ISSN:2231-0711

Utility Pattern Approach for Mining High Utility Log Items from Web Log Data

T.Anitha , M.S.Thanabal

Department of CSE, PSNA College of Engineering and Technology, Dindigul

Abstract—. Mining frequent log items is an active area in data mining that aims at searching interesting relationships between items in databases. It can be used to address a wide variety of problems such as discovering association rules, sequential patterns, correlations and much more. Weblog that analyzes a Web site's access log and reports the number of visitors, views, hits, most frequently visited pages, and so forth. Mining frequent log items from web log data can help to optimize the structure of a web site and improve the performance of web servers. Existing methods often generate a huge set of potential high utility log items and their mining performance is degraded consequently. Two novel algorithms as well as a compact data structure for efficiently discovering high utility log items are proposed. High utility log items are maintained in a tree-based data structure called utility pattern tree. Implementing mining process is done through Discarding Local Unpromising Items and Decreasing Local Node Utility strategies. Experimental results predict that these strategies can keep track of previously accessed pages of a user, identify needed links to improve the overall performance of a web page, and improve the actual design of web pages with only two database scans.

Index Terms— frequent log items, high utility log items, Web Log file, data mining

.

1. INTRODUCTION

Data mining is the process of revealing non-trivial, previously unknown and potentially useful information from large databases. Discovering useful patterns hidden in a database plays an essential role in several data mining tasks, such as frequent pattern mining, weighted frequent pattern mining and high utility pattern mining. Among them, frequent pattern mining is a fundamental research topic that has been applied to different kinds of databases, such as transactional databases,Streaming databases and time series databases and various application domains, such as bioinformatics, Web click-stream analysis and mobile environments. Web mining is used to discover interest patterns which can be applied to many real world problems like improving web sites, better understanding the visitor's behavior, product recommendation etc.

Web usage mining is one of the prominent research areas due to these following reasons. a) One can keep track of previously accessed pages of a user. These pages can be used to identify the typical behavior of the user and to make prediction about desired pages. Thus personalization for a user can be achieved through web usage mining. b) Frequent access behavior for the users can be used to identify needed links to improve the overall performance of future accesses.Prefetching and caching policies can be made on the basis of frequently accessed pages to improve latency time. c) Common access behaviors of the users can be used to improve the actual design of web pages and for making other modifications to a Web site. d) Usage patterns can be used for business intelligence in order to improve sales and advertisement by providing product recommendations.

2. RELATED WORK

The frequent pattern mining techniques for discovering different types of patterns in a Web log. Web mining involves a wide range of applications that aims at discovering and extracting hidden information in data stored on the Web. Another important purpose of Web mining is to provide a mechanism to make the data access more efficiently and adequately. The third interesting approach is to discover the information which can be derived from the activities of users, which are stored in log files for example for predictive Web caching. Thus, Web mining can be categorized into three different classes based on which part of the Web is to be mined; these three categories are (i) Web content mining, (ii) Web structure mining and (iii) Web usage mining. Web content mining is the task of discovering useful information available on-line. There are different kinds of Web content which can provide useful information to users, for example multimedia data, structured (i.e. XML documents), semi-structured (i.e. HTML documents) and unstructured data (i.e. plain text). The aim of Web content mining is to provide an efficient mechanism to help the users to find the information they seek. Web content mining includes the task of organizing and clustering the documents and providing search engines for accessing the different documents by keywords, categories, contents etc.

Existing methods often generate a huge set of potential high utility item sets and their mining performance is degraded consequently. This situation may become worse when databases contain many long transactions or low thresholds are set. The huge number of potential high utility item sets forms a challenging problem to the mining performance since the more potential high utility item sets the algorithm generates, the higher processing time it consumes. To address this issue, we propose two novel algorithms as well as a compact data structure for efficiently discovering high utility item sets from transactional databases.

Major contributions of this work are summarized as follows:

1. Two algorithms, named *UP-Growth* (*Utility Pattern Growth*) and *UP-Growth+*, and a compact tree structure, called *UP-Tree* (*Utility Pattern Tree*), for discovering high utility item sets and maintaining important information related to utility patterns within

databases are proposed. High utility item sets can be generated from UP-Tree efficiently with only two scans of original databases.

2. Several strategies are proposed for facilitating the mining processes of UP-Growth and UP-Growth+ by maintaining only essential information in UP-Tree. By these strategies, overestimated utilities of candidates can be well reduced by discarding utilities of the items that cannot be high utility or are not involved in the search space. The proposed strategies can not only decrease the overestimated utilities of potential high utility item sets but also greatly reduce the number of candidates.
3. Different types of both real and synthetic datasets are used in a series of experiments to compare the performance of the proposed algorithms with the state-of the-art utility mining algorithms. Experimental results show that UP-Growth and UP-Growth+ outperform other algorithms substantially in terms of execution time, especially when databases contain lots of long transactions or low minimum utility thresholds are set.

3. ORGANIZING LOG FILES

The Log Files are collected from the data catalogs. The patterns are generated as per the logic such as each user is initialized with their own id.The quantities which determine the number of times user accessed the websites. For each log, the profit table is initialized. However the transaction utility (TU) hereby called as log utility (LU) is estimated by multiplying the quantity and log Profit value.
Given a finite set of items I = {i1, i2, …, im}, each item ip has a unit profit pr(ip). An item set X is a set of k distinct items {i1, i2, …, ik}, where ij□I, k is the length of X. An item set with length k is called a k-item

TABLE 1.
AN EXAMPLE DATABASE

TID	Transaction	TU
T_1	(A,1) (C,10) (D,1)	17
T_2	(A,2) (C,6) (E,2) (G,5)	27
T_3	(A,2) (B,2) (D,6) (E,2) (F,1)	37
T_4	(B,4) (C,13) (D,3) (E,1)	30
T_5	(B,2) (C,4) (E,1) (G,2)	13
T_6	(A,1) (B,1) (C,1) (D,1) (H,2)	12

TABLE 2.
PROFIT TABLE

Item	A	B	C	D	E	F	G	H
Profit	5	2	1	2	3	5	1	1

For example, in Tables 1 and 2, *u*({A}, T1) = 5×1 = 5; *u*({AD},T1) = *u*({A},T1) + *u*({D},T1) = 5+2 = 7; *u*({AD}) =*u*({AD},T1) + *u*({AD},T3) + *u*({AD},T6) = 7+22+7 = 36. If *min_util* is set to 30, {AD} is a high utility item set.

3.1Transaction-weighted Downward Closure

Compute the minimum weighted utility. Compute the Transaction utility of a transaction Td.Compute the Transaction-weighted utility of an item set X is the sum of the transaction utilities of all the transactions containing X, which is denoted as TWU(X).Estimate the high transaction weighted utility item set . It is the one which is not less than min_util.Evaluate the Transaction Weighted Downward Closure by downward closure property which can be done by applying the transaction weighted utility

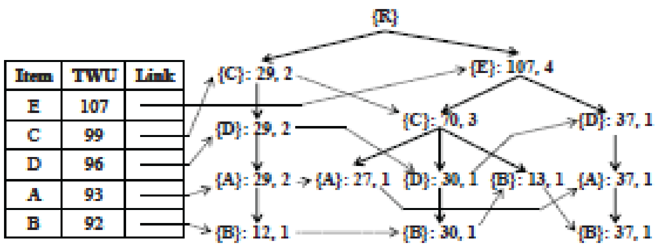


Fig. 1. An IHUP-Tree when min_util = 40.

To efficiently generate HTWUIs in phase I and avoid scanning database too many times, Ahmed et al. [3] proposed a tree-based algorithm, named IHUP. A tree-based structure called IHUP-Tree is used to maintain the information about item sets and their utilities. Each node of an IHUP-Tree consists of an item name, a TWU value and a support count. IHUP algorithm has three steps: (1) construction of IHUP-Tree, (2) generation of HTWUIs and (3) identification of high utility item sets. In step 1, items in transactions are rearranged in a fixed order such as lexicographic order, support descending order or TWU descending order. Then the rearranged transactions are inserted into an IHUP-Tree. Fig. 1 shows the global IHUPTree
for the database in Table 1, in which items are arranged in the descending order of TWU. For each node in Fig. 1, the first number beside item name is its TWU and the second one is its support count. In step 2, HTWUIs are generated from the IHUP-Tree by applying FP-Growth [14]. Thus, HTWUIs in phase I can be found without generating any candidate for HTWUIs. In step 3, high utility item sets and their utilities are identified from the set of HTWUIs by scanning the original database once..

3.2 Utility Pattern-Tree

To facilitate the mining performance and avoid scanning original database repeatedly, we use a compact tree structure, named *UP-Tree* (*Utility Pattern Tree*), to maintain the information of transactions and high utility item sets. Two strategies are applied to minimize the overestimated utilities stored in the nodes of global UP-Tree. In following subsections, the elements of UP-Tree are first defined. Next, the two strategies are introduced. Finally, how to construct an UP-Tree with the two strategies is illustrated in detail by a running example.

3.2.1 The Elements in UP-Tree

In a UP-Tree, each node *N* consists of *N.name*, *N.count*,*N.nu*, *N.parent*, *N.hlink* and a set of child nodes. *N.name* is the node's item name. *N.count* is the node's support count.*N.nu* is the node's *node utility*, i.e., overestimated utility of the node. *N.parent* records the parent node of *N*. *N.hlink* is a node link which points to a node whose item name is the same as *N.name*.A table named *header table* is employed to facilitate the traversal of UP-Tree. In header table, each entry records an item name, an overestimated utility, and a link. The link points to the last occurrence of the node which has the same item as the entry in the UP-Tree. By following the links in header table and the nodes in UP-Tree, the nodes having the same name can be traversed efficiently. In following subsections, two strategies for decreasing the overestimated utility of each item during the construction of a global UP-Tree are introduced.

3.2.2 Strategy DGU: Discarding Global Unpromising Items during Constructing a Global UP-Tree

The construction of a global UP-Tree can be performed with two scans of the original database. In the first scan,TU of each transaction is computed. At the same time,TWU of each single item is also accumulated. By TWDC property, an item and its supersets are unpromising to be high utility item sets if its TWU is less than the minimum utility threshold. Such an item is called an *unpromising item*. Definition 8 gives a formal definition of what are unpromising items and promising items

3.2.3Constructing a global UP-Tree by Applying DGU and DGN

Recall that the construction of a global UP-Tree is performed with two database scans. In the first scan, each Transaction's TU is computed; at the same time, each 1-Item's TWU is also accumulated. Thus we can get promising items and unpromising items. After getting all promising items, DGU is applied. The transactions are reorganized by pruning the unpromising items and sorting the remaining promising items in a fixed order. Any ordering Can be used such as the lexicographic, support or TWU Order. Each transaction after the above reorganization is Called a *reorganized transaction.* In the following paragraphs, we use the TWU descending order to explain the whole process since it is mentioned that the performance of this order.

```
Subroutine: Insert_Reorganized_Path (N, i x )
Line 1: If N has a child N ix such that N ix .item i x,
        increment N ix .count by p j .count.
        Otherwise, create a new child node N ix with
        N ix .item
Line 2: Increase N ix .nu
Line 3:If x n, callinsert_Reorganized_Transaction(N ix
        , i x 1 )
```

Fig. 2. The subroutine of Insert_Reorganized_Path

TABLE 3.
REORGANIZED TRANSACTIONS AND THEIR RTUS

TID	Reorganized transaction	RTU
T1'	(C,10) (D,1) (A,1)	17
T2'	(E,2) (C,6) (A,2)	22
T3'	(E,2) (D,6) (A,2) (B,2)	32
T4'	(E,1) (C,13) (D,3) (B,4)	30
T5'	(E,1) (C,4) (B,2)	11
T6'	(C,1) (D,1) (A,1) (B,1)	10

Then a function *Insert_Reorganized_Transaction* is called to apply DGN during constructing a global UP-Tree. Its subroutine is shown in Fig. 2. When a reorganized transaction
tj' = {*i1*, *i2*, …, *in*}) is inserted into a global UP-Tree, *Insert_Reorganized_Transaction*(N, *ix*) is called, where N is a node in UP-Tree and *ix* is an item in *tj'*(*ix*□*tj'*, 1 □ *x* □ *n*). First, (NR, *i1*) is taken as input, where NR is the root node of UP-Tree. The node for *i1*,1 N*i* , is found or created under NR and its support is updated in Line 1.
Then DGN is applied in Line 2 by discarding the utilities of descendant nodes under 1 N*i* , i.e.,2 N*i* to N . *in* Finally in Line 3, (1 N*i* , *i2*) is taken as input recursively. An example is given to explain how to apply the two strategies during the construction of a global UP-Tree. Consider the transaction database in Table 1 and the profit table in Table 2. Suppose *min_util* is 50. In the first scan of database, TUs of all transactions and TWUs of distinct items are computed. Five promising items, i.e.,{A}:93, {B}:92, {C}:99, {D}:96 and {E}:107, are sorted in the header table by the descending order of TWU, that is, {E},{C}, {D}, {A} and {B}. Then the transactions are reorganized by sorting promising items and subtracting utilities of unpromising items from their TUs. The reorganized transactions and their RTUs are shown in Table 3. Comparing Table 3 and Table 1, the RTUs of T2, T3 and T5 in Table 3 are less than the TUs in Table 1 since the utilities of {F}, {G} and {H} have been removed by DGU.After a transaction has been reorganized, it is inserted into the global UP-Tree. When T1' = {(C,10)(D,1)(A,1)} is inserted, the first node NC is created with NC.*item* = {C}and NC.*count* = 1. NC.nu is increased by RTU(T1') minus the utilities of the rest items that are behind {C} in T1', that is, NC.*nu* = RTU(T1') – (*u*({D},T1') + *u*({A},T1')) = 17–(2+5) =10. Note that it can also be calculated as the sum of utilities of the items that are before item {D} in T1', i.e., NC.*nu*= u({C},T1') = 10. The second node ND is created with ND.*item* = {D}, ND.*count* = 1 and ND.*nu* = RTU(T1') –*u*({A},T1') = 17–5 = 12. The third node NA is created with NA.*item* = {A}, NA.*count* = 1 and NA.*nu* = RTU(T1') = 17.After inserting all reorganized transactions by the same way, the global UP-Tree shown in Fig. 3 is constructed.Comparing with the IHUP-Tree in Fig. 1, node utilities of the nodes in UP-Tree are less than those in IHUP-Tree since the node utilities are effectively decreased by the two strategies DGU and DGN.

3.3 Utility Pattern-Growth

After constructing a global UP-Tree, a basic method for generating PHUIs is to mine UP-Tree by FP-Growth [14].However too many candidates will be generated. Thus, we propose an algorithm *UP-Growth* (*Utility Pattern Growth*) by pushing two more strategies into the framework of FP-Growth

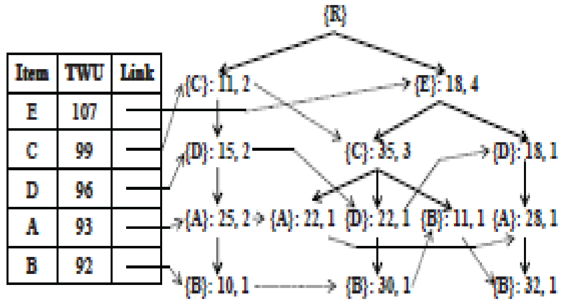


Fig. 3. A UP-Tree by applying strategies DGU and DGN.

TABLE 4.
MINIMUM ITEM UTILITY TABLE

Item	A	B	C	D	E
Minimum item utility	5	2	1	2	3

By the strategies, overestimated utilities of item sets can be decreased and thus the number of PHUIs can be further reduced. In following subsections, we first propose the two strategies and then describe the process of UP-Growth in detail by an example.

3.3.1 Strategy DLU: Discarding Local Unpromising Items during Constructing a Local UP-Tree

The common method for generating patterns in tree based algorithms [3, 14] contains three steps: (1) Generate *conditional pattern bases* by tracing the paths in the original tree, (2) construct *conditional trees* (also called *local trees* in this paper) by the information in conditional pattern bases and (3) mine patterns from the conditional trees. However, strategies DGU and DGN can not be applied
into conditional UP-Trees since actual utilities of items in different transactions are not maintained in a global UP Tree. We cannot know actual utilities of unpromising items that need to be discarded in conditional pattern bases unless an additional database scan is performed. To overcome this problem, a naïve solution is to maintain items' actual utilities in each transaction into each node of global UP-Tree. However, this is impractical since it needs lots of memory space. In view of this, we propose two strategies, named DLU and DLN, that are applied in the first two mining steps and introduced in this and next subsections, respectively. For the two strategies, we maintain a *minimum item utility table* to keep *minimum item utilities* for all global promising items in the database.

For example, *pu*(<ADC>,{B}-CPB), which is the path utility of the leftist path in Figure 3 in {B}-CPB, is defined as NB.*nu*, i.e., 10, in that path. By Definitions 9 and 10,assume that there is a path *p* in {*im*}-CPB and *im CPBUI* { }is the set of unpromising items in {*im*}-CPB. Path utility of *p*in {*im*}-CPB, i.e., *pu*(*p*,{*im*}-CPB), is recalculated and reduced
According to minimum item utilities as below:

$$pu(p,\{i_m\}-CPB) = N_{i_m}.nu - \sum_{\forall i \in UI_{\{i_m\}-CPB} \wedge i \subseteq p} miu(i) \times p.count \quad (1),$$

where *p.count* is the support count of *p* in {*im*}-CPB.

3.3.2 Strategy DLN: Decreasing Local Node Utilities during Constructing a Local UP-Tree

As mentioned in the subsection 3.1.3, since {*im*}-Tree must not contain the information about the items below *im* in the original UP-Tree, we can discard the utilities of descendant nodes related to *im* in the original UP-Tree while building {*im*}-Tree. (Here, *original UP-Tree* means the UP Tree which is used to generate {*im*}-Tree.) Because we cannot know actual utilities of the descendant nodes, we use minimum item utilities to estimate the discarded utilities.

3.3.3 UP-Growth: Mining a UP-Tree by Applying DLU and DLN

The process of mining PHUIs by UP-Growth is described as follows. First, the node links in UP-Tree corresponding to the item *im*, which is the bottom entry in header table, are traced. Found nodes are traced to root of the UP-Tree to get paths related to *im*. All retrieved paths, their path utilities and support counts are collected into *im*'s conditional pattern base.A conditional UP-Tree can be constructed by two scansof a conditional pattern base. For the first scan, local promising and unpromising items are learned by summing the path utility for each item in the conditional pattern base. Then, DLU is applied to reduce overestimated

```
Subroutines: UP-Growth (TX, HX, X)
Input: A UP-Tree TX, a header table HX for TX, an item set X, and a  minimum utility threshold min_util.
Output: All PHUIs in TX.
(1) For each entry ik in HX do
(2) Trace each node related to ik via ik.hlink and accumulate ik.nu to nusum (ik)
   /* nusum (ik): the sum of node utilities of ik */
(3) If nusum (ik)min_util, do
(4) Generate a PHUI Y = X ik ;
(5)  Set pu(ik) as estimated utility of Y;
(6) Construct Y-CPB;
(7)Put local promising items in Y-CPB into HY
(8) Apply DLU to reduce path utilities of the paths;
(9)Apply Insert_Reorgnized_Path to insert paths into TY with DLN;
(10)If TY null then call UP-Growth (TY, HY, Y);
(11) End if
(12)End for
```

Fig. 4. The subroutine of UP-Growth.

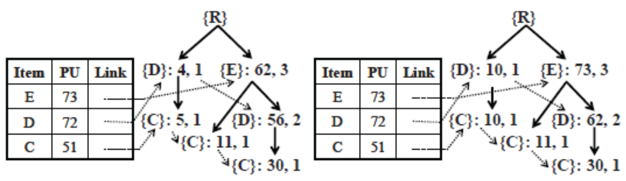
Utilities during the second scan of the conditional pattern base. When a path is retrieved, unpromising items and their estimated utilities are eliminated from the path and its path utility by Eq (1). Then the path is reorganized by the descending order of path utility of the items in the conditional pattern base.DLN is applied during inserting reorganized paths into a conditional UP-Tree. Assume a reorganized path *pj*= <1 N*i*2 N*i* ...' N *im* >, where*ik* N is the nodes in UP-Tree and 1 □ *k* □ *m'*. When *item i* N . 1 , *i1*, is inserted into the conditional UP-Tree,thefunction in*ert_Reorgnized_Path*(N*R'*,*i1*), as shown in Fig. 4, is called, where N*R'* is root node of the conditional UP-Tree. The node for *i1*,1 N*i* , is found or created under N*R'* and its support is updated in Line 1.Then DLN is applied in Line 2 by decreasing estimated utilities of descendant nodes under 1 N*i* , i.e.,2 N*i* to N . *im*'Finally in Line 3, (1 N*i* , *i2*) is taken as input recursively.

3.4 An Improved Mining Method: UP-Growth+

UP-Growth achieves better performance than FP-Growth by using DLU and DLN to decrease overestimated utilities of item sets. However, the overestimated utilities can be closer to their actual utilities by eliminating the estimated utilities that are closer to actual utilities of unpromising items and ascendant nodes.

TABLE 5.
{B}-CPB AFTER APPLYING DGU, DGN AND DLU

Retrieved path: Path utility	Reorganized path: Path utility (after DLU)	Support count
<ADC>: 10	<DC>: 5	1
<DCE>: 30	<EDC>: 30	1
<CE>: 11	<EC>: 11	1
<ADE>: 32	<ED>: 27	1



(a) Four strategies (b) Strategy DGU and DGN

Fig.5 {B}-Trees with different strategies

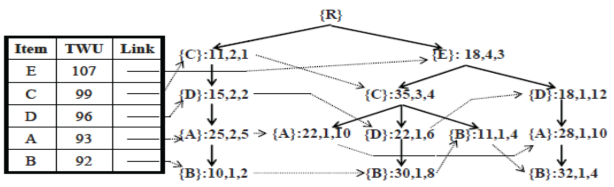


Fig. 6. A UP-Tree with minimal node utilities.

Assume that there is a path *p* in {*im*}-CPB and *im CPB UI* { }□is the set of unpromising items in {*im*}-CPB. The path utility of *p* in {*im*}-CPB, i.e., *pu*(*p*,{*im*}-CPB), is recalculated as below equation:

$$pu(p,\{i_m\}-CPB) = p.\{i_m\}.nu - \sum_{\forall i \in UI_{\{i_m\}-CPB} \wedge i \subseteq p} mnu(i) \times p.count \quad (2)$$

Where *p.count* is the support count of *p* in {*im*}-CPB.Assume that a reorganized path *p* = <1 N'*i* 2 N'*i* ..' N' *im* >in {*im*}-CPB is inserted into the path <1 N*i* 2 N*i* ...' N *im* > in {*im*}-Tree, where *m'* □ *m*. For the node *ik* N in {*im*}-Tree, where 1 □ *k* □ *m'*, *nu ik* N . is recalculated as below:

$$N_{i_k}.nu_{new} = N_{i_k}.nu_{old} + pu(p,\{i_m\}-CPB) - \sum_{j=k+1}^{m'} mnu(i_j) \times p.count \quad (3)$$

where *i old nu k* N . is the node utility of *ik* N in {*im*}-Tree before adding *p*.

TABLE 6.
PARAMETER SETTINGS OF SYNTHETIC DATASETS.

Parameter Descriptions		Default
|D|:	Total number of transactions	100k
T:	Average transaction length	10
|I|:	Number of distinct items	1000
F:	Average size of maximal potential frequent itemsets	6
Q:	Maximum number of purchased items in transactions	10

TABLE 7
CHARACTERISTICS OF REAL DATASETS

Dataset	|D|	T	|I|	Type
Accidents	340,183	33.8	468	Dense
Chain-store	1,112,949	7.2	46,086	Sparse
Chess	3,196	37.0	75	Dense
Foodmart	4,141	4.4	1,559	Sparse

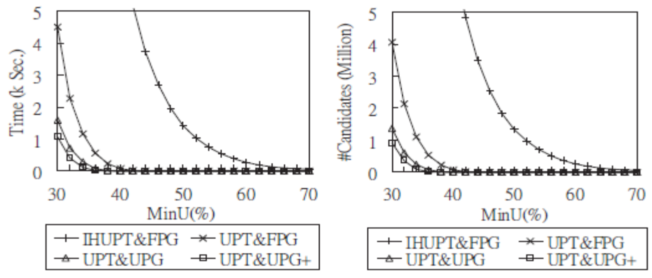
Consider the UP-Tree in Fig. 7 and assume that *min_util* is set to 50. First, node links of the bottom entry {B} in header table are traced. Four paths are retrieved and added into {B}-CPB: {<A(5)D(2)C(1)>: 10, 1},{<D(6)C(4)E(3)>: 11, 1}, {<C(4)E(3)>: 30, 1} and {<A(10)D(12)E(3)>: 32, 1}. Note that the number in bracket beside each item is minimal node utility recorded in that node
After mining the whole UP-Tree by UP-Growth+, we can obtain all PHUIs, i.e., {A}:75, {B}:83 and {D}:55 in the UP-Tree. In this example, the number of PHUIs of UPGrowth+is less than that of UP-Growth. It means that the number of PHUIs, as well as the overestimated utilities of item sets, are further reduced by UP-Growth+.

4. PERFORMANCE EVALUATION

Performance of the proposed algorithms is evaluated in this section. The experiments were performed on a 2.80 GHz Intel Pentium D Processor with 3.5 GB memory. The operating system is Microsoft Windows 7. The algorithms are presented in Java language. Both real and synthetic datasets are used in the experiments. Synthetic datasets were generated from the data generator .Parameter descriptions and default values of synthetic datasets are shown in Table 7. Real world data sets Accidents and Chess are obtained from FIMI repository ; Chain-store is obtained from NU-Mine Bench 2.0 ;
Food mart is acquired from Microsoft food mart 2000 database.Table 8 shows characteristics of the above datasets. In the above datasets, except Chain-store and Food mart, unit profits for items in utility tables are generated

TABLE 8.
CHARACTERISTICS OF REAL DATASETS

Dataset	|D|	T	|I|	Type
Accidents	340,183	33.8	468	Dense
Chain-store	1,112,949	7.2	46,086	Sparse
Chess	3,196	37.0	75	Dense
Foodmart	4,141	4.4	1,559	Sparse



(a) Runtime for phase I on Chess (b) Number of candidates on Chess

Fig. 7 Performance Comparison on Dense Dataset.

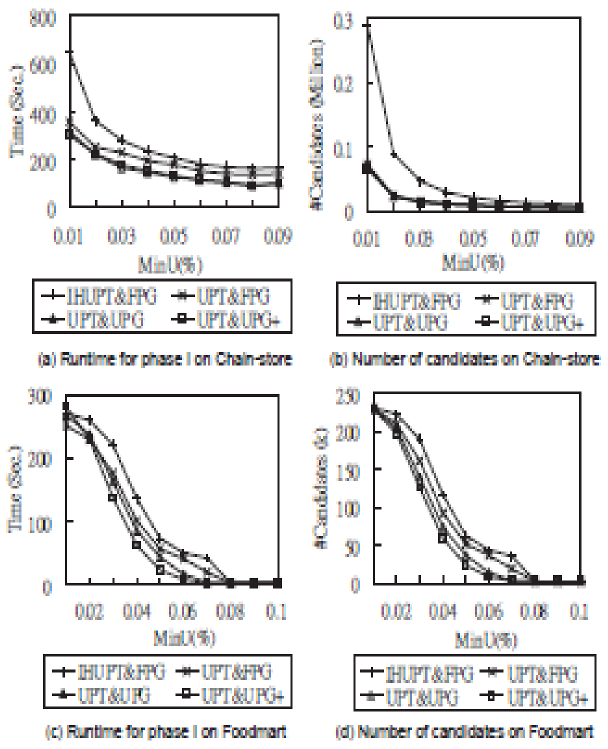


Fig. 8. Performance Comparison on Sparse Datasets.

In Fig. 7 (a), we can observe that the performance of proposed methods substantially outperforms that of previous methods. The runtime of IHUPT&FPG is the worst, followed by UPT&FPG,UPT&UPG and UPT&UPG+ is the best. The main reason is the performance of IHUPT&FPG and UPT&FPG is decided by the number of generated candidates. In Fig. 7(b)runtime of the methods is just proportional to their number of candidates, that is, the more candidates the method produces, the greater its execution time.

Experimental results on real sparse datasets are shown in Fig. 8. The performance on Chain-store dataset is shown in Fig. 8 (a) and (b)the runtime of IHUPT&FPG is the worst, followed by UPT&FPG,UPT&UPG and UPT&UPG+ is the best. The performance of IHUPT&FPG is the worst since it generates the most candidates. Besides, although the number of candidates of UPT&FPG, UPT&UPG and UPT&UPG+ are almost the same, the execution time of UPT&FPG is the worst among the three methods since UP-Growth+ and UP-Growth efficiently prune the search space of local UP-Trees. Fig. 8 (c) and (d) show the results on Foodmart dataset.In Fig.8 (d)number of candidates of the compared methods is almost the same when *min_util* is larger than 0.08% or less than 0.01%. The reason is that when *min_util* is larger than 0.08%, few redundant candidates whose length are larger than 2 are generated by IHUP&FPG. On the other hand, when *min_util* is less than 0.01%, since it is too small, almost all possible candidates are generated by all compared methods. When *min_util* is between 0.02% and 0.08%, the best performer is UPT&UPG+, followed by UPT&UPG, UPT&FPG, and finally IHUPT& FPG. However, when *min_util* is 0.01%, the number of candidates is almost the same for each method.

CONCLUSION

In this paper, we have proposed two efficient algorithms named UP-Growth and UP-Growth+ for mining high utility item sets from transaction databases. A data structure named UP-Tree was proposed for maintaining the information of high utility item sets. Potential high utility item sets can be efficiently generated from UP-Tree with only two database scans. Moreover, we developed several strategies to decrease overestimated utility and enhance the performance of utility mining. In the experiments, both real and synthetic datasets were used to perform a thorough performance evaluation. Results show that the strategies considerably improved performance by reducing both the search space and the number of candidates. Moreover, the proposed algorithms, especially UPGrowth+, outperform the state-of-the-art algorithms substantially especially when databases contain lots of long transactions or a low minimum utility threshold is used.

REFERENCES

[1] R. Agrawal and R. Srikant. "Fast algorithms for mining association rules," in *Proc. of the 20th VLDB Conf.*, pp. 487-499, 1994.
[2] R. Agrawal and R. Srikant, "Mining Sequential Patterns," in *Proc. of the 11th Int'l Conference on Data Engineering*, pp. 3-14, Mar., 1995.
[3] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong and Y.-K. Lee. "Efficient tree structures for high utility pattern mining in incremental databases," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 21, Issue 12, pp. 1708-1721, 2009.
[4] C. H. Cai, A. W. C. Fu, C. H. Cheng and W. W. Kwong, "Mining Association Rules with Weighted Items," in *Proc. of the Int'l Database Engineering and Applications Symposium* (IDEAS 1998), pp. 68-77, 1998.
[5] R. Chan, Q. Yang and Y. Shen. "Mining high utility item sets," in *Proc. of Third IEEE Int'l Conf. on Data Mining*, pp. 19-26, Nov., 2003.
[6] J. H. Chang, "Mining weighted sequential patterns in a sequence database with a time-interval weight," *Knowledge-Based Systems*, Vol. 24, Issue 1, 2011.
[7] M.-S. Chen, J.-S. Park and P. S. Yu, "Efficient data mining for path traversal patterns," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10, no. 2,pp. 209-221, 1998.
[8] C. Creighton and S. Hanash, "Mining Gene Expression Databases for Association Rules," *Bioinformatics*, Vol. 19, No. 1, pp. 79-86, 2003.
[9] M. Y. Eltabakh, M. Ouzzani, M. A. Khalil, W. G. Aref and A. K. Elmagarmid,"Incremental mining for frequent patterns in evolving time series databases, "*Technical Report of Purdue University,* CSD TR#08-02, 2008.
[10] A. Erwin, R. P. Gopalan and N. R. Achuthan, "Efficient mining of high utility item sets from large datasets," in *Proc. of PAKDD 2008, LNAI 5012,* pp. 554-561.
[11] E. Georgii, L. Richter, U. Rucker and S. Kramer, "Analyzing microarray data using quantitative association rules," *Bioinformatics*, Vol. 21, pp. 123-129, 2005.