JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 28, 193-209 (2012)

A Projection-Based Approach for Discovering High Average-Utility Itemsets

GUO-CHENG LAN[1], TZUNG-PEI HONG[*] AND VINCENT S. TSENG[1,2]
[1]*Department of Computer Science and Information Engineering*
[2]*Institute of Medical Informatics*
National Cheng Kung University
Tainan, 701 Taiwan
E-mail: rrfoheiay@gmail.com; tsengsm@mail.ncku.edu.tw
[*]*Department of Computer Science and Information Engineering*
National University of Kaohsiung
Kaohsiung, 811 Taiwan
[*]*Department of Computer Science and Engineering*
National Sun Yat-sen University
Kaohsiung, 804 Taiwan
E-mail: tphong@nuk.edu.tw

Utility mining has recently been an important issue due to its wide applications. An itemset in traditional utility mining considers individual profits and quantities of items in transactions regardless of its length. The average-utility measure, which is the total utility of an itemset divided by its number of items within it, was then proposed to reveal a better utility effect than the original utility measure. A mining algorithm was also proposed to find high average-utility itemsets from a transaction database. However, the previous mining approach was based on the principle of level-wise processing to find high average-utility itemsets from a database. In this paper, we thus propose an efficient average-utility mining approach which adopts a projection technique and an indexing mechanism to speed up the execution and reduce the memory requirement in the mining process. The proposed approach can project relevant sub-databases for mining, thus avoiding some unnecessary checking. In addition, a pruning strategy is also designed to reduce the number of unpromising itemsets in mining. Finally, the experimental results on synthetic datasets and two real datasets show the superior performance of the proposed approach.

Keywords: data mining, association rule mining, utility mining, average utility, indexing mechanism

1. INTRODUCTION

In knowledge discovery, the main purpose of data mining is to extract desired patterns or rules from databases. Many related techniques in data mining have also widely been applied to various practical applications, such as supermarket promotions, biomedical data applications, multimedia data applications, mobile data applications, and so forth. Of these, association-rule mining [3] is one of the most important issues in data mining since it, which can be used to find frequent itemsets from databases, is a fundamental work for knowledge discovery. However, an itemset in traditional association-rule mining only consider the occurrence of items. Thus, it does not reflect any other factors, such as price, cost or profit. In addition, the same significance in association-rule mining is also assumed for

Received March 1, 2011; revised August 7, 2011; accepted August 27, 2011.
Communicated by I-Chen Wu.

all the items in a database. The actual significance of an itemset cannot then easily be recognized. For example, assume there is an itemset like "{DVD players, LCD TVs}". Although it may not be high frequency in comparison with sale of breads, it may contribute a large portion to the overall profit in a database due to LCD TVs. In order to deal with this, a new discussed issue, namely utility mining, was thus proposed by Chan *et al.*, in 2003 [4]. In this issue, they considered both the individual profits and quantities of products (items) in a database, and used both the factors to measure the actual utility for each itemset in the database. Thus the high utility itemsets, which had their utility values larger than or equal to a predefined threshold, were found as the desired.

In utility mining, however, the actual utility value of an itemset is the summation of the utility values of all items in it all the transactions including it regardless of its length. Thus, the actual utility values of itemsets are increased along with the increase of their lengths. However, itemsets with different lengths by using the same utility threshold in utility mining together are measured whether they are really high utility itemsets or not. Accordingly, it is not fair for the condition above. Hong *et al.* then proposed an average-utility measure to deal with the problem above [9]. The measure also considered the number of all items in an itemset in addition to profits and quantities of items used in the original utility measure. The actual average-utility of an itemset could be defined as the summation of all utility values of all items in it in transactions appearing it divided by its number of items within it. Also, since the downward-closure property used in utility mining was not directly applied to the average-utility mining, they designed an average-utility upper bound [9], which the average-utility upper bound of an itemset was defined as the summation of the maximal utility among utility values of items in each transaction appearing it. However, although this upper bound was applied to their proposed two-phase average-utility (*TPAU*) approach, *TPAU* adopted a level-wise technique to achieve the mining task. Thus, it may have to spend a good deal of time cost generating a large number of candidates and counting their upper bounds.

The concept of projection techniques is borrowed from the query language of database management systems. In database queries, the records satisfying a query condition are extracted from the database as the temporary results, and this idea has also been applied to the field of data mining. For example, Han *et al.* used the projection technique to sequential pattern mining [14], which was extended from association-rule mining with additional consideration of the time factor. In their approach, each candidate sequence could be thought of as a query condition, and the tuples in a database which contained these sequences were projected.

In this paper, we thus propose an efficient projection-based approach for mining high average-utility itemsets (*PBAU*) and design a new indexing structure to implement it. In particular, an effective pruning strategy is also adopted to avoid generating unpromising candidates in the mining. In addition, both the average-utility upper bound and the actual average-utility values of itemsets in our proposed approach are calculated at the same time. Both the memory usage and the efficiency can thus be raised for mining. The experimental results also show that the memory requirement is less than that needed by the *TPAU* algorithm [9] when working with both of synthetic datasets and two real datasets, *BMSPOS* and *chess* [6], and also *PBAU* executes faster than *TPAU*.

The remaining parts of this paper are organized as follows. Some related works are reviewed in section 2. The problem to be solved and the proposed mining algorithm with an indexing structure and a pruning strategy for finding high average-utility itemsets from a set of data are stated in section 3. An example is given to illustrate the execution process of the proposed algorithm in section 4. The experimental evaluation is shown in section 5. Conclusions and future works are finally given in section 6.

2. RELATED WORK

As mentioned in the part of Introduction, the main propose of data mining is to extract desired patterns or rules from a set of data. One common type of data mining is to derive association rules from transaction data, such that the presence of certain items in a transaction will imply the presence of some other items. In order to achieve this purpose, Agrawal *et al.* proposed several mining algorithms based on the concept of large itemsets to find association rules from transaction data [1-3]. In association-rule mining, only binary itemsets are thus considered. In real-world applications, however, products bought in transactions contain both profits and quantities. In particular, some high-profit products may occur with low frequency in a transaction database. For example, both jewel and diamond have high utility values but may not be frequent product combinations when compared to food and drink in a transaction database. Thus high-profit but low-frequency itemsets may not be found by the traditional association-rule mining approaches. To deal with this, Chan *et al.* proposed utility mining to discover high utility itemsets from a database [4]. In this study [4], a utility itemset considers not only the quantities of the items in transactions, but also their profits. Formally, local transaction utility and external utility are used to measure the utility of an item. The local transaction utility of an item is directly obtained from the information stored in a transaction database, like the quantity of the item sold in a transaction. The external utility of an item, like its profit, is given by users, and can be represented by a utility table or a utility function. By using a transaction dataset and a utility table together, the discovered itemset is able to better match a user's expectations than if found by considering only the transaction dataset itself.

However, association-rule mining keeps the downward-closure property, but utility mining does not, and thus the latter is much harder than the former. Liu *et al.* proposed a two-phase utility mining algorithm to discover high utility itemsets from a database by adopting the downward-closure property [13], and this approach was named as the transaction-weighted utilization (abbreviated as *TWU*) model. It used the summation of utility values of all the items in a transaction as the upper bound of any itemset in that transaction to keep the downward-closure property. Based on the model [13], several other studies about utility mining have also been published [5, 7, 10, 12, 15, 17-20].

In utility mining, however, the actual utility of an itemset is increased along with the increase of its number of items within it. It is thus not fair for using the same utility threshold to judge whether an itemest with different length is a high utility itemset or not. To deal with this, Hong *et al.* proposed a new measure (called average-utility measure), which considered the effect of itemset length in addition to profits and quantities of items, to reveal a better utility effect for itemsets [9]. The actual average-utility of an itemset could be defined as the summation of all utility values of all items in it in transactions appearing it divided by its number of items within it. Then, the average-utility measure could effectively be reduced the search space of finding high utility itemsets required by using original utility one. In addition, Hong *et al.* designed an average-utility upper bound, which the average-utility upper bound of an itemset was defined as the summation of the maximal utility among utility values of items in each transaction appearing it. The upper bound in their proposed algorithm (two-phase average-utility mining algorithm, *TPAU*) was then applied to overestimate all possible high average-utility itemsets in a database [9]. Its process could be divided into two phases. In the first phase, the average-utility upper bound was adopted to all high average-utility upper bound itemsets (*HAUUB*) level by level, which their average-utility upper bound values satisfy the minimum average-utility threshold. In the second phase, an additional database scan is required to find their actual average-utility values in the database. Finally, the average-utility itemsets which their actual average-utility values were larger than or equal to the minimum threshold, were found from the set of *HAUUB*. However, since the *TPAU* algorithm has to generate a large number of candidates and to count their average-utility upper bounds, it has to spend a good deal of time cost dealing with them.

3. THE PROPOSED MINING ALGORITHM

In this paper, the problem is to find the itemsets with their average-utility values larger than or equal to a predefined minimum average-utility threshold. Two strategies, including the indexing mechanism and the pruning strategy, in our algorithm are utilized to help its execution efficiency. The indexing mechanism is then described below first.

3.1 The Indexing Structure

A new index structure in our proposed algorithm is designed to raise the efficiency of average-utility mining. With the structure, the transactions to be handled do not need to be actually projected into a sub-database for mining. The structure is an index table with two fields: transaction identifier and last item position. Each itemset is associated with a table which will be dynamically generated whenever necessary. The transaction identifier field keeps the identifiers of the transactions in which the itemset to be processed appears. The last item position field keeps the position of the last item in the itemset in the corresponding transaction.

Besides, an index table of an itemset with n items can be easily derived from the index table of its preceding sub-itemset with $n-1$ items (called prefix itemset). For example, the prefix itemset of the itemset $\{ABC\}$ is $\{AB\}$. The index table of the itemset $\{ABC\}$ can thus be derived by checking the transactions indexed by the index table of its prefix itemset $\{AB\}$. The index table can imitate the projection technique to really project the transaction data required by the prefix currently being processed. An example is then given below to illustrate the building and usage of the index table.

Assume the three transactions to be processed are $Trans_1$: $\{1A, 2C, 1D\}$, $Trans_2$: $\{1B, 25C\}$ and $Trans_3$: $\{1B, 12C\}$, where numbers represent quantities and symbols represent items, as shown in Table 1. There are four items in the transactions, denoted as A to D, and their profits are 3, 10, 1 and 6, respectively. Also, a minimum average-utility threshold is set at 20. The maximum utility values of the three transactions in Table 1 are 6, 25 and 12.

First, the average-utility upper bound and the actual average-utility values of each item are found. Take the item B as an example. It appears in the two transactions, $Trans_2$

Table 1. The transaction data used in the example.

TID	Items
$Trans_1$	1*A*, 2*C*, 1*D*
$Trans_2$	1*B*, 25*C*
$Trans_3$	1*B*, 12*C*

Table 2. The index table for the 1-itemset {*B*} in the set of $HAUUB_1$.

Transaction Identifier	Last Item Position
2	1
3	1

and $Trans_3$, and their maximum utility values are 25 and 12. In addition, its quantity values in the two transactions are all 1, and its profit value and itemset length are 10 and 1. The average-utility upper bound value and the actual average-utility value of the item *B* can be then calculated as 25 + 12 (= 37) and ((1 * 10) + (1 * 10))/1 (= 20), respectively. By the same way, the average-utility upper bounds of the four items, *A*, *B*, *C*, and *D*, are 6, 37, 43 and 6, respectively, and their actual average-utility values are 3, 20, 39 and 6.

Next, the high average-utility upper bound 1-itemsets ($HAUUB_1$) and high average-utility 1-itemsets (HAU_1) are found. In this example, only the two 1-itemsets, {*B*} and {*C*}, are put in the set of $HAUUB_1$ since their average-utility upper bound values satisfy the minimum average-utility threshold (= 20). Also, the actual average-utility values of the two itemsets in the set of $HAUUB_1$ are larger than the minimum average-utility threshold. They are then put in the set of HAU_1 as well. Next, the 1-itemset {*B*} in the set of $HAUUB_1$ is first processed in its alphabetical order, and the index table for it is then built, as shown in Table 2.

In Table 2, the content of the first tuple is (2, 1), where '2' means that the itemset {*B*} appears in the second transaction and '1' means that the last item *B* from the itemset {*B*} is the first item in the second transaction {1*B*, 25*C*}. By the index table of {*B*}, each transaction associated with {*B*} can be efficiently found, and then the 2-itemsets with {*B*} as their prefix itemset are quickly found. In this example, only one 2-itemset {*BC*} is produced in the two transactions indexed by the index table of {*B*}, and its average-utility upper-bound and actual average-utility values in the indexed transactions are then found as 37 and 28.5, respectively. For the itemset {*BC*}, since its average-utility upper bound and actual average-utility values satisfy the minimum average-utility threshold (= 20), it is put in the set of $HAUUB_2$ with {*B*} and in the set of HAU_2 with {*B*}. The next itemset to be processed is {*BC*} in the set of $HAUUB_2$ with {*B*} after the process of prefix {*B*} is done. Then, the index table of {*BC*} can be easily derived from only the transactions appearing in the index table of {*B*}, as shown in Table 3.

Table 3. The index table for the 2-itemset {*BC*} with the prefix item {*B*}.

Transaction Identifier	Last Item Position
2	2
3	2

However, since no 3-itemsets can be produced in the transactions linked by the index table of {*BC*}, the process of finding high average-utility 3-itemsets with the prefix {*BC*} is terminated. In addition, no other 2-itemsets exist in the set of $HAUUB_2$ with the prefix {*B*}, the next prefix to be processed is thus {*C*}. The above process is recursively performed until all the 1-itemsets in the set of $HAUUB_1$ have been done. As described in this example, the indexing structure in the proposed algorithm can speed up the execution efficiency in finding high average-utility itemsets (*HAU*) from a database.

3.2 The Pruning Strategy

In this study, a simple pruning approach is designed to reduce the number of unpromising candidates for mining. Its main concept is that all the items appearing in the currently processed set of high average-utility upper bound itemsets of *r* items ($HAUUB_r$) are gathered in the recursive mining process. The additional (*r* + 1)th item of each generated (*r* + 1)-itemset will be checked for whether it is appeared in the set of gathered items or not. If it is, the generated (*r* + 1)-itemset will be put in the set of (*r* + 1)-itemsets; otherwise, it is pruned. An example is given below to illustrate the pruning effectiveness of unpromising candidate itemsets in the mining process.

Assume there is a transaction {1*A*, 3*B*, 2*C*, 1*D*, 3*E*, 2*F*}, where numbers represent quantities and symbols represent items. Also assume the current prefix sub-itemset is the 1-itemset {*A*}, and the set of $HAUUB_2$ includes the two *2*-itemsets, {*AC*} and {*AE*} after the process of finding high average-utility itemsets with {*A*} as their prefix. Then, since mining procedure of prefix {*A*} has been done, the next prefix in the set of *HAUUB* with the prefix {*A*} in their alphabetical order is {*AC*} as described in section 3.1. All the items appearing in the current set of $HAUUB_2$ with {*A*} as their prefix are then gathered, which contain *A*, *C* and *E*. The 3-itemset {*ACE*} with {*AC*} as its prefix must thus include item {*E*} since only {*E*} in addition to {*A*} and {*C*} appears in the set of the gathered items. By using the pruning strategy, other 3-itemsets {*ACD*} and {*ACF*} in the transaction cannot then be produced. As seen in this example, the number of unpromising candidates can be reduced. The pruning strategy can thus speed up the execution efficiency of our proposed algorithm.

3.3 The Projection-Based Average-Utility Mining Algorithm with Index Tables

Input: A set of items, each with a profit value; a transaction database, in which each transaction includes a subset of items with quantities; the minimum average-utility threshold λ.
Output: A final set of high average-utility itemsets (*HAU*).
Step 1: For each *y*th transaction $Trans_y$ in *D*, do the following substeps,

1.1 Calculate the utility value u_{yj} of each *j*th item i_{yj} in $Trans_y$ as:

$$u_{yj} = s_{yj} * q_{yj},$$

where s_{yj} is the profit of item i_{yj} and q_{yj} is the quantity of i_{yj}.

1.2 Find the maximal utility value m_{uy} in $Tran_{sy}$ as:

$$mu_y = \max\{u_{y1}, u_{y2}, \ldots, u_{yj}\},$$

where u_{yj} is the utility of the *j*th item i_{yj} in $Trans_y$.

Step 2: For each item *I* in *D*, do the following substeps,

2.1 Calculate the average-utility upper bound ub_I of each item *I* as the summation of the maximum utility values of the transactions which include the item *I*. That is:

$$ub_I = \sum_{I \in Trans_y} mu_y.$$

2.2 Calculate the actual average-utility au_I of each item I as the summation of the utility values of the transactions which include the item I against the number of items in an itemset. That is:

$$au_I = \frac{\sum_{I \subset Trans_y} u_{yI}}{|I|},$$

where u_{yI} is the utility of the item I in $Trans_y$ and $|I|$ is the number of items in I.

Step 3: Do the following substeps for each item I in the set of candidate 1-itemsets.

3.1 If the average-utility upper bound of I is larger than or equal to the minimum average-utility threshold, put it in the set of high average-utility upper bound 1-itemsets, $HAUUB_1$.

3.2 If the actual average-utility of I satisfies the minimum average-utility threshold, put it in the set of high average-utility 1-itemsets, HAU_1.

Step 4: Process each itemset I in the set of $HAUUB_1$ in alphabetical order by the following substeps.

4.1 Build the index table IT_I of I, in which each tuple consists of two fields: transaction identifier and the last item position of I in the corresponding transaction.

4.2 Set $r = 1$, where r represents the number of items in the processed itemset.

4.3 Find all the high average-utility itemsets with I as their prefix item by the *Finding-HAU*(I, IT_I, r) procedure. Let the set of returned high average-utility itemsets be HAU_I.

Step 5: Output the set of high average-utility itemsets in all the HAU_I.

After step 5, all the high average-utility itemsets are found. The *Finding-HAU*(x, IT_x, r) procedure finds all the high average-utility itemsets with the r-itemset x as their prefix and is stated as follows.

The *Finding-HAU*(*x*, *IT*$_x$, *r*) procedure

Input: A prefix r-itemset x and its corresponding index table IT_x.

Output: The high average-utility itemsets with x as its prefix sub-itemset.

PStep 1: Initialize the temporary itemset table as an empty table, in which each tuple consists of three fields: itemset, average-utility upper bound (*ub*), and actual average-utility (*au*).

PStep 2: Acquire the items appearing in the set of $HAUUB_r$ and denote them as S_r.

PStep 3: Do the following substeps for each tuple in the index table of the given prefix r-itemset x.

3.1 Get the transaction identifier t and the last item position p of x from the tuple.

3.2 Link to the transaction t through the identifier.

3.3 Get each item I located after p in t.

3.4 Check whether I appears in S_r or not. If it does, then generate the $(r + 1)$-itemset composed of the prefix r-itemset x and I, put it in the temporary itemset table, and add its maximum utility and actual average-utility in the corresponding fields in the itemset table; otherwise, omit the item.

PStep 4: Do the following substeps for each $(r + 1)$-itemset s in the itemset table.

4.1 If the average-utility upper bound of s is larger than or equal to the minimum average-utility threshold, put it in the set of $HAUUB_{r+1}$.

4.2 If the actual average-utility of s is larger than or equal to minimum average-utility threshold, put it in the set of HAU_{r+1}.

PStep 5: Do the following substeps for each high average-utility upper bound itemset s in the set of $HAUUB_{r+1}$ with x as their prefix itemset in alphabetical order.

5.1 Build the index table IT_s for itemset s from the index table IT_x for x by keeping the tuples with s and its positions in the corresponding transactions.

5.2 Recursively find all the high utility itemsets with s as their prefix items by invoking the *Finding-HAU*(s, IT_s, $r + 1$) procedure. Let the set of returned high utility itemsets be HAU_s.

PStep 6: Return the set of high average-utility itemsets in all the HAU_x.

4. AN EXAMPLE

In this section, a simple example is given to show how the proposed algorithm can easily be used to find high average-utility itemsets from a set of transactions. Assume there are ten transactions shown in Table 4 for mining. Each transaction consists of two features, transaction identification (*TID*) and items purchased. There are six items in the transactions, respectively denoted as *A* to *F*. The value attached to each item in the corresponding slot is the quantity sold in the transactions. Also, assume that the profit values for the six items in this example are defined as 3, 10, 1, 6, 5, and 2, respectively.

Moreover, the minimum average-utility threshold is set as 25. The proposed mining algorithm proceeds as follows in order to find high average-utility itemsets (*HAU*) from the transaction data in Table 4.

Table 4. The set of ten transaction data in this example.

TID	*A*	*B*	*C*	*D*	*E*	*F*
$Trans_1$	1	0	2	1	1	1
$Trans_2$	0	1	25	0	0	0
$Trans_3$	0	0	0	0	2	1
$Trans_4$	0	1	12	0	0	0
$Trans_5$	2	0	8	0	2	0
$Trans_6$	0	0	4	1	0	1
$Trans_7$	0	0	2	1	0	0
$Trans_8$	3	2	0	0	2	3
$Trans_9$	2	0	0	1	0	0
$Trans_{10}$	0	0	4	0	2	0

Step 1: Maximum utility *mu* value of each transaction in the transaction data in Table 2. Take the second transaction ($Trans_2$) as an example. $Trans_2$ includes the two items, *B* and *C*, and their quantity values are 1 and 25. Also, their profit values are 10 and

1. Their utility values can be then calculated as 1×10 (= 10) and 25×1 (= 25), respectively, and thus the maximum utility for $Trans_2$ is 25. All the other transactions in Table 2 can be similarly processed, and thus the maximum utility values for the ten transactions are found as 6, 25, 10, 12, 10, 6, 6, 20, 6, and 10, respectively.

Step 2: After this step is done, the results for the average-utility upper bound (*ub*) and actual average-utility (*au*) values of all the items in Table 4 are shown in Table 5.

Table 5. The average-utility upper bound and the average-utility values for the items.

1-itemset	*ub*	*au*
{*A*}	42	24
{*B*}	57	40
{*C*}	75	57
{*D*}	24	24
{*E*}	56	45
{*F*}	42	12

Step 3: The five 1-itemsets in Table 5, {*A*}, {*B*}, {*C*}, {*E*} and {*F*}, are put in the set of $HAUUB_1$ since their average-utility upper-bounds satisfy the minimum threshold (= 25). In addition, actual average-utility values of the other four items among them in addition to {*F*} satisfy the threshold (= 25), and thus they are put in the set of HAU_1.

Step 4: The five 1-itemsets in the set of $HAUUB_1$ are sequentially processed in their alphabetical order, and 1-itemset {*A*} is thus processed first, and its index table includes four tuples, (1, 1), (5, 1), (8, 1), and (9, 1), respectively. The variable *r* is then set at 1, where *r* represents the number of items in the itemsets to be processed. All high average-utility itemsets with the prefix itemset {*A*} are then found by the *Finding-HAU*(x, IT_x, r) procedure with the parameters $x = \{A\}$, $IT_x = IT_{\{A\}}$ and $r = 1$. The details of the *Finding-HAU*(x, IT_x, r) procedure executed on this example are described below.

PStep 1: The temporary itemset table is initialized as an empty table, in which each tuple consists of three fields: itemset, average-utility upper bound (*ub*) of the itemset, and actual average-utility (*au*) of the itemset.

PStep 2: The items appearing in the current set of $HAUUB_r$ are acquired, and denoted as S_r. For now, $r = 1$, and only the five 1-itemsets, {*A*}, {*B*}, {*C*}, {*E*} and {*F*} are kept in the set of $HAUUB_1$. They are then denoted as S_1.

PStep 3: The following substeps are done for each tuple in the index table of the given prefix 1-itemset {*A*}. The index table of {*A*} includes the four tuples, (1, 1), (5, 1), (8, 1), and (9, 1), respectively. For the first tuple is (1, 1), it represents that {*A*} appears in the first transaction and the last item position of the itemset {*A*} in the transaction is 1. The postfix items include *C*, *D*, *E*, and *F*, and then these items are sequentially checked to see whether they exist in S_1 or not. Since only the three items *C*, *E* and *F* exist in S_1, the 2-itemsets with the prefix {*A*} are generated from $Trans_1$ as {*AC*}, {*AE*} and {*AF*}.

Table 6. The temporary itemset table with the prefix item {*A*} in this example.

2-itemset	*ub*	*au*
{*AB*}	20	14.5
{*AC*}	16	7
{*AE*}	36	21.5
{*AF*}	26	10

Next, since the maximum utility value of $Trans_1$ is 6, and the actual average-utility values of the three 2-itemset {*AC*}, {*AE*} and {*AF*} in $Trans_1$ are 2.5, 4 and 2.5, these values are added in the corresponding field values of the itemset table. All the other tuples in the index table of {*A*} can be processed in the same way. The 2-itemset table with the prefix {*A*} is shown in Table 6.

PStep 4: In Table 6, the two 2-itemsets {*AE*} and {*AF*} are high average-utility upper bound 2-itemsets and are put in current $HAUUB_2$. Also, no high average-utility 2-itemsets with the prefix {*A*} are found in the example.

PStep 5: The 2-itemset {*AE*} in current set of $HAUUB_2$ with prefix itemset {*A*} is first processed in their alphabetical order. The index table of {*AE*} can be directly built from the index table of {*A*}. Thus, the index table $IT_{\{AE\}}$ of {*AE*} only includes the three tuples (1, 4), (5, 3), and (8, 3), respectively. Note that the process of checking can be terminated early if the item to be checked does not appear in a transaction but another item with a later prefix order appears. For the above example, if the item *E* does not appear in the first transaction, then the checking can be early terminated if an item with a prefix order after *E* appears (like *F* in the example) by their alphabetical order.

After the index table for {*AE*} has been built, the high average-utility itemsets with the prefix itemset {*AE*} are then found by recursively invoking the *Finding-HAU*(x, IT_x, r) procedure with the parameters x = {*AE*}, $IT_x = IT_{\{AE\}}$ and $r = 2$. According to the index table of {*AE*}, only the 3-itemset {*AEF*} with the prefix itemset {*AE*} is generated and put in the temporary itemset table. Its average-utility upper bound utility and the actual average-utility in the temporary table are 26 and 11.67, respectively. Since only its average-utility upper bound satisfies the minimum average-utility threshold (= 25), {*AEF*} is put in the current $HAUUB_3$. All the high average-utility itemsets with the prefix itemset {*AEF*} are then found by invoking the *Finding-HAU*(x, IT_x, r) procedure again, with x = {*AEF*}, $IT_x = IT_{\{AEF\}}$ and $r = 3$. In this example, no 4-itemsets with the prefix itemset {*AEF*} are existed in the index table of {*AEF*}. The process of finding high average-utility itemsets with the prefix {*AEF*} is thus stopped. The next itemset to be processed is then {*AF*} in $HAUUB_2$. The above process is recursively executed until all the 1-itemsets in $HAUUB_1$ are processed. Only the two itemsets, {*C*} and {*BC*}, are high average-utility itemsets, and their actual average-utility values are 57 and 28.5.

Step 5: In this example, the two itemsets in the set of *HAU* are output to users.

5. EXPERIMENTAL EVALUATION

A series of experiments were conducted to compare the performance of the proposed projection-based average-utility approach (*PBAU*) and the two-phase average-utility approach (*TPAU*) [9] for different parameters. They were implemented in J2SDK 1.5.0 and executed on a PC with 3.0 GHz CPU and 1 GB memory.

5.1 Experimental Datasets

To show the practical performance, two real public datasets, *BMS-POS* and *chess* [6], were used in the experiments. These datasets were used in the KDDCUP 2000 competition. The *BMS-POS* collected several years of point-of-sale data from a large electronics retailer, making it very suitable as a test dataset in the experiments. The details of this dataset were follows. Each transaction consisted of all the product categories purchased by a customer at one time. There were 515,597 transactions in this dataset and the total number of different items was 1,657. In addition, the maximal length of a transaction was 164 and the average length of the transactions was 6.5. On the other hand, a dense dataset *chess* [6] was used to evaluate the practical effectiveness of the proposed pruning strategy in terms of candidate itemsets and execution efficiency. There were 3,196 records in the real dataset *chess*, the total number of different items was 75, and the average length of the records was 37.

Besides, a public *IBM* data generator was also used in our experiments [8] to generate required datasets. The parameters, *T*, *I*, *N*, and *D* represented the average length of items per transaction, the average length of maximal potentially frequent itemsets, the total number of different items, and the total number of transactions, respectively. In this work, we also developed a simulation model, which was similar to that used in Liu *et al.* [13], to generate the quantities of the items in the transactions. Each quantity ranged among 1 to 5 according to the described way in [13]. Moreover, for each dataset generated, a corresponding utility table was also produced in which a profit value in the range from 0.01 to 10.00 was randomly assigned to an item. Fig. 1 showed the profit-value distribution of all the items generated by the simulation model in the utility table.

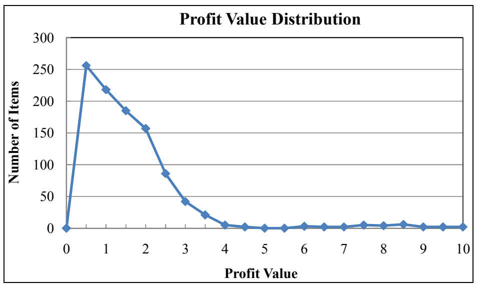

Fig. 1. The profit-value distribution in the real dataset.

5.2 Evaluation of the Pruning Strategy

Experiments were first made on the dataset, *chess*, to evaluate the effectiveness of our

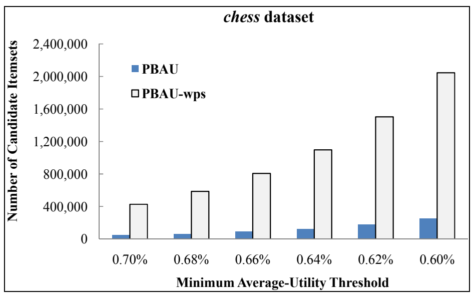

Fig. 2. The difference in the numbers of candidate itemsets along with different thresholds.

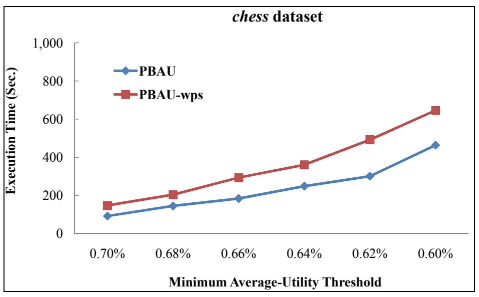

Fig. 3. The execution time of the two algorithms along with different thresholds.

proposed pruning strategy. Here the proposed *PBAU* algorithm without the pruning strategy was abbreviated as *PBAU-wps*. Figs. 2 and 3 showed the difference between the numbers of candidate itemsets, and execution efficiency required by the two algorithms, *PBAU* and *PBAU-wps*, for the *chess* dataset with the thresholds varying from 0.70% to 0.60%, respectively.

In these figures, it could be observed that the execution time and the numbers of candidates required by the *PBAU* algorithm with the pruning strategy were obviously less than those required by the *PBAU-wps* algorithm. The main reason for this was that the proposed strategy could effectively avoid generating unnecessary itemsets for mining. As a result, *PBAU* was more likely to prune candidate itemsets when compared to *PBAU-wps*. Also, the execution efficiency of *PBAU* was better than that of *PBAU-wps*.

5.3 Evaluation on Synthetic Datasets

Experiments were then made to evaluate the performance of the two algorithms. Figs. 4 and 5 showed the difference in the number of candidates generated and the memory usage required by the algorithms on the T10I4N4K datasets with different data sizes varying from 100K to 500K when the minimum average-utility threshold (*min_autil*) was set at 0.1%. Note that *HAU* represented the number of high average-utility itemsets. In addition, Fig. 6 showed the execution time on the T10I4N4KD200K dataset for different thresholds varying from 0.10% to 0.02%.

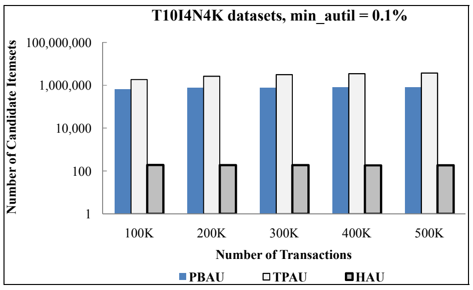

Fig. 4. The difference in the numbers of candidate itemsets along with different minimum average-utility thresholds.

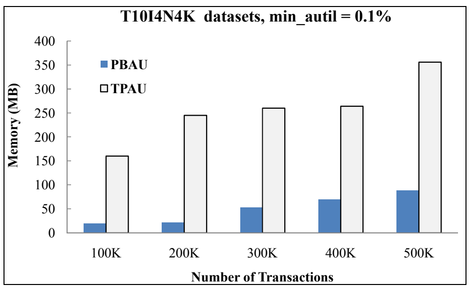

Fig. 5. The difference in the memory cost long with different minimum average-utility thresholds.

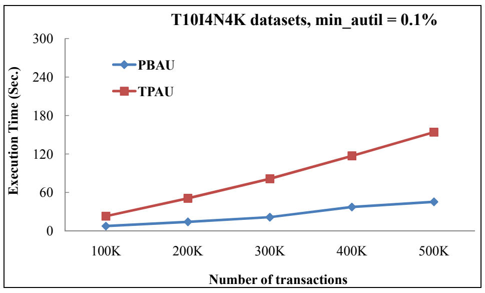

Fig. 6. The execution time along with different minimum average-utility thresholds.

It was seen from these figures that the proposed *PBAU* approach obviously performed better than the traditional *TPAU* algorithm for the synthetic datasets with regard to the number of candidate itemsets, memory requirement and execution efficiency. The main reason was that the proposed *PBAU* algorithm adopted the prefix concept to efficiently find high average-utility itemsets. In addition, it used the designed pruning strategy to effectively reduce the number of unpromising candidates in the recursive mining process. The additional memory requirement from the index tables was less than that from copying the original database for mining. With the indexing technique presented in this work, the proposed *PBAU* algorithm could thus find high average-utility itemsets with only a little additional load. The effects were even better when the minimum average-utility threshold value decreased.

On the other hand, although the performance of the *PBAU* algorithm was not evaluated when the database could not be fitted into the memory, the concept of partition techniques [11, 16] could be applied to the proposed algorithm to effectively accomplish the mining task. A large database could be divided into several sub-databases which could be fitted into the memory by using the partition techniques. The identifier of each transaction in each sub-database could be recorded and kept into the memory by using indexing strategy. Thus, transactions in each sub-database could be quickly accessed by the indexing structure to find high average-utility itemsets. A large-scale database could thus still be efficiently processed by the proposed algorithm when the partition techniques [11, 16] were adopted.

5.4 Evaluation on a Real Dataset

The real dataset *BMS-POS* was used to evaluate the performance of the two algorithms *PBAU* and *TBAU* under different thresholds. Fig. 7 showed the difference in the number of candidates generated by the algorithms for different thresholds, varying from 1.00% to 0.20%. Fig. 8 showed the difference in the memory cost consumed by the algorithms for different thresholds, varying from 1.00% to 0.20%. Finally, Fig. 9 showed execution time for different thresholds, varying from 1.00% to 0.20%. As could be seen, the performance of the proposed *PBAU* approach was better than that of the traditional *TPAU* algorithm on the real dataset *BMS-POS* when the minimum average-utility threshold value decreased. The reason was the same as that mentioned previously. The performance of *PBAU* was thus better than that of *TPAU* for discovering the high average-utility itemsets in the real dataset *BMS-POS*.

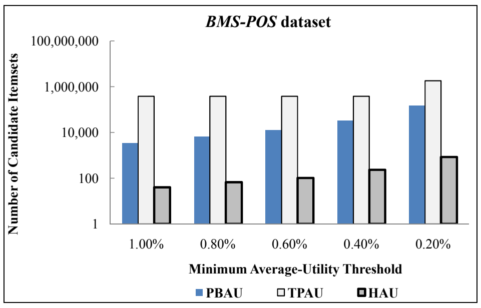

Fig. 7. The difference in the numbers of candidate itemsets along with different thresholds.

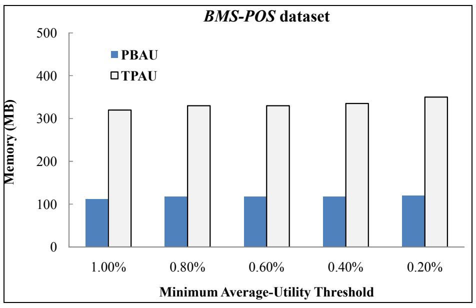

Fig. 8. The difference in the memory cost along with different thresholds.

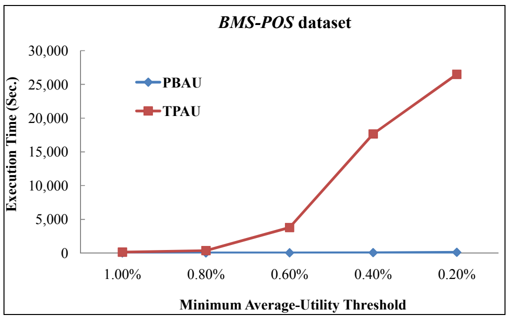

Fig. 9. The execution time of the two algorithms along with different thresholds.

6. CONCLUSIONS

In this paper, we have proposed an efficient projection-based average-utility mining approach (*PBAU*), to achieve the average-utility mining task. In particular, an indexing structure is designed to quickly link the transactions of each itemset to be processed in the database. By using the structure, the proposed algorithm can directly generate the required itemsets from the transactions in the mining process. In addition, the original database is not copied for each itemset to find high average-utility itemsets, but instead, they are directly extracted through the indexing structure. The memory consumed is thus less than that needed by directly copying the original database for mining. Finally, the proposed pruning strategy could effectively skip unpromising itemsets and thus further save time. The experimental results show that the memory requirement was less than that needed by the traditional *TPAU* algorithm, and the proposed *PBAU* algorithm was able to execute faster than *TPAU* for the two real dataset *BMSPOS* and *chess*.

REFERENCES

1. R. Agrawal and R. Srikant, "Fast algorithm for mining association rules," in *Proceedings of International Conference on Very Large Data Bases*, 1994, pp. 487-499.
2. R. Agrawal, R. Srikant, and Q. Vu, "Mining association rules with item constraints,"

in *Proceedings of the 3rd International Conference on Knowledge Discovery in Databases and Data Mining*, 1997, pp. 67-73.

3. R. Agrawal, T. Imielinksi, and A. Swami, "Mining association rules between sets of items in large database," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1993, pp. 207-216.
4. R. Chan, Q. Yang, and Y. Shen, "Mining high utility itemsets," in *Proceedings of the 3rd IEEE International Conference on Data Mining*, 2003, pp. 19-26.
5. C. J. Chu, V. S. Tseng, and T. Liang, "Mining temporal rare utility itemsets in large databases using relative utility thresholds," *International Journal of Innovative Computing*, *Information and Control*, Vol. 4, 2008, pp. 2775-2792.
6. Frequent Itemsets Mining Dataset Repository (FIMI), http://fimi.cs.hels inki.fi/data/.
7. J. Hu and A. Mojsilovic, "High-utility pattern mining: A method for discovery of high-utility item sets," *Pattern Recognition*, Vol. 40, 2007, pp. 3317-3324.
8. IBM Quest Data Mining Project, "Quest synthetic data generation code," http://www.almaden.ibm.com/cs/quest/syndata.html.
9. T. P. Hong, C. H. Lee, and S. L. Wang, "Effective utility mining with the measure of average utility," *Expert Systems with Application*, Vol. 38, 2011, pp. 8259-8265.
10. G. C. Lan, T. P. Hong, and V. S. Tseng, "Discovery of high utility itemsets from on-shelf time periods of products," *Expert Systems with Application*, Vol. 38, 2011, pp. 5851-5857.
11. K. Lal and N. C. Mahanti, "Mining association rules in large database by implementing pipelining technique in partition algorithm," *International Journal of Computer Applications*, Vol. 2, 2010, pp. 33-39.
12. Y. C. Li, J. S. Yeh, and C. C. Chang, "Isolated items discarding strategy for discovering high utility itemsets," *Data and Knowledge Engineering*, Vol. 64, 2008, pp. 198-217.
13. Y. Liu, W. Liao, and A. Choudhary, "A fast high utility itemsets mining algorithm," in *Proceedings of Utility-Based Data Mining Workshop*, 2005, pp. 90-99.
14. J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, "Mining sequential patterns by pattern-growth: The PrefixSpan approach," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, 2004, pp. 1424-1440.
15. V. S. Tseng, C. J. Chu, and T. Liang, "Efficient mining of temporal high utility itemsets from data streams," in *Proceedings of ACM International Conference on Utility-Based Data Mining*, 2006, pp. 18-27.
16. A. Savasere, E. Omiecinski, and S. B. Navathe, "An efficient algorithm for mining association rules in large databases," in *Proceedings of International Conference on Very Large Data Bases*, 1995, pp. 432-444.
17. H. Yao and H. J. Hamilton, "Mining itemset utilities from transaction databases," *Data and Knowledge Engineering*, Vol. 59, 2006, pp. 603-626.
18. H. Yao, H. J. Hamilton, and C. J. Butz, "A foundational approach to mining itemset utilities from databases," in *Proceedings of the 4th SIAM International Conference on Data Mining*, 2004, pp. 482-486.
19. J. S. Yeh, C. Y. Chang, and Y. T. Wang, "Efficient algorithms for incremental utility mining," in *Proceeding of International Conference on Ubiquitous Information Management and Communication*, 2008, pp. 229-234.

20. G. Yu, K. Li, and S. Shao, "Mining high utility itemsets in large high dimensional data," in *Proceedings of International Workshop on Knowledge Discovery and Data Mining*, 2008, pp. 17-20.

Guo-Cheng Lan (藍國誠) received his B.S. and M.S. degrees from the Department of Information Management in Southern Taiwan University, Taiwan, in 2000 and 2004, respectively. He is currently a Ph.D. student in the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan. His research interests include data mining, web mining, medical data mining, ontology knowledge, and fuzzy theory.

Tzung-Pei Hong (洪宗貝) received his B.S. degree in Chemical Engineering from National Taiwan University in 1985, and his Ph.D. degree in Computer Science and Information Engineering from National Chiao Tung University in 1992. He was in charge of the whole computerization and library planning for National University of Kaohsiung in Preparation from 1997 to 2000 and served as the first director of the library and computer center in National University of Kaohsiung from 2000 to 2001, as the Dean of Academic Affairs from 2003 to 2006 and as the Vice President from 2007 to 2008 and from 2010 to 2011. He is currently a Professor at the Department of Computer Science and Information Engineering and at the Department of Electrical Engineering. His current research interests include parallel processing, machine learning, data mining, soft computing, management information systems, and www applications.

Vincent S. Tseng (曾新穆) is currently a Professor at Department of Computer Science and Information Engineering at National Cheng Kung University (NCKU), Taiwan. Before this, he was a postdoctoral research fellow in Computer Science Division of University of California at Berkeley, U.S.A. during January 1998 and July 1999. He has acted as the director for Institute of Medical Informatics of NCKU since August 2008. During February 2004 and July 2007, he had also served as the director for Informatics Center in National Cheng Kung University Hospital, Taiwan. Dr. Tseng received his Ph.D. degree from National Chiao Tung University, Taiwan, in 1997, majored in computer science. Dr. Tseng has a wide variety of research interests covering data mining, biomedical informatics, mobile computing and web technologies. He has published more than 180 research papers in referred journals and international conferences. He has also held (or filed) more than 15 patents in U.S.A. and Taiwan. He is a member of IEEE, ACM and honorary member of Phi Tau Phi Society. He also served as board member for various societies like Taiwanese Association for Artificial Intelligence and Taiwan Bioinformatics Society. Dr. Tseng is on the editorial board of International Journal of Data Mining and Bioinformatics, Journal of Information Science and Engineering. He has also served as chairs/program committee for a number of premier international conferences related to data mining and databases.