Maintaining Anti-Monotone Property for Generator with Weight and Its Mining Method

Bingzheng Wang
Zhengzhou University of Light Industry, Zhengzhou, China
Email: wbz@zzuli.edu.cn

Ran Liu
North China University of Water Conservancy and Electric Power, Zhengzhou, China
Email: liuran@ncwu.edu.cn

Abstract—Generator is a concise representation for frequent itemset. And it has the anti-monotone property as the frequent itemset does, which is an important property in real applications. But when itemsets are attached with weights to balance importances between themselves, the anti-monotone property of generator may not hold. Additionally generator with weight may become tough to be dealt with in many circumstances. In this paper, we adapt support weight calculation to generator definition under weight support framework through specific techniques. The anti-monotone property of generator with weight can be kept to facilitate mining works. A new method for mining generators with weights is proposed. It exploits depth-first mining strategy and prunes search space with little cost. Experimental results show that the proposed method runs properly and achieves good performance.

Index Terms—concise association rule, weighted generator, support-significant, algorithm

I. INTRODUCTION

Data mining research provides us effective tools to analyze huge amount of data automatically and promotes productivity. Association rule [1] expresses interesting relations among itemsets present in large number of transactions. It is an important knowledge representation in data mining area. Extensive studies [2-4] have been devoted into association rules mining. Traditional mining methods exploit an anti-monotone property of itemsets to facilitate mining works. But in real applications, traditional methods still confront the problem of balancing the significances of different itemsets while processing mining works smoothly. First, traditional association rule definition assumes items in transaction database have same significances, which is irrational in real circumstances. In the definition, the anti-monotone property of itemset holds, but the mining result may be biased with expectation of users. And the mining methods are flooded in the combinatorial explosion of insignificant relationships. Second, although there exists some concise representation for frequent itemset, but the anti-monotone property of itemsets for representation may not hold, which depress the ease of use in real applications.

Traditional association rule mining process has tow steps. One is to achieve the whole set of frequent itemsets, the other is to enumerating all association rules [1]. Let \( I \) be the set of all items and \( TDB=\{T_1, T_2, \ldots, T_n\} \) be a transaction database, where each transaction \( T_i \) is an itemset \( I \subseteq I \). The count of an itemset \( X \), denoted as \( \text{count}(X) \), is the number of transactions \( Y \) containing \( X \). Given a minimum support threshold \( \xi \), an itemset \( X \) is called a frequent itemset(pattern) if \( \text{count}(X)/|TDB| \geq \xi \). Frequent itemset keeps the anti-monotone property. From above, we can see that there exist \( 2^n \cdot 2 \) frequent subsets generated if a transaction with \( m \) items is frequent. For a not large database, there would be too many frequent patterns to be processed.

In real applications, items in same database may have different significances for different semantics in context. It is important to express the differences of items in database to get rational result. An alternative is to assign different weights to items to balance the differences between them. A method is proposed to express different significance of itemset with weight [5]. It adjusts support calculating method and keeps “weighted down closure property”. An Apriori-like method is proposed to achieve all the significant itemsets. But the mining result still comprises too many redundant itemsets as described in above paragraph. How do we keep the anti-monotone property of itemset with weight while representing them concisely?

Generator is an interesting alternative. It is a concise representation for frequent itemset and it can exactly derive all frequent itemsets and their supports. Generator keeps the anti-monotone property which is useful in many applications. But properties of traditional generator may not hold when being assigned different weights. And based on general frequent itemset mining methods, many proposed generator mining algorithms maintain candidate generators in main memory and filter non-generators, which conducts much redundant work and costs much time and space.

In this paper, we study the problem of how to derive concise representation for itemsets with weights while keeping the anti-monotone property. Weighted generator is defined under weighted support-significant framework.
We prove that weighted generator defined is a concise representation for itemsets, which can derive all original information. And the anti-monotone property of weighted generator can be kept when the “downward closure property” holds. Through exploiting efficient search space pruning methods, an effective approach for mining weighted generator is proposed. Through exploiting the depth-first mining strategy, exactly itemset checking can be conducted. Then duplicate itemset search space can be identified early and pruned. Redundant itemset checking works are avoided. Specific filtering process, weighted generator can be generated without the need of keeping the whole set of candidate generators in main memory at the same time. Experimental studies show that the method proposed in this paper leads to reasonable result and achieves good performance.

The remaining of the paper is organized as follow. In Section 2, we introduce definitions and terminology about weighted generator. In Section 3, we present weighted support calculating methods for weighted generators. In section 4, we introduce general enumerating and candidate checking approach for mining generators. Section 5 explores mining methods based on tree structure. In Section 6, performance study is presented. Then we conclude our work in Section 7.

II. DEFINITION AND TERMINOLOGY

Let \( I=\{i_1, i_2, ..., i_n\} \) be a set of items and \( TDB=\{T_1, T_2, ..., T_n\} \) be a transaction database, where \( T_i (1 \leq i \leq n) \) is a transaction which contains a set of items in \( I \). A set of items is called an itemset or a pattern. A transaction \( T \) is said to contain itemset \( X \) if and only if \( X \subseteq T \). The number of transactions in \( TDB \) that contain \( X \) is called the support count of \( X \), denoted as \( \text{count}(X) \), and the support of \( X \) is denoted as \( \text{support}(X) \), which equals to \( \text{count}(X)/|TDB| \), where \( |TDB| \) is the total number of transactions in \( TDB \).

Definition 1. Given a transaction database \( TDB \) and a minimum support threshold \( \xi \), an itemset \( X \) is called a frequent generator if both of the following conditions are true:

1) \( \text{support}(X) \geq \xi \);

2) \( \forall Y \subset X \) such that \( \text{support}(Y) > \text{support}(X) \).

From above definition, we can conclude that all frequent itemsets can be derived from all frequent generators mined in a specific transaction database.

Given a specific transaction database and support threshold \( \xi \), generators generation is to find the complete set of frequent generators.

From the above definition, we can have following lemmas:

Property 1. For an itemset \( Y \), \( \exists X \subseteq Y \), \( \text{support}(X) = \text{support}(Y) \), then \( Y \) can’t be a generator.

Proof. We can have the lemma directly from the definition of generator.

Property 2. For itemsets \( X, Y, Z \) (\( X \subseteq Y \subseteq Z \)), \( \text{support}(X) = \text{support}(Z) \), then \( \forall W \ (W \supseteq Y) \), \( W \) can’t be a generator.

Proof. From support definition of itemset and \( \text{support}(X) = \text{support}(Z) \), we have that every transaction containing \( X \) contains \( Y \) and \( Z \). For \( W \supseteq Y \), every transaction containing \( X \cup (W-Y) \) contains \( Y \cup (W-Y) \), then \( \text{support}(X \cup (W-Y)) = \text{support}(W) \). For \( X \subseteq Y \), then \( X \cup (W-Y) \subseteq Y \cup (W-Y) \) and \( X \cup (W-Y) \subset W \), from lemma 1, we have the conclusion.

Definition 2. Weighted item. Given a transaction database \( TDB \), and the item sets \( I=\{i_1, i_2, ..., i_n\} \) which appear in \( TDB \), we attach a value \( \text{weight}_{i_m} \) to each \( i_m \) representing its significance. Such an item is called weighted item.

Definition 3. Weighted itemset. Given an itemset \( X \), for each item in it has a weight, itemset \( X \) may take different significance beyond the other. We denote the significance of \( X \) \( \text{weight}(X) \), which varies according to the itemset \( X \).

Definition 4. Weighted support. Given an itemset \( X \) with items attached with weights, from above definition, we define the weighted support of \( X \) as \( \text{ws}(X) \).

To balance different significances of itemsets in mining process, weights and weighted supports of itemsets can’t be defined respectively. It can only be calculated from weights of items contained in itemset reasonably according to the need of specific application.

Definition 5. Given a transaction database \( TDB \) and weights for items with a minimum support threshold \( \xi \), an itemset \( X \) is a frequent weighted generator if both of the following conditions are true:

1) \( \text{ws}(X) \geq \xi \);

2) \( \forall Y \subseteq X \) such that \( \text{ws}(Y) > \text{ws}(X) \).

The problem of weighted generator mining is to find the complete set of frequent weighted generators in a given transaction database with respect to the given support threshold \( \xi \).

Definition 6. Given a transaction database \( TDB \), weights for items, a minimum support threshold \( \xi \) and a minimum confidence threshold \( \xi_2 \), an rule \( X \Rightarrow Y \) is a weighted concise association rule based on generator if both of the following conditions are true:

1) \( X, Y(Y \supseteq X) \) are frequent generators;

2) \( \text{ws}(Y)/\text{ws}(X) \geq \xi_2 \).

The problem of concise weighted association rule mining based on weighted generator is to find the complete set of concise association rule based on closed weighted generators in a given transaction database with respect to the given support threshold \( \xi_1 \) and confidence threshold \( \xi_2 \).

III. CALCULATING WEIGHTED SUPPORT WHILE KEEPING ANTI-MONOTONE PROPERTY FOR WEIGHTED GENERATOR

In order to keep the properties of generator with weight, we should properly define the weights of itemsets and give reasonable calculating methods. Additionally rational mining algorithm should be proposed to enumerate all weighted generators smoothly. Different calculating methods of itemset’s support weight may lead to different results, but properties of traditional generator may not hold. The core of the problem is how to design weighted support calculating method and enumerate...
transactions containing itemset \( X \) we can define weight as follow:

\[
ws(X) = \frac{\sum_{T_i \in X} weight(T_i)}{|T|} \quad (1)
\]

But how do we calculate \( weight(T_i) \) for each transaction \( T_i \) based on weights of items. For any \( T_i \in TDB \), we can define \( ws(T_i) \) as we will reasonably. Generally weighted support of itemset \( X \) is defined as follow:

\[
weight(T_i) = \frac{\sum_{I \in T_i} weight(I)}{|T_i|} \quad (2)
\]

**Lemma 1.** (downward closure property). For an itemset \( X \), \( \exists Y \supseteq X \), \( ws(Y) \leq ws(X) \).

**Lemma 2.** For an itemset \( Y \), \( \exists Z \subseteq Y \), \( ws(Y) = ws(Z) \), then \( Y \) can’t be a weighted generator.

**Proof.** From definition 5, we can have the lemma.

**Lemma 3.** For itemsets \( X, Y, Z \), \( X \subseteq Y \subseteq Z \), \( ws(X) = ws(Y) \), then \( Z \) can’t be a weighted generator.

**Proof.** Let \( TS(X) = \{ T_i \subseteq TDB \} \), \( T \supseteq X \) be the set of transactions containing itemset \( X \), then we can have that:

\[
ws(X) = \sum_{T_i \in TS(X)} weight(T_i) \quad (3)
\]

For \( X \subseteq Y \subseteq Z \), we have that \( TS(X) \supseteq TS(Y) \supseteq TS(Z) \), then

\[
\sum_{T_i \in TS(X)} weight(T_i) \geq \sum_{T_i \in TS(Y)} weight(T_i) \geq \sum_{T_i \in TS(Z)} weight(T_i) \quad (4)
\]

From (3) and (4), we can have

\[
ws(Y) \geq \sum_{T_i \in TS(Y)} weight(T_i) \quad (5)
\]

\[
ws(Z) \geq \sum_{T_i \in TS(Z)} weight(T_i)
\]

Then \( ws(X) \geq ws(Y) \geq ws(Z) \), and \( ws(X) = ws(Y) \), then we have \( TS(X) = TS(Y) \supseteq TS(Z) \), then we can have \( TS(X \cup \{i\}) = TS(Y \cup \{i\}) \) from above, we can get

\[
ws(X \cup \{i\}) = ws(Y \cup \{i\}) = ws(Z)
\]

From lemma 2 and \( Z \subseteq X \cup \{i\} \), we have the conclusion.

**Corollary 1.** (anti-monotone property). For a weighted generator \( Z \), then \( \forall Y (Y \subseteq Z) \), \( Y \) is a weighted generator.

**Proof.** We assume that

\[
\exists X (X \subseteq Z) \text{ and } X \text{ is not a weighted generator.}
\]

From the definition of weighted generator, we have

\[
\exists W (W \subseteq X \text{ and } ws(W) = ws(X))
\]

From lemma 3 and \( X \subseteq Z \), we have that \( Z \) can’t be a weighted generator, which is opposite to the fact in corollary. Then we have the conclusion.

Based on above definitions and lemmas, the anti-monotone property holds on weighted generator. Through exploiting properties of weighted generator, we can derive all weighted generators in specific mining framework.

**IV. ENUMERATING WEIGHTED GENERATORS BY ADDING ITEMS TO ITEMSETS**

Generally itemset search space is explored in specific sequence of items, which facilitates data projection from original database. Itemsets with more items are always generated first in the process. But from the properties of generator, we can’t tell a longer itemset is generator or not when its subsets are not checked. In this section, a specific exactly itemset checking approach is proposed. Candidate weighted generators are explored in depth-first manner. Itemset search space can be pruned early with little cost, which simplifies the mining process.

**A. Conditional Weighted Generator Set**

We assume that there is a partial order on itemset \( I \), denoted as \( < \), which is the ascending weighted support order of items on \( I \).

**Definition 8.** Given the transaction database \( TDB \), and \( I \) contains all items appearing in it, the conditional weighted generator set of \( \{i_1, i_2, \ldots, i_k\} \) is denoted as \( CWG(I_1, i_2, \ldots, i_k) \), while following conditions are satisfied:

1) \( i_k < i_{k-1} < \ldots < i_1 \);

2) For \( \forall X \subseteq CWG(i_1, i_2, \ldots, i_k) \), such that \( \forall I \subseteq X, i_k < I \), and \( \exists T \subseteq TDB \) such that \( X \cup \{i_1, i_2, \ldots, i_k\} = \emptyset \);

3) \( CWG() \) is the conditional weighted generator set of \( \emptyset \).

**B. Transferring Weights via Conditional Transactions**

The weighted supports of weighted itemset and weighted transaction are strictly defined in above subsection. In the mining process, original weight
supports of transaction are transferred to subsets projected. We needn’t calculate real weight support for subsets projected from original database.

First, we initialize \( CWG() \). We calculate all weighted supports of all items and transaction. Then items whose weighted support is less than \( \xi \) are eliminated from all transactions. And itemsets left with original weighted supports are inserted into \( CWG() \). If there is a same itemset, then eliminate one and double weighted support of the other.

Second, for every frequent item \( i_m \) appearing in \( CWG(i_1,i_2,...,i_k) \), we do as follows:

For every itemset \( i \) containing \( i_m \) in \( CWG(i_1,i_2,...,i_k) \), we collect all the frequent item \( i \) where \( i_m < i \) as a new itemset to be inserted into \( CWG(i_1,i_2,...,i_k,i_m) \). Then we set \( CWG(i_1,i_2,...,i_k) \) to \( \emptyset \). With the mining process recursively going on, all the conditional weighted generator sets are initialized.

C. Eliminating Exactly Same Candidate Generators

For two itemsets \( s_1 \) in \( CWG(i_1,i_2,...,i_m,i_m) \) and \( s_2 \) in \( CWG(i_1,i_2,...,i_m,i_m) \) \((i_m < i \) if \( s_1 \) equals \( s_2 \) and their local support weights are same, itemset \( s_1 \cup \{i_1,i_2,...,i_m\} \) can’t be a weighted generator, then we remove itemset \( s_1 \) from \( CWG(i_1,i_2,...,i_m,i_m) \). We say that we do \( i_1 \) as follows: for every item \( i_m \) in \( CWG(i_1,i_2,...,i_m,i_m) \), we assume all \( CWG(i_1,i_2,...,i_m,i_m) \) \( (i_m < i \) have been dealt with, then for every \( i_m \), we do \( CandidateChecking(i_1,i_2,...,i_m,i_m) \). After each item \( i_m \) in \( CWG(i_1,i_2,...,i_m,i_m) \) have been dealt with, we do as follows: for every item \( i_m \) in \( CWG(i_1,i_2,...,i_m,i_m) \), item \( i_m \) is inserted into every element of \( CWG(i_1,i_2,...,i_m,i_m) \), then elements of \( CWG(i_1,i_2,...,i_m,i_m) \) are removed to \( CWG(i_1,i_2,...,i_m,i_m) \).

D. Enumerating Weighted Generators Framework

Then we give the general framework for deriving weighted generators. For a given database \( TDB \) and minimum support threshold \( \xi \) all the weighted generators can be get from \( CWG() \) by calling procedure \( EnumerateWG() \) as shown in Fig. 1.

In the procedure, the transaction projecting and candidate checking methods are conducted recursively, which are described in previous subsection.

Procedure: \( EnumerateWG(\alpha) \) (Here \( \alpha \) represents \( i_1,i_2,...,i_n \), \((i_m < i_2 < ... < i_m)\))

1. recalculate all weighted support of item \( i \) in the new transactions projected in \( CWG(\alpha) \);
2. if(\( \forall i \in \alpha \) \( \forall i \geq \xi \) )
3. set \( CWG(\alpha) = \{\emptyset\} \);
4. return;
5. }
6. for item \( i = \text{min to max contained in } CWG(\alpha) \) do }
7. if(\( \forall \alpha \in \alpha \) \( \forall i \geq \xi \) )
8. project sub-transactions \( \alpha ' \) in \( CWG(\alpha) \) to \( CWG(\alpha, i) \);
9. \( EnumerateWG(\alpha, i) \);
10. for \( i = \text{min to } i_\alpha \) then \( CandidateChecking(\{i_1,i_2,...,i_k\},i_\alpha) \);
11. ]]
12. if(there is an item \( i \) in \( CWG(\alpha) \) and \( \forall i \in \alpha \) \( \forall i \geq \xi \) )
13. set \( CWG(\alpha, i) = \emptyset \);
14. else set \( CWG(\alpha, i) = \emptyset \);

V. PATH PROJECTION: MINING WEIGHTED GENERATORS

We realize the mining works based on a tree-like data structure, which is called weighted support tree (\( WST \)). We call the weighted support tree \( WST(\alpha) \) built for \( CWG(\alpha) \). Through exploiting enumerating weighted generators framework, weighted supports are projected along the tree links. And weighted supports of itemsets can be calculated correctly. Based on tree structure, suffix links are formed by linking nodes having been mined without additional space. Additionally several optimizing methods are proposed, which can detect duplicate search space early and avoid many redundant checking works.

Suffix Weight Projection Optimization: Links of tree represent correlations between items contained in it. In \( CWG(\alpha) \), when an item \( i_m \) is concerned, relationship between item \( i_m \) and smaller items always be ignored. But the relationship still be retain in tree links. We project weights of \( i_m \) on smaller items than \( i_m \) through adjusting its descendant linked branches. After all the descendant branches are combined, only the items with smaller weight than that of item \( i_m \) are omitted.

Suffix Optimization: In the mining process, the larger items are always considered first in depth-first manner. From the suffix weight projection optimization and lemma 3, if weighted support of item \( i_m \) is equal to that of \( \{i_1,i_m\} \) in \( WST(\alpha) \), \( \forall i \in CWG(\alpha,i_m,i_m) \), \( is \cup \{i,i_m\} \) can’t be weighted generator. \( CWG(\alpha,i_m,i_m) \) can be set \( \emptyset \).

Following theorem ensures the framework can mine all the weighted closed itemsets correctly.

Theorem 1. Procedure \( EnumerateWG(\alpha) \) derives all weighted generators when it is called.

Proof. First, every weighted generator can be included in \( CWG() \). From line 7 in the procedure, we can see that each possible weighted generator \( cwg \) can be enumerated. Line 8 ensures all the information about \( cwg \) is inherited correctly from original database. Line 15 and line 16 ensure that itemset \( cwg \) can be formed. All the weighted generators can be get from \( CWG() \) after \( EnumerateWG(\alpha) \) is executed.

Second, any itemset in \( CWG() \) is generator after \( EnumerateWG(\alpha) \) is executed.

For an itemset \( Y = \{i_1,i_2,...,i_l\} \) \( (i_1 < i_2 < ... < i_l) \) is not a weighted generator, with lemma 2, there must be an subset \( X \) of \( Y \) that \( ws(X) = ws(Y) \) and \( |X| = n-1 \). There there must be an item \( i_\alpha \in Y \), that \( X = \{i_1,i_2,...,i_\alpha,...,i_l\} \) \( (i_1 < i_2 < ... < i_\alpha < ... < i_l) \) is included in \( CWG(\alpha) \). When \( EnumerateWG(\alpha,i_1,i_2,...,i_{l-1}) \) is called and \( CandidateChecking(\{i_1,i_2,...,i_{l-1}\},i_l) \) is executed, candidate generator \( \{i_1,i_2,...,i_{l-1}\} \) is eliminated in \( CWG(\alpha,i_1,i_2,...,i_l) \) after deeper recursive procedure are completed. Then itemset \( X \) can not be included in final \( CWG() \).
Then we needn’t conduct \textit{CandidateChecking}\{(\{a\},i_{a},i_{b})\} checking.

Based on tree structure and above optimizations, we can realize enumerating weighted generator framework. An example is conducted as follow.

\textbf{Example 1.} Given the transaction database $TDB$ with the item weight settings, the minimum weighted support threshold $\alpha = 0.2$ as in Fig. 2. The items’ supports and weighted supports are calculated as in the figure. Item $f$ is omitted for its weighted support is below $\alpha$. The ascending support order on items is $f$\textless{}$a$\textless{}$d$\textless{}$e$\textless{}$c$\textless{}$b$. Weighted generators: \textit{links} of item $e$ in generator $\{e,c\}$. But there is common item $d$ in suffix \textit{support} isn’t less than that of their ancestors. Those nodes having been mined. Nodes in suffix \textit{links} are dashed. And \textit{weighted generator} \textit{support} when item $e$ is concerned:

\begin{tabular}{|c|c|c|c|c|}
\hline
item & weight & \textit{TID} & items & ws \\
\hline
$a$ & 0.4 & 100 & b, d, e & 0.9667 \\
$b$ & 0.6 & 200 & a, b, c, d, e & 0.86 \\
$c$ & 1 & 300 & a, b & 0.5 \\
$d$ & 1.1 & 400 & b, c & 0.8 \\
$e$ & 1.2 & 500 & b, c, d & 0.9 \\
$f$ & 1.3 & 600 & c, e & 0.95 \\
$g$ & 1.4 & 700 & c & 1.1 \\
$\text{all}$ & 1.5 & 800 & a, c, d, e & 0.925 \\
\hline
\end{tabular}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig2.png}
\caption{Transaction database $TDB$ and weight settings.}
\end{figure}

The created tree structure $WST()$ for CWG() is shown in Fig. 3.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig3.png}
\caption{data structure $WST()$}
\end{figure}

During the mining process, original transactions are projected to build weighted support tree. For there are no subsets with same weighted support, all itemsets with one frequent item are generated as weighted generators. In $WST()$ as shown in Fig. 4, suffix links comprise nodes having been mined. Nodes in suffix links are dashed. And nodes in suffix links are omitted whose weighted supports is less than that of their ancestors. Those nodes are dotted. When item $e$ is concerned, all of the suffix links don’t contain common item, then we get conditional generator $\{\{e\},c\}$. But there is common item $d$ in suffix links of item $e$ in $CWG(a)$, then $ws(\{a,d,e\}) = ws(\{a,e\})$, itemset $\{a,d,e\}$ can’t be generator.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig4.png}
\caption{data structure $WST()$ when item $e$ is concerned}
\end{figure}

As mining process continues, we can derive following weighted generators:

\{a\}, \{d\}, \{e\}, \{c\}, \{b\}, \{a,d\}:0.2549, \{a,e\}:0.2549, \{a,c\}:0.2549, \{d,e\}:0.3930, \{c,b\}:0.2609, \{c,b\}:0.3656, etc.

Through being attached with different weights, items with larger weights are more likely derived than those with smaller weights. Mining result is more reasonable in real application than traditional methods.

\textbf{VI. EXPERIMENTAL RESULTS}

In this section, experiments on public data sets are conducted. The proposed algorithm is implemented in C++. Algorithm A-Close mines all closed itemsets, but it derives generators first, which consume most of mining time. And algorithm CLOSET+ [9] mines all closed itemsets, but it is much more efficient than A-Close. Here we compare the proposed algorithm with CLOSET+.

Tested datasets include mushroom, chess, pumsb, kosarak, T104D100K, which can be download from UCI Repository [10] or Frequent Itemset Mining Implementation Repository. Here we just demonstrate the results on mushroom and T40110D100K. The items are set with specific weights. Mushroom dataset contains all kinds of mushroom information with different properties. The target platform is a Lenovo PC equipped with 2.6G clock rate CPU and 1024M main memory. The operation system is WindowsXP Professional.

The proposed algorithm adopts effective method for filtering candidate weighted generators, and it doesn’t need to keep the whole set of candidate sets in main memory at the same time. It achieves good results as shown in following figures.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig5.png}
\caption{Runtime comparison on dataset mushroom}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig6.png}
\caption{Runtime comparison on dataset T40110D100K}
\end{figure}
From the experiments, when an item is attached higher weight, the according properties are more likely mined. The mushroom is more likely poisonous containing items with higher weight. And the number of all weighted generators may vary under specific weights setting.

For the number of weighted generator derived doesn’t vary too much under general weights setting, the space consumption is no more than those of general ones. And through exploiting specific mining technique, the proposed method runs smoothly.

VII. CONCLUSION
In this paper, we proposed a method to balance significances of generators with weights while keeping the anti-monotone property. We find that it is possible to complete mining works under weighted support-the anti-monotone property. We find that it is possible to calculate significances of generators with weights while keeping scalability.

Experimental results show that it leads to reasonable results and has good time scalability.

ACKNOWLEDGMENT
This work is supported by the Key Technologies R&D Program of Henan Province (No. 092102210108) and the Heng Yang Joint Funds of The Hunan Provincial Natural Sciences Foundation of China (No. 11JJ9010) and the Doctoral Scientific Research Foundation of Zhengzhou University of Light Industry (No. 2008BSJJ010).

REFERENCES
[2] Jiawei Han, Jian Pei, Yiwen Yin, “Mining frequent patterns without candidate generation”, In Proceeding(s) of the ACM International Conference on Management Of Data, pp.1-12, 2000.

Bingzheng Wang was born in Puyang, Henan, P.R. China, on December 11, 1978. He received doctoral degree of engineering in computer science from Nankai University, Tianjin, P.R. China in 2007. He is currently lecturer of computer science at Zhengzhou University of Light Industry, Zhengzhou, Henan, P.R. China. His research interests include data mining, intelligent information processing and semantic web.

Ran Liu was born in Xinyang city, Henan, P.R. China in December 23, 1979. He received master’s degree in engineering from North China University of Water Resources and Electric Power, Zhengzhou, Henan, P.R. China. He is lecturer of computer science, and the director of multimedia technology department, the Software Institute, North China University of Water Resources and Electric Power, Henan, P.R. China. He has published more than 10 papers in Journal of Science and Technology. His research interests include data mining, distributed database and multimedia technology.