + tu(T_4) + tu(T_5) + tu(T_6) = 42 + 48 + 27 + 40 = 157 > 120$, $twu(B) = tu(T_2) + tu(T_4) + tu(T_6) + tu(T_7) + tu(T_9) = 71 + 52 + 48 + 105 + 40 = 361 > 120$, $twu(D) = tu(T_2) + tu(T_3) + tu(T_4) + tu(T_5) + tu(T_6) = 71 + 42 + 52 + 27 = 192 > 120$. $twu(E) = tu(T_1) + tu(T_2) + tu(T_3) + tu(T_6) + tu(T_7) + tu(T_8) = 31 + 71 + 22 + 48 + 105 + 27 = 304 > 120$. Then, 10 candidate 2-itemsets $\{AB, AC, ADAE, BC, BD, BE, CD, CE, DE\}$ are generated by the high transaction-weighted utilization 1-itemsets $\{A,B,C,D,E\}$ in the first database scan. In the same way, the high transaction-weighted utilization 2-itemset $\{BE\}$ are generated since $twu(AB) = tu(T_6) + tu(T_9) = 48 + 40 = 88 < 120$, $twu(AC) = tu(T_3) + tu(T_5) = 27 + 40 = 67 < 120$, $twu(AD) = tu(T_3) + tu(T_6) = 42 + 27 = 69 < 120$, $twu(BE) = tu(T_2) + tu(T_8) = 48 + 27 = 75 < 120$, $twu(BC) = tu(T_7) = 40 < 120$, $twu(BD) = tu(T_4) = 52 < 120$, $twu(BE) = tu(T_2) + tu(T_5) + tu(T_7) = 71 + 48 + 105 = 224 > 120$, $twu(CD) = tu(T_5) + tu(T_8) = 27 < 120$, $twu(CE) = tu(T_7) + tu(T_8) + tu(T_9) = 31 + 22 + 48 + 105 + 27 = 304 > 120$. After processing db1,3, the high transaction-weighted utilization itemsets in db1,3 are obtained as $\{A, B, C, D, E, BE\}$.

In Phase II, the high transaction-weighted utilization itemsets $\{A, B, C, D, E, BE\}$ is used to scan db1,3 to find high utility itemsets. The resulting high utility itemsets are $\{B\}$ and $\{BE\}$ since $u(A) = u(A)$, $T_3 + u(A)$, $T_6 + u(A)$, $T_9 + u(A)$, $T_9 = 45 < 120$, $u(B) = u(B)$, $T_2 + u(B)$, $T_4 + u(B)$, $T_6 + u(B)$, $T_7 + u(B)$, $T_9 = 220 < 120$, $u(C) = u(C)$, $T_1 + u(C)$, $T_3 + u(C)$, $T_5 + u(C)$, $T_8 = 66 < 120$, $u(D) = u(D)$, $T_2 + u(D)$, $T_3 + u(D)$, $T_4 + u(D)$, $T_9 = 72 < 120$, $u(E) = u(E)$, $T_1 + u(E)$, $T_3 + u(E)$, $T_5 + u(E)$, $T_6 + u(E)$, $T_8 = 35 < 120$ and $u(B, E) = u(B, E)$, $T_2 + u(B, E)$, $T_9 + u(B, E)$, $T_7 = 215 > 120$.

Our algorithm THUI-Mine is based on the principle of the Two-Phase algorithm (Liu et al., 2005), and we extend it with the sliding window filtering (SWF) technique and focus on utilizing incremental methods to reduce the candidate itemsets and execution time. In essence, by partitioning a transaction database into several partitions from data streams, algorithm THUI-Mine employs a filtering threshold in each partition to deal with the transaction-weighted utilization itemsets (TWUI) generated. The cumulative information in the prior phases is selectively carried over toward the generation of TWUI in the subsequent phases by THUI-Mine. In the processing of a partition, a progressive transaction-weighted utilization set of itemsets is generated by THUI-Mine. Explicitly, a progressive transaction-weighted utilization set of itemsets is composed of the following two types of TWUI: (1) the TWUI that were carried over from the previous progressive candidate set in the previous phases and remain as TWUI after the current partition is taken into consideration; (2) the TWUI that were not in the progressive candidate set in the previous phase but are newly selected after taking only the current data partition into account. As such, after the processing of a phase, algorithm THUI-Mine outputs a cumulative filter, denoted as CF, which consists of a progressive transaction-weighted utilization set of itemsets with their occurrence counts and the corresponding partial support required.

THUI-Mine is different from other existing methods like Lossy Counting (Manku and Motwani, 2002), which uses bucket boundaries and maximal possible error to update or delete the itemsets with frequency. The CF computes the occurrence counts of itemsets in memory and then deletes itemsets that do not satisfy utility threshold in every partial database. With these design considerations, algorithm THUI-Mine is shown to have very good performance for mining temporal high utility itemsets from data streams. In Section 3.2, we give an example for mining temporal high utility itemsets from a data stream. The details of THUI-Mine algorithm is described in Section 3.3.

3.2. An example for mining temporal high utility itemsets

The proposed THUI-Mine algorithm can be best understood by the illustrative transaction database in I and Fig. 2 where a scenario of generating high utility itemsets from data streams for mining temporal high utility itemsets is given. For real life applications, this illustrative transaction database can be mapped to the customer transactions in a supermarket. We set the utility threshold as 120 with nine transactions. Without loss of generality, the temporal mining problem can be decomposed into two procedures:

1. Pre-processing procedure: This procedure deals with mining on the original transaction database.
2. Incremental procedure: The procedure deals with the update of the high utility itemsets from data streams.

The pre-processing procedure is only utilized for the initial utility mining in the original database, e.g., db1,3. For mining high utility itemsets in db1,3, db1,4, db1,5 and

Table 2

<table>
<thead>
<tr>
<th>TID</th>
<th>Transaction utility</th>
<th>TID</th>
<th>Transaction utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>31</td>
<td>T7</td>
<td>105</td>
</tr>
<tr>
<td>T2</td>
<td>71</td>
<td>T8</td>
<td>27</td>
</tr>
<tr>
<td>T3</td>
<td>42</td>
<td>T9</td>
<td>40</td>
</tr>
<tr>
<td>T4</td>
<td>52</td>
<td>T10</td>
<td>62</td>
</tr>
<tr>
<td>T5</td>
<td>22</td>
<td>T11</td>
<td>42</td>
</tr>
<tr>
<td>T6</td>
<td>48</td>
<td>T12</td>
<td>21</td>
</tr>
</tbody>
</table>
so on, the incremental procedure is employed. Consider the database in 1. Assume that the original transaction database $db_{1,3}^1$ is segmented into three partitions, namely, $\{P_1, P_2, P_3\}$, in the pre-processing procedure. Each partition is scanned sequentially for the generation of candidate 2-itemsets in the first scan of the database $db_{1,3}^1$. Since there are three partitions, the utility threshold of each partition is $120/3 = 40$. Such a partial utility threshold is called the filtering threshold in this paper. After scanning the first segment of the three transactions, 1-itemsets $\{A, B, D, E\}$ are kept to generate 2-itemsets because $\text{twu}(A) = 42 > 40$, $\text{twu}(B) = 71 > 40$, $\text{twu}(C) = 31 < 40$, $\text{twu}(D) = 113 > 40$ and $\text{twu}(E) = 102 > 40$. Then, 2-itemsets $\{AB, ADAE, BD, BE, DE\}$ are generated by 1-itemsets $\{A, B, D, E\}$ in partition $P_1$ as shown in Fig. 2. In addition, each potential candidate itemsets $c \in C_2$ has two attributes: (1) $c$.start, which contains the identity of the starting partition when $c$ was added to $C_2$ and (2) transaction-weighted utility which is the sum of the transaction utilities of all the transactions containing $c$ since $c$ was added to $C_2$. Itemsets whose transaction-weighted utility are below the filtering threshold are removed. Then, as shown in Fig. 2, only $\{AD, BD, BE, DE\}$, marked by “@”, remain as temporal high transaction-weighted utilization 2-itemsets (TWU2I) whose information is then carried over to the next phase of processing. Similarly, after scanning partition $P_2$, the temporal high TWU2I are recorded.

From Fig. 2, it is noted that since there are also three transactions in $P_2$, the filtering threshold of those itemsets carried out from the previous phase is $40 + 40 = 80$ and that of newly identified candidate itemsets is 40. It can be seen from Fig. 2 that we have four temporal high TWU2I in $C_2$ after the processing of partition $P_3$, and two of them are carried from $P_1$ to $P_2$ and two of them are newly identified in $P_3$. Finally, partition $P_3$ is processed by algorithm THUI-Mine. The resulting temporal high TWU2I are $\{AB, AC, BC, BD, BE\}$ as shown in Fig. 2. Note that although itemset $\{AE\}$ appears in the previous phase $P_2$, it is removed from temporal high TWU2I once $P_3$ is taken into account since its transaction-weighted utility does not meet the filtering threshold then, i.e., $75 < 120$. However, we do have two new itemsets, i.e., $AC$ and $BC$, which join the $C_2$ as temporal high TWU2I. Consequently, we have five temporal high TWU2I generated by $THUI-Mine$, where two of them are carried from $P_1$ to $P_3$, one of them is carried from $P_2$ to $P_3$, and two of them are newly identified in $P_3$. Note that only five temporal high TWU2I are generated by $THUI-Mine$, while 10 candidate itemsets would be generated if Two-Phase algorithm were used as mentioned in Section 3.1. After processing $P_1$ to $P_3$, those temporal high TWU1 in $db_{1,3}^1$ are obtained as $\{AB, C, D, E, A, B, C, A, B, C, D, E\}$.

After generating temporal high TWU2I from the first scan of database $db_{1,3}^1$, we use a skill to reduce the number of database scan. In fact, it will take $k - 1$ database scan to generate $k$-candidate itemsets by using temporal high transaction-weighted utilization $(k - 1)$-itemsets directly. Instead, we use temporal high TWU2I to generate $C_k$ ($k = 3, 4, \ldots, n$), where $C_n$ is the candidate last itemset. It can be verified that temporal high TWU2I generated by $THUI-Mine$ can be used to generate the candidate 3-itemsets. Clearly, a $C_3$ can be generated from temporal high TWU2I. For example, the 3-candidate itemset $\{ABC\}$ is generated from temporal high TWU2I $\{AB, AC, BC\}$ in $db_{1,3}^1$. However, the temporal high TWU2I generated by $THUI-Mine$ is very close to the high utility itemsets. Similarly, all $C_k$ can be stored in main memory and we can find temporal high utility itemsets together when the second scan of the database $db_{1,3}^1$ is performed. Thus, only two scans of the original database $db_{1,3}^1$ are required in the pre-processing step. In this way, the number of database scan is reduced effectively. The resulting temporal high utility itemsets are $\{B\}$ and $\{BE\}$ since $u(B) = 220 > 120$ and $u(\{B, E\}) = 215 > 120$. 

![Fig. 2. Temporal high utility itemsets generated from data streams by THUI-Mine.](image-url)
One important merit of THUI-Mine lies in its incremental procedure. As depicted in Fig. 2, the mining of database will be moved from \(db^{1,3}\) to \(db^{2,4}\). Thus, some transactions like \(T_1, T_2\) and \(T_3\) are deleted from the mining database and other transactions like \(T_{10}, T_{11}\) and \(T_{12}\) are added. To illustrate it more clearly, this incremental step can also be divided into three sub-steps: (1) generating temporal high TWU2I in \(D^- = db^{1,3} - D\), (2) generating temporal high TWU2I in \(db^{2,4} = D^- + D^+\) and (3) scanning the database \(db^{2,4}\) only once for the generation of all temporal high utility itemsets. In the first sub-step, \(db^{1,3} - D = D^-\), we check the pruned partition \(P_i\) and reduce the value of transaction-weighted utility and set \(c.start = 2\) for those temporal TWU2I where \(c.start = 1\). It can be seen that itemset \(\{BD\}\) was removed. Next, in the second sub-step, we scan the incremental transactions in \(P_4\). The process in \(D^- + D^+ = db^{2,4}\) is similar to the operation of scanning partitions, e.g., \(P_2\), in the pre-processing step. The new itemset \(\{BD\}\) joins the temporal high TWU2I after the scan of \(P_4\). In the third sub-step, we use temporal high TWU2I

to generate \(C_k\) as mentioned above. Finally, those temporal high TWU2I in \(db^{2,4}\) are \(\{B, C, D, E, BC, BD, BE\}\). By scanning \(db^{2,4}\) only once, THUI-Mine obtains temporal high utility itemsets \(\{B, BC, BE\}\) in \(db^{2,4}\).

Table 3

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(db^i)</td>
<td>Partitioned database ((D)) from (P_i) to (P_j)</td>
</tr>
<tr>
<td>(s)</td>
<td>Utility threshold in one partition</td>
</tr>
<tr>
<td>(</td>
<td>P_i</td>
</tr>
<tr>
<td>(TUP_k(I))</td>
<td>Transactions in (P_k) that contain itemset (I) with transaction utility</td>
</tr>
<tr>
<td>(</td>
<td>db^{1,n}(I)</td>
</tr>
<tr>
<td>(C')</td>
<td>The progressive candidate sets of (db^i)</td>
</tr>
<tr>
<td>(Thtw^i)</td>
<td>The progressive temporal high transaction-weighted utilization 2-itemsets of (db^i)</td>
</tr>
<tr>
<td>(Thu^i)</td>
<td>The progressive temporal high utility itemsets of (db^i)</td>
</tr>
<tr>
<td>(D^-)</td>
<td>The deleted portion of an ongoing database</td>
</tr>
<tr>
<td>(D^+)</td>
<td>The unchanged portion of an ongoing database</td>
</tr>
<tr>
<td>(\Delta^+)</td>
<td>The added portion of an ongoing database</td>
</tr>
</tbody>
</table>

Fig. 3. Pre-processing procedure of THUI-Mine.
In contrast, Two-Phase algorithm has to scan the whole database like \( \text{db}^{2,4} \) and more candidate itemsets, i.e., \{BC, BD, BE, CD, CE, DE\}, will be generated whenever some transactions are deleted and other transactions are added. Then, Two-Phase algorithm needs one more database scan than THUI-Mine to obtain temporal high TWU2I. Finally, Two-Phase algorithm scans database again to produce temporal high utility itemsets. Hence, more database scans and candidate itemsets are incurred by Two-Phase algorithm in comparison with THUI-Mine.

### 3.3. THUI-Mine algorithm

For easier illustration, the meanings of various symbols used are given in Table 3. The pre-processing procedure and the incremental procedure of algorithm \( \text{THUI-Mine} \) are described in Sections 3.3.1 and 3.3.2, respectively.

#### 3.3.1. Pre-processing procedure of THUI-Mine

The pre-processing procedure of Algorithm \( \text{THUI-Mine} \) is shown in Fig. 3. Initially, the database \( \text{db}^{1,n} \) is partitioned into \( n \) partitions by executing the pre-processing

1. Original database = \( \text{db}^{m,n} \);
2. New database = \( \text{db}^{i,j} \);
3. Database removed \( \Delta^- = \Sigma_{k=m+1} P_k \);
4. Database added \( \Delta^+ = \Sigma_{k=n+1} P_k \);
5. \( D^- = \Sigma_{k=m+1} P_k \);
6. \( \text{db}^{i,j} = \text{db}^{m,n} - \Delta^- + \Delta^+ \);
7. Loading Thtwu \( ^{m,n} \) of \( \text{db}^{m,n} \) into CF where \( I \in \text{Thtwu}^{m,n} \);
8. begin for \( k = m \) to \( i - 1 \) // one scan of \( \Delta^- \)
9. \begin{itemize}
   
   \item begin for each 2-itemset \( I \subseteq P_k \)
   \item if (\( I \subseteq \text{CF} \) and \( I \) start \( \leq k \))
   \item \( \text{ltwu} = \text{ltwu} - \text{TUPk} (I) \);
   \item CF = CF - I;
   \item end
   \end{itemize}
10. end
11. begin for \( k = n + 1 \) to \( j \) // one scan of \( \Delta^+ \)
12. begin for each 2-itemset \( I \subseteq P_k \)
13. \begin{itemize}
   \item if (\( I \not\subseteq \text{CF} \))
   \item \( \text{ltwu} = \text{TUPk} (I) \);
   \item if (\( \text{ltwu} \geq s \times \text{P.count} \))
   \item CF = CF + I;
   \item \end{itemize}
14. end
15. end
16. select Thtwu \( ^{i,j} \) from \( I \) where \( I \subseteq \text{CF} \);
17. keep Thtwu \( ^{i,j} \) in main memory;
18. \( h = 2 \);
19. begin while (Thtwu \( ^{i,j} \) \( \neq \phi \)) // Database scan reduction
20. if (\( h = 2 \))
21. \( C_h^{i,j} = \text{Thtwu}^{i,j} \times \text{Thtwu}^{i,j} \);
22. else
23. \( h = h + 1 \);
24. end refresh Ltwu = 0 where \( I \subseteq \text{Thtwu}^{i,j} \);
25. begin for \( k = i \) to \( j \) // 2nd scan of \( \text{db}^{i,j} \)
26. \begin{itemize}
   \item begin for each itemset \( I \subseteq \text{Thtwu}^{i,j} \cup C_{h+1}^{i,j} \)
   \item \( I_u = I_u + \text{UPk} (I) \);
   \item end
   \end{itemize}
27. for each itemset \( I \subseteq \text{Thtwu}^{i,j} \cup C_{h+1}^{i,j} \)
28. \begin{itemize}
   \item if (\( I_u \geq s \times \text{P.count} \))
   \item \( \text{Thu}^{i,j} = \text{Thu}^{i,j} \cup I \);
   \item end
   \end{itemize}
29. return Thu \( ^{i,j} \);

Fig. 4. Incremental procedure of THUI-Mine.
procedure (in Step 2), and CF, the cumulative filter, is empty (in Step 3). Let \( T\text{hw}^{1,n} \) be the set of progressive temporal high TWU2I of \( d^{b,j} \). Algorithm \( \text{THUI-Mine} \) only records \( T\text{hw}^{1,n} \) which is generated by the pre-processing procedure to be used by the incremental procedure. From Step 4 to Step 16, the algorithm processes one partition at a time for all partitions. When partition \( P_i \) is processed, each potential candidate 2-itemset is read and saved to CF. The transaction-weight utility of an itemset \( I \) and its start-time utility value is \( I\text{.start} \) and \( I\text{.time} \). Initially, after some update activities, \( D_0 \) and new transactions \( D_t \) are added portions of an ongoing transaction database are added to the database \( D_0 \) in \( \text{Step} \ 6 \). Note that \( \Delta^- \subseteq \text{db}^{m,n}. \) Denoting the updated database as \( \text{db}^{b,j} \), note that \( \text{db}^{b,j} = \text{db}^{m,n} - \Delta^- + \Delta^+ \). We denote the unchanged transactions by \( D^+ = \text{db}^{m,n} - \Delta^- = \text{db}^{b,j} - \Delta^+ \). After loading \( \text{Thw}^{m,n} \) of \( \text{db}^{m,n} \) into CF where \( I \in \text{Thw}^{m,n} \), we start the first sub-step, i.e., generating temporal high TWU2I in \( D^- = \text{db}^{m,n} - \Delta^- \). This sub-step reverses the cumulative processing which is described in the pre-processing procedure. From Step 8 to Step 16, we prune the occurrences of an itemset \( I \) which appeared before partition \( P_i \), by deleting the value \( I\text{.twu} \) where \( I \in \text{CF} \) and \( I\text{.start} < i \). Next, from Step 17 to Step 39, similarly to the cumulative processing in Section 3.3.1, the second sub-step generates temporal high TWU2I in \( \text{db}^{b,j} = D^+ + \Delta^+ \) and employs the scan reduction technique to generate \( C_{b,j}^{1,n} \). Finally, to generate temporal high utility itemsets, i.e., \( \text{Thw}^{b,j} \), in the updated database, we scan \( \text{db}^{b,j} \) only once in the incremental procedure to find temporal high utility itemsets. Note that \( \text{Thw}^{b,j} \) is kept in main memory for the next generation of incremental mining.

### 4. Experimental evaluation

To evaluate the performance of \( \text{THUI-Mine} \), we conducted experiments using synthetic datasets generated via a randomized dataset generator provided by IBM Quest project (Agrawal and Srikant, 1995). However, the IBM Quest data generator only generates the quantity of 0 or 1 for each item in a transaction. In order to fit databases into the scenario of utility mining, we randomly generate the quantity of each item in each transaction, ranging from 1 to 5, as is similar to the model used in Liu et al. (2005). Utility tables are also synthetically created by assigning a utility value to each item randomly, ranging from 1 to 1000. Because it is observed from real world databases that most items are in the low profit range, we generate the utility values using a log normal distribution, as is similar to the model used in Liu et al. (2005). Fig. 5 shows the utility value distribution of 1000 items.

The simulation is implemented in C++ and conducted in a machine with 2.4 GHz CPU and 1 GB memory. For comparison with \( \text{THUI-Mine} \) algorithm, the two-Phase algorithm is extended with sliding window scenario. The extended Two-Phase algorithm scans the database according to the set time window and then performs the computation within the time window. This process is repeated over sliding time window for the database. The main performance metric used is execution time. We recorded the execution time of \( \text{THUI-Mine} \) in finding temporal high utility itemsets. The comparison on the number of generated itemsets for \( \text{THUI-Mine} \), Two-Phase and MEU is presented in Section 4.1. Section 4.2 shows the performance comparison of \( \text{THUI-Mine} \) and Two-Phase. The results of scale-up experiments are presented in Section 4.3. Section 4.4 shows the performance comparison of \( \text{THUI-Mine} \) and Two-Phase on another dense dataset.

#### 4.1. Evaluation on number of generated candidates

In this experiment, we compare the average number of candidates generated in the first database scan on the sliding windows and incremental transaction number d10K.
with different support values for THUI-Mine, Two-Phase (Liu et al., 2005) and MEU (Yao et al., 2004). Without loss of generality, we set \( |d| = |A^-| = |A^+| \) for simplicity. Thus, by denoting the original database as \( db^{[n]} \) and the new mining database as \( db^{[n+1]} \), we have \( |db^{[n+1]}| = |db^{[n]}| = |d| \), where \( A^- = db^{[n+1]} \) and \( A^+ = db^{[n+1]} \).

Tables 4 and 5 show the average number of candidates generated by THUI-Mine, Two-Phase and MEU on two datasets, respectively. The number of items is set at 1000, and the minimum utility threshold varies from 0.2% to 1%. The experimental results show that the number of candidate itemsets generated by THUI-Mine at the first database scan decreases dramatically as the threshold goes up. Especially, when the utility threshold is set as 1%, the number of candidate itemsets is 0 in database T10.I6.D100K.d10K where \( T \) denotes the average size of the transactions and \( I \) the average number of frequent itemsets. The default size of the sliding window is set as 30K. In fact, we also varied the size of sliding window and the experimental results show that THUI-Mine outperforms Two-Phase algorithm under different sliding windows sizes. Due to space limitation, we only show the representative results with the sliding window size set as 30K. However, the number of candidates generated by Two-Phase is still very large and that for MEU is always 499,500 because it needs to process all combinations of 1000 items. THUI-Mine generates far fewer candidates when compared to Two-Phase and MEU.

We obtain similar experimental results for different datasets. For example, only 118 candidate itemsets are generated by THUI-Mine, but 183,921 and 499,500 candidate itemsets are generated by Two-Phase and MEU, respectively, when the utility threshold is set as 1% in dataset T20.I6.D100K.d10K. In the case of dataset T20.I6.D100K.d10K, more candidates are generated, because the transaction is longer than that in T10.I6.D100K.d10K. In overall, our algorithm THUI-Mine always generates far fewer candidates compared to Two-Phase and MEU for various kinds of databases. Hence, THUI-Mine is verified to be very effective in pruning candidate itemsets to find temporal high utility itemsets.

### 4.2. Evaluation of execution efficiency

In this experiment, we compare only the relative performance of Two-phase and THUI-Mine since MEU spends much higher execution time and becomes incomparable. Figs. 6 and 7 show the execution times for the two algorithms on datasets T20.I6.D100K.d10K and T10.I6.D100K.d10K, respectively, as the minimum utility threshold is decreased from 1% to 0.2%. It is observed that when the minimum utility threshold is high, there are only a limited number of high utility itemsets produced. However, as the minimum utility threshold decreases, the performance difference becomes prominent in that THUI-Mine significantly outperforms Two-Phase. As shown in Figs. 6 and 7, THUI-Mine leads to prominent performance improvement under different sizes of transaction. Explicitly, THUI-Mine is significantly faster than Two-Phase and the margin grows as the minimum utility threshold decreases. For example, THUI-Mine is 10 times faster than Two-Phase when threshold is 0.2 for T20.I6.D100K.d10K. In overall, THUI-Minespends much less time than Two-Phase with higher stability in finding temporal high utility itemsets. This is because the Two-Phase algorithm produces more candidate itemsets and needs more database scans to find high utility itemsets than THUI-Mine. To measure the improvement on execution time for THUI-Mine compared to Two-Phase algorithm, we define the Improvement Ratio as follows:

![Fig. 6. Execution time for Two-Phase and THUI on T20.I6.D100K.d10K.](image-url)
From the data illustrated in Fig. 6, we see that the Improvement Ratio is about 85.6% with the threshold set as 0.2%. In Fig. 7, the average improvement is about 67% with minimum utility threshold varied from 0.2% to 1%. Obviously, THUI-Mine reduces substantially the time in finding high utility itemsets. Moreover, the high utility itemsets obtained by Two-Phase are not suitable for applications in data streams since Two-Phase needs more database scans and increased execution time in finding high utility itemsets. Hence, THUI-Mine meets the requirements of high efficiency in terms of execution time for data stream mining.

### 4.3. Scale-up on incremental mining

In this experiment, we investigate the effects of varying incremental transaction size on the execution time of mining results. To further understand the impact of $|d|$ on the relative performance of THUI-Mine and Two-Phase, we conduct scale-up experiments which are similar to those described in Lee et al. (2001) with minimum support thresholds being set as 0.2% and 0.4%, respectively. Fig. 8 shows the experimental results where the value in $y$-axis corresponds to the ratio of the execution time of THUI-Mine to that of Two-Phase under different values of $|d|$. It can be seen that the execution-time ratio remains stable with the growth of the incremental transaction number $|d|$ since the size of $|d|$ has little influence on the performance of THUI-Mine. Moreover, the execution time ratio of the scale-up experiments with minimum support thresholds varied from 0.6% to 1% remains constant at approximately 0.4%. This implies that the advantage of THUI-Mine over Two-Phase is stable and less execution time is taken as the amount of incremental portion increases. This result also indicates that THUI-Mine is useful for mining data streams with large transaction size.

### 4.4. Evaluation on dense data

Typically, the synthetic data sets are very sparse. For testing various kinds of databases, we evaluate another dense dataset, the gazelle data set as used in Zaki and Hsiao (2005). The gazelle data set comes from click-stream data from a dot-com company named Gazelle.com, a
legware and legcare retailer. This data set was used in the KDD-Cup 2000 competition and publicly available from www.ecn.purdue.edu/ KDDCUP. In order to fit databases into the scenario of utility mining, we also randomly generate the quantity of each item in each transaction, ranging from 1 to 5. The utility tables are also synthetically created by assigning a utility value to each item randomly, ranging from 1 to 1000.

Fig. 9 shows the execution time for the two algorithms as the minimum utility threshold is varied from 0.1% to 0.02%. It is observed that \textit{THUI-Mine} still spends less time than Two-Phase with higher stability for finding temporal high utility itemsets even under the dense data. This is because the Two-Phase algorithm produces more candidate itemsets and needs more database scans to find high utility itemsets than \textit{THUI-Mine}. Hence, this result also indicates that \textit{THUI-Mine} is effective for mining temporal high utility itemsets under both of sparse and dense datasets.

5. Conclusions

In this paper, we addressed the problem of discovering temporal high utility itemsets in data streams. Under the stream database situation, the memory is often limited and it is hard to store large itemsets in memory. We propose a new algorithm, namely \textit{THUI-Mine}, which can discover temporal high utility itemsets from data streams efficiently and effectively. The novel contribution of \textit{THUI-Mine} is that it can effectively identify the temporal high utility itemsets with less candidate itemsets such that the execution time can be reduced efficiently. In this way, the process of discovering the temporal high utility itemsets in data streams can be achieved effectively with less memory space and execution time. This meets the critical requirements of time and space efficiency for mining data streams.

The experimental results show that \textit{THUI-Mine} can discover the temporal high utility itemsets with higher performance by generating less candidate itemsets as compared to other algorithms under different experimental conditions, including both of sparse and dense datasets. Across the experiments, THUI-Mine is faster than Two-Phase by 2–10 times, and the performance gain becomes more significant as the minimum utility threshold decreases. For example, THUI-Mine is 10 times faster than Two-Phase when the threshold is 0.2 for dataset T20.I6.D100K.d10K. This performance enhancement comes mainly from the good feature of THUI-Mine in producing far fewer candidate itemsets. Moreover, the experimental results also show that \textit{THUI-Mine} is scalable with large databases. Therefore, it is indicated that the advantage of \textit{THUI-Mine} over Two-Phase is stable and less execution time is taken as the amount of incremental portion of databases increases. Hence, \textit{THUI-Mine} is promising for mining temporal high utility itemsets in data streams. For future work, we would extend the concepts proposed in this work to discover other interesting patterns in data streams like utility items with negative profit.

References

