WTMaxMiner: Efficient Mining of Maximal Frequent Patterns Based on Weighted Directed Graph Traversals*

Runian GENG[1,2], Xiangjun DONG[2]
2. School of Information Science and Technology
Shandong Institute of Light Industry
Jinan, China
gengrnn@163.com, d-xj@163.com,

Ping ZHANG[1], Wenbo XU[1]
1. School of Information Technology
Jiangnan University
Wuxi, China
zpshh@126.com, xwb_sytu@hotmail.com

***Abstract*—Frequent itemset mining for traversal patterns have been found useful in several applications. However, (closed) frequent mining can generate huge and redundant patterns, and traditional model of traversal patterns mining considered only un-weighted traversals. In this paper, a transformable model between EWDG (Edge-Weighted Directed Graph) and VWDG (Vertex-Weighted Directed Graph) is proposed. Based on the model, an effective algorithm, called WTMaxMiner (Weighted Traversals-based Maximal Frequent Patterns Miner), is developed to discover maximal weighted frequent patterns from weighted traversals on directed graph. Experimental comparison results with previous work on synthetic data show that the algorithm has a good performance and scalable property to the problem of mining maximal frequent patterns based on weighted graph traversals.**

***Keywords*—data mining, traversal patterns, maximal weighted frequent pattern mining, closed pattern mining**

I. INTRODUCTION

Data mining on graph traversals have been an active research field during recent years. Graph and traversal on it are widely used to model several classes of data in real world. One example for this is WWW. The structure of Web site can be modeled as a graph in which the vertices represent Web pages, and the edges represent hyperlinks between the pages. That is, user's navigation on the Web site can be modeled as traversals on the graph. Capturing user access patterns in such environments is referred to as mining traversal patterns [1]. Traditional model of traversal patterns mining hardly considered weighted traversals on the graph [1][2][3]. To reflect importance difference of Web sites, we can assign a weight to each site.

Since the frequent itemsets mining problem (**FIM**) was first addressed [4], frequent itemsets mining in large database has become an important problem. The drawback of mining all frequent itemsets is that if there is a large frequent itemset with size l, then almost all 2^l candidate subsets of the items might be generated. So the large length of frequent itemset leads to no feasible of **FIM**. Smaller alternatives to **FIM** that still contain compact yet lossless representation of the frequent itemsets is frequent closed itemset mining (**FCIM**)[5][6][7]. It is the task of discovering frequent itemsets whose support counts are different than those of their supersets. However, when the frequent patterns are long (more 15 to 20 items), **FIM** and **FCIM** become very large and most traditional methods count too many itemsets to be feasible [8]. A frequent itemset is called maximal frequent itemset (**MFI**) if it has no frequent superset. It is straightforward to see that the following relationship holds: **MFI** $\subseteq$ **FCI** $\subseteq$ **FI**. Since frequent itemsets are upward closed, it is sufficient to discover only all maximal frequent itemsets (**MFI**).

In this paper, we extend previous works by attaching weights to the traversals and propose a new effective & scalable algorithm called *WTMaxMiner* (*W*eighted *T*raversals-based *M*aximal *F*requent *P*atterns *Miner*) to discover maximal frequent patterns from weighed traversals on graph, which exploits a *divide-and-conquer* approach in a *bottom-up* manner and incorporates the maximal property with weight constrains to reduce effectively search space and extracts succinct and lossless patterns from weighted graph traversal TDB. To our knowledge, ours is the first work to considering maximal frequent patterns mining from directed graph with weight constraints.

The organization of this paper is as follows. Section Ⅱ reviews previous works. The related definitions and notions of problem are given in Section Ⅲ. Section Ⅳ proposes the algorithm named *WTMaxMiner*. Experimental research and the performance analysis of algorithm are reported in Section Ⅴ. Finally, Section Ⅵ gives the conclusion as well as future research works.

II. RELATED WORKS

The main stream of data mining related to our work, can be divided into three categories, i.e. the traversal pattern mining, the maximal frequent pattern mining and the (weight) constraint based pattern mining. For the traversal pattern

*This work was supported in part by the Natural Science Fund of Shandong Province (No.Y2007G25), the Excellent Young Scientist Foundation of Shandong Province, China (No.2006BS01017) and the Scientific Research Development Project of Shandong Provincial Education Department, China (No. J06N06).

mining, there have been few works. Chen et al. [1] proposed two algorithms—FS and SS about traversal pattern mining. However, they did not consider graph structure on which the traversals occur. Nanopoulos et al. [2] proposed three algorithms which considered the graph structure. However, the above works dealt with the mining of un-weighted traversal patterns.

In the last several years, extensive studies have proposed fast algorithms for mining maximal frequent itemsets, such as Mafia [8], MaxMiner [9], Genmax [10], CfpMfi[11] and Fpmax[12]. CfpMfi and Fpmax are based on the pattern growth method [13] which has a high performance.

For the (weight) constraint mining, most of previous works are related to the mining of association rules and frequent itemsets [14][15][16]. Recently, constraint-based frequent pattern mining algorithms [17] based on the pattern growth method were suggested. Although the above works take the notion of weight into account as examined in this paper, they only concerned on the mining from items, but not from traversals.

III. Problem Statement

A. Correlative Definitions of Mining Traversal Pattern

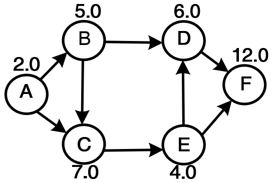
Definition 1 (Weighed Directed Graph) A *WDG* (*W*eighted *D*irected *G*raph), denotes as *G*, is a finite set of vertices and edges, in which each edge joins one ordered pair of vertices, and each vertex or edge is associated with a weight value.

By definition 1, we know that there should be two kinds of *WDG*s. One is *VWDG* (*V*ertex-*WDG*) which assigns weights to each vertex in the graph, and the other is *EWDG* (*E*dge-*WDG*) which assigns weights to each edge (We will know they are essentially equivalent in next section,. So, we only study the former in this paper). For example, Fig.1 (a) is *VWDG G* which has 6 vertices and 8 edges. Next, we will know they are essentially equivalent.

Definition 2 (Traversal on Graph) A traversal on graph is a sequence of consecutive vertices along a sequence of edges on a *G*.

Clearly, a traversal can be regarded as a pattern. To easily consider, we may assume that every traversal has no repeated vertices. The length of a traversal is the number of vertices in the traversal. A traversal database *T* is a set of traversals. Since there are two *WDG*, then it must exit that there are two types of traversals-traversals on *VWDG* and traversals on *EWDG*. Figure 2(a) and (b) respectively describe them, and (c) is the combination of two cases.

Definition 3 (Subtraversal) A subtraversal is any subsequence of consecutive vertices in a traversal.



(a) Example of VWDG

TID	Traversal
1	<A, B>
2	<B, C, E, F>
3	<A, C>
4	<B, C, E>
5	<A>
6	<A, C, E, D>

(b) Traversal database T

Figure 1. An example of VWDG & traversal database T

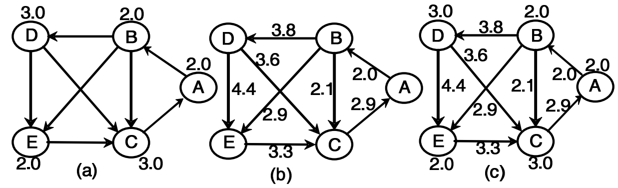


Figure 2. Three cases of assigning weights to traversals

Definition 4 (Sup_count & support.) The support count of a pattern *P,* denoted as *sup_count* (*P*), is the number of traversals containing the pattern. The support of a pattern *P*, *support* (*P*), is the fraction of traversals containing the pattern *P*, denoted as: (|*T*| be the number of traversals.)

$$support(P)=\frac{sup\_count(P)}{|T|} \quad . \tag{1}$$

Definition 5 (Maximal frequent traversal pattern) Given a threshold minimum support *min_sup*, a traversal pattern Y is a maximal frequent traversal pattern if support (*Y*) ≥*min_sup* and there exists no proper superset $Y' \supset Y$ such that support (Y') ≥min_sup.

Definition 6 (Weighted Pattern) A weighted pattern is a set of items each which has a weight.

Definition 7 (Weight of Pattern) The weight of pattern is an average value of weights of all items in it.

Given a weighted pattern $P=<p_1,p_2,....,p_k>$, the weight of each item in *P*, denoted as $w(p_i)$ $(i=1,2,...,k)$, then the weight of *P* is represented as follows.

$$Weight(P)=\frac{\sum_{i=1}^{k} w(p_i)}{|P|} \tag{2}$$

Definition 8 (Weighted Support) The weighted support of a pattern P, called wsupport (*P*), is

$$wsupport(P)=Weight(P)(support(P)) \tag{3}$$

A *P* is said to be weighted frequent when its weighted support is no less than a given minimum weighted support threshold called *wminsup*, i.e.,

$$wsupport(P)\geq minwsup \tag{4}$$

Thus, the problem concerned in this paper is stated as follows. Given a weighted directed graph *G* and a set of path traversals on the graph — traversal database *T*, we find all maximal frequent patterns with weight constraint in *T*. However, the weight constraint is neither the anti-monotone nor the monotone constraint. So we cannot directly use the *anti-monotone* property to prune weighted infrequent patterns.

B. Model of transforming EWDG into VWDG

Essentially, the two *WDG*s cases can be reduced to one case. There we reduce the two cases to one case— assigning weight to vertices. The reason why we can reduce is that two nodes with a weighted edge in an *EWDG* can be thought as a node with same weight value in the corresponding *VWDG*, and the edges between vertices in corresponding *VWDG* have no

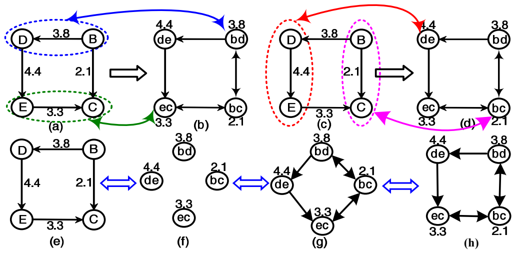


Figure 3. The change process from EWDG to VWDG

weight value, and their linking directions refer to the source *EWDG*.

Figure 3 describes this change method: Fig.3 (a), (c) and (e) are *EWDG*s, and Fig.3(b),(d),(g) and (h)are *VWDG*s. In Fig.3(b) or (d), the nodes named '*bd*', '*de*', '*ec*' and '*bc*' respectively represents the corresponding nodes with directed edges named <*B,D*>,<*D,E*>,<*E,C*> and <*B,C*> in Fig.3(a) or (c). Figure 3(e), (f), (g) and (h) truly describe how to change *EWDG* into *VWDG*. The transformable process undergoes three phases.

Phase 1: Fields transforming phase.

In this phase, all nodes and the edge between each pair of nodes in EWDG are converted to some corresponding nodes with the same weight value as that of source edge. For example, the nodes named '*bd*:3.8', '*de*:4.4', '*ec*:3.3' and '*bc*:2.1' in Fig.3(f) respectively represents the corresponding nodes with weighted directed edges named <*B,D*>:3.8, <*D,E*>:4.4, <*E,C*>:3.3 and <*B,C*>:2.1 in Fig.3(e). There *x*: *number* means that *x* is the name of node or edge and *number* is the corresponding weight value.

Phase 2: Generating edges' direction between nodes generated from phase 1 in new birth *VWDG.*

Each edge's direction in new birth *VWDG* is based on the balance of indegree and outdegree of fields in source *EWDG*, e.g., for node '*de*' in Fig.3(g), because its corresponding field in (e) is joined by two directed edges named <*B,D*> (i.e. indegree) and <*E,C*> (i.e. outdegree), so there are two edges named <'*bd*'→'*de*'> and <'*de*'→'*ec*'> in (g), the other edges' direction are similar to the above method. That is to say that it is balance between indegree and outdegree of source field in *EWDG* by which we decide each new edge's direction in the new *VWDG*.

Phase 3: Shape rotation phase.

In this phase, we rotate the new birth *VWDG* generated by phase 2 to a shape easy to distinguish. Clearly, this phase can be omitted according the actual situation.

Thus, we construct a union between *EWDG* and *VWDG*. This union is convenient to reduce pattern mining problem on weighted graph. In practice, the fields in Fig. 3(a) and (c) may represent subnet of any size under the Web environments.

C. Revised Weighted Support

We know that the weight constraint is neither the anti-monotone nor the monotone constraint. Can we use *anti-monotone* property to mine weighted frequent pattern mining? The answer is 'yes'. We know that the weights of items on graph containing n nodes , denoted as w_i ($i=1,2,\dots,n$), must satisfy: $\min(W) \le w_i \le \max(W)$, there, $W=\{w_1,w_2,\dots,w_n\}$. To let weighted patterns satisfy the *anti-monotone* property (i.e., if *wsupport* (P) ≤ *minwsup* ⇒ *wsupport* (P') ≤ minwsup), we revise *weight* (p_i) as two following representations.

$$weight(p_i)=min(W) \quad \text{or} \quad weight(p_i)=max(W)$$

That is, the weight of pattern is revised a*s:*

$$weight(P)=\frac{\sum_{i=1}^{k} w(p_i)}{|P|}=\frac{\sum_{i=1}^{k} min(W)}{|P|}\,. \tag{5}$$

or

$$weight(P)=\frac{\sum_{i=1}^{k} w(p_i)}{|P|}=\frac{\sum_{i=1}^{k} max(W)}{|P|}\,. \tag{6}$$

Thus, the anti-monotone property can be used in mining weighted frequent patterns since w*support* (P') ≤ w*support* (P). However, if we adopt (5), we could prune some patterns which should have been weighted frequent to lead to incorrect mining results. Avoiding this flaw, we adopt (6) to compute revised weight of the pattern Note, the weight support value computed by this is only an approximate value, therefore, in final step, we should check if the pattern is really weighted frequent pattern with his real weight value, by the condition:

$$\frac{\sum_{i=1}^{k} weight(p_i)}{k}\left(support(P_k)\right)\ge minwsup\,.$$

Data mining from itemsets with the *anti-monotone* property can adopt the pattern growth strategy.

D. Order of Joining the Closure and Constraint

There are two ways to joining the closure and constrains with frequent patterns mining [18].

$$\text{I}.\ Cl\left(FTh_D\left(sup_D(X)\ge min\,sup\right)\right)\cap FTh_D(C_{am}). \tag{7}$$

$$\text{II}.\ Cl\left(FTh_D\left(\left(sup_D(X)\ge min\,sup\right)\wedge C_{am}\right)\right). \tag{8}$$

There, C_{am} is an anti-monotone constraint, as to weight constraint, it is revised weight constraints. In Ⅰ, frequent patterns are first tested whether the patterns are closed frequent patterns, and then for the closed patterns weighted constraints are applied to discover weighted closed frequent patterns. Ⅱ is one the way round: the weight constraint is first mined and then the closures of the weighted frequent patterns are computed. From [18], we already know only way Ⅱ not can lead to information loss. In addition, because a maximal frequent pattern must be a closed frequent pattern (i.e. **MFI⊆CFI**), so we adapt way Ⅱ in our problem, i.e., we first extract the weighted frequent patterns then check if they are maximal . We call these patterns extracted by way Ⅱ as maximal weighted frequent patterns.

IV. Mining Maximal Frequent Patterns from WDG Traversals Using Weighted FP-trees

As we described above, revised weighted setting has an anti-monotone property. We also know that the way of weight constraints being first mined and then the maximal property of the weighted frequent patterns being checked can mine the correct information. We devise an efficient and scalable algorithm, called WTMaxMiner which is based on a weighted FP-tree

1. First, scan the TDB once to identify the set of weighted frequent items and their corresponding support counts, and sort them by ***sup_count*** descending to form the set of ***f-list***.
2. Second, weighted infrequent items whose weighted supports are less than a *minwsup* are removed .
3. Next, those remaining weighted frequent items in each transaction are sorted by the ***sup_count*** descending order.
4. Lastly, scan the new TDB again and the sorted weighted frequent items in each transaction are sequentially inserted in a global weighted FP-tree along a path from the root to the corresponding node. If a new node is inserted in the path, the 'sup-count' field of the node is set to one. If an existing node in the FP-tree is met, the 'sup-count' of node in the FP-tree is incremented by 1.
5. Stop when all the transactions are insert to the global weighted FP-tree.

Figure 4. Process of constructing the weighted FP-tree

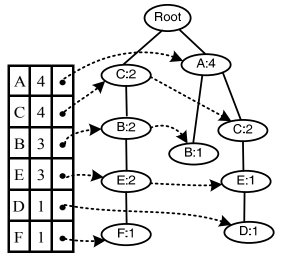


Figure 5. Global weighted FP-tree

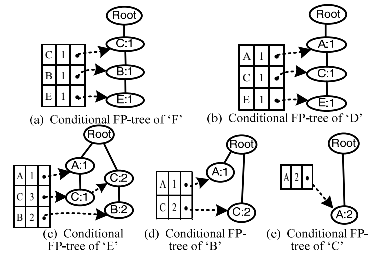


Figure 6. Conditional weighted FP-trees

and exploits a divide-and-conquer strategy with a pattern growth method, to mine the maximal frequent patterns with weight constraint from directed graph.

A. *Weighted FP-tree Construction*

FP-tree is a compact representation of all relevant frequency information in a database [13]. To get a high performance, we adopt a modified weighted FP-tree as a compression technique based on pattern growth method. Each node in the weighted FP-tree has four fields: 'item-name', 'sup_count' 'weight' and 'node-link'. Additionally, for each weighted FP-tree, there is a header table which has there fields: 'item-id', 'support' and a 'headpointer' to the first node in the FP-tree with the item-id. Initially, in our approach, a modified FP-tree has only a root node. The weighted FP-trees in our algorithm are constructed as shown as Fig. 4. Through this way, compressed data from the original TDB is stored in the FP-tree. For the transaction database shown in Fig.1 with support threshold *minwsup*=1.5 (max(W) =12). Fig. 5 presents the corresponding global weighted FP-tree and a header table.

B. *Search Space Pruning Techniques and Necessary Checkings in WTMaxMiner*

Because we use a revised weight support, so we only generates approximate weighted frequent patters and can not assure that they are real maximal weighted frequent patterns on directed graph G. From [7], we already know that newly found frequent patterns cannot be included by any later found frequent patterns based on the *divide-and-conquer* approach. Therefore, we only need to do subset-checking in order to assure a newly found weighted frequent pattern is maximal. That is, a newly found weighted frequent pattern is compared with already generated maximal weighted patterns to know if the newly found pattern is a subset of already found maximal weighted patterns. In our algorithm, we use MFI-tree [12] to store so far found maximal weighted frequent patterns. Every newly found maximal frequent pattern named *Sc* is compared with maximal weighted frequent patterns in the MFI-tree. If there is no superset of the pattern *Sc* in MFI-tree, the pattern *Sc* is inserted into the MFI-tree. In addition, because our all works are all based on graph traversals, we must check if the result mined patterns are included in G and if the order of vertices in result mined patterns is corrected order referring to G.

In summary, through the process of mining, we must do five ordered necessary checkings: (1) Extract the ***candidate approximate*** weighted frequent patterns P (which is the local candidate weighted frequent pattern) from weighted FP-tree, by checking if $wsurpport\ (P) = support\ (P)*\max\ (W) \geq minwsup$. (2) Toward newly found (local) candidate approximate weighted frequent patterns, we check their ***real*** *wsupport*, and those whose real $wsupport \geq minwsup$ are remained, the others are removed. (3) Toward local really weighted frequent patterns, we check if they are ***included*** in the path of G according to the original TDB T on G. The checking contents consist of two aspects. One is to check the order of vertices in it, and the other is to check the *sup_count* of it with correct order vertices. The patterns included in the path of G but the order of vertices in it is not correct, must be revised the correct vertices order and corresponding *sup_count* referring to the path on G. If corresponding *sup_count* of some pattern P is changed, we must check if P satisfies the condition: $wsurpport\ (P) = support\ (P)*Weight\ (P) \geq minwsup$. Those patterns whose real *wsupport* is less than *minwsup* are pruned. The remained patterns are ***local really*** weighted frequent patterns included in path of G. The reason of doing the above checking is that the traversal on G is a ordered sequence of vertices, but the method of minding weighted frequent patterns by weighted FP-tree approach break down the original vertices order of weighted frequent patterns. (4) Toward local really weighted frequent patterns, we do local maximal property checking to remove he non-maximal local patterns and remain ***local*** maximal weighted patterns. (5) Lastly, for local maximal weighted patterns, compare them with already generated maximal weighted patterns in MFI-tree to check if it satisfies ***global*** maximal property. Those frequent patterns which satisfy global maximal weighted property are inserted into MFI-tree.

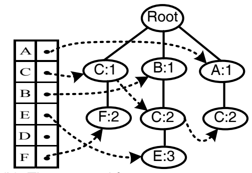
C. *Bottom- up Traversal of Weighted FP-tree with Divide-and-conquer Strategy*

With the global weighted FP-tree, *WTMaxMiner* mines maximal weighted frequent patterns by adapting the *divide-and-conquer* approach. For the traversal transaction database shown in Fig. 1(b) (*minwsup=1.5*), it divides mining the FP-tree into mining smaller FP-trees with *bottom up* traversal of the FP-tree, and mines first (1) the patterns containing item 'F' and then (2) the patterns including 'D' but not 'F',. . .and finally the patterns containing item 'A'. Fig. 6 is the conditional (weighted) FP-tree of each node in head table.

Using 5 necessary ordered checkings described in section B, we can get the final global maximal weighted frequent patterns of each node in f_list. It is shown in Fig. 7(a). Due to

Suffix node *in f-list*	Result patterns
F	<C,F>
D	∅
E	<B,C,E>
B	∅
C	<A,C>
A	∅

(a) Result patterns of each node



(b) The maximal frequent itemset tree

Figure 7. Result patterns of *f-list* & MFI-tree

WTMaxMiner (*Weighted* **T**reversals-based **M**aximal **F**requent **P**atterns **M**iner)

Inputs: (1)A traversal transaction database TDB based on directed graph, ***G***; (2) A minimum support threshold: ***minwsup***, and (3) Weights of each node on graph ***G***
Output: Set of real maximal weighted frequent patterns
Method:
1. Scan TDB once to find the global weighted frequent items , and sort them by ***sup_count*** descending to form the set of ***f-list***.
2. Remove weighted infrequent items and sort the remaining weighted frequent items in each transaction by ***sup_count*** descending order.
3. Scan TDB and build FP-tree using the ***f-list***.
4. Initialize ***Max***. //***Max*** is the result set of closed weighted frequent patterns
5. Call ***WTMaxM*** **(*T, Max, MFI-tree*)**

Figure 8. Algorithm WTMaxMiner

Procedure WTMaxM **(*T, Max, MFI-tree*)**

puts: (1) ***T***, an FP-tree, (2) ***Max***, the result set of maximal weighted frequent patterns, and (3) ***MFI-tree***, an tree building with real closed weighted frequent patterns
Output: Set of real maximal weighted frequent patterns
Method:

```
1: if T only contains a single path B {
2:   from the single path B mine all (approximate) maximal weighted frequent patterns set, AWF
3:   for (∀P∈AWF) {
4:     if (∑i=1,|P|weight(pi)/|P|*support(P)≥minwsup){//get local real candidate pattern
5:     SP={sp1,sp2,…,spm|spi is a certain presentation of P with the original orders of vertices
         according to original TDB T on G, 1≤i≤m, ∑i=1,m sup_count(spi)= sup_count(P)}
6:     for (∀spi∈SP) {
7:        if (∑j=1,|spi|weight(tj)/|spi|*support(spi)≥minwsup) //tj is the vertex in spi
8:          LRWFP=LRWFP∪ syi;} //end of 6 for
9:     } //end of line 4 if
10:  } //end of line 3 for
11:  set LM=localmaximal(LRWFP) //LM is the local weighted maximal patterns
12:  if (not maximal-subset-checking(LM,MFI-tree)) {
13:      Max=Max∪LM;
14:      insert LM into MFI-tree; } //end of line 12 if
15: } //end of line 1 if
16: else {
17:  for (∀i∈f-list in T.header) { //from bottom to up
18:   initialize LRWFP=∅ //LRWFP is set of local really weighted frequent patterns
19:   set Y={i}∪T.base;
20:   if (max(W)*support(Y) ≥ minwsup) {
21:     if (∑i=1,|Y|weight(yi)/|Y|*support(Y) ≥ minwsup){
22:      SY={sy1,sy2,…,sym|syi is a certain presentation of Y with the original orders of vertices
          according to original TDB T on G, 1≤i≤m, ∑i=1,m sup_count(syi)= sup_count(Y)}
23:      for (∀syi∈SY) {
24:        if (∑j=1,|syi|weight(tj)/|syi|*support(syi)≥minwsup) //tj is the vertex in syi
25:          LRWFP=LRWFP∪syi;} //end of 23 for
26:    } //end of 21 if
27:   } //end of 20 if
28:   Gen−conditionalDB(i); //generate i's conditional database
29:   scan Gen−conditionalDB(i) once, remove items whose approximate wsupport≤minwsup and
      sort the remaining weighted frequent items in each transaction by sup_count descending order.
30:   Gen-FP-tree(Y); //generate the FP-tree of Y
31:   call WTMaxM (FP-treeY,Max,MFI-tree);
32:  } //end of 17 for
33: } //end of line16 else
```

Figure 9. Procedure WTMaxM

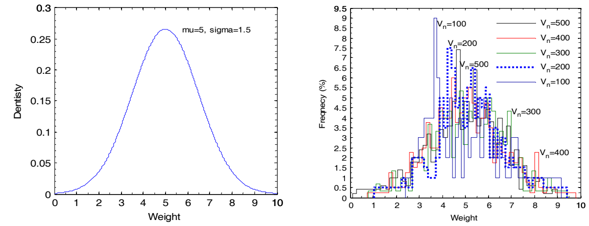
the space limitation, here we do not give the detail of mining process.

D. *WTMaxMiner Algorithm*

In *WTMaxMiner*, a descending *sup_count* order method and a *divide-and-conquer* traversal paradigm are used to mine weighted FP-tree for mining closed weighted patterns in bottom-up manner. *WTMaxMiner* algorithm is given in Fig. 8. Fig. 9 gives the procedure *WTMaxM* (*T, Max, MFI-tree*), in it, the MFI-tree is used to store so far found (global) maximal weighted frequent patterns. For the traversal transaction in Fig. 1, after mining, the set of real maximal weighted frequent patterns is :{ <CF>, <BCE>, <AC>}, and the MFI-tree is shown as Figure 7(b), there the node '*x:l*' means that the node is for item '*x*', its level is '*l*' (the length of path from this node to the root node).

V. EXPERIMENTAL EVALUATION

We explored our experimental results on the performance of WTMaxMiner in comparison with FPmax [12]. Because there are not real datasets about WDG currently, we test the algorithm performance using synthetic dataset. We implement our algorithm with C++ language, running under Microsoft VC++ 6.0. The experiments were performed on Windows XP Professional operating system with Pentium IV PC at 2.93 GHz and 768MB of main memory. We used Microsoft SQL Server 2000 database to generate simulation of *WDG* and the traversals no it. All the reported runtimes are in seconds.



(a) Gauss distribution density (b) Weight distributions of different V_n

Fig. 10 Weight distribution

A. *Generate Synthetic Datasets*

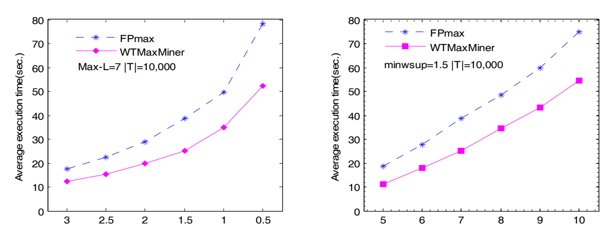
During the experiment, the *WDG* is generated mainly according to following parameters: number of vertices and max number of edges per vertex. And then, we assigned random weight to each vertex of the graph. To easily compare algorithm's performance, we generated 8 sets of traversal database with the same set of weights, in each of which the maximum length of traversals varies from 5 to 10. The distribution of weight is generated from Gauss distribution (μ=5.0, σ=1.5) shown as Fig.10. All experimental results are average value of 8 sets of synthetic datasets. Due to space considerations, our experimental evaluation only illustrates the results of one smaller value of *minwsup* (*minwsup*=1.5) in all of our experiments. However, our own experimental evaluation (not presented in this paper) showed that it is equally effective for other value of *minws*up. The characteristics of these datasets are summarized in Table I.

B. *Effectiveness Comparison of WTMaxMiner and FPmax*

For algorithms *FPmax* and *WTMaxMiner*, we made a set of comparing experiments among *minwsup*, *Max_L* and execution time etc. The difference between *WTMaxMiner* and *FPmax* results from weight constraints. Figure 11 shows the trend of the execution time of *WTMaxMiner* and *FPmax* with respect to different *minwsup* and *Max_L* based on |*T*|*=10,000.* As shown in Fig. 11(a), along with *minwsup*'s decreasing, the average execution time of two algorithm increases. When the

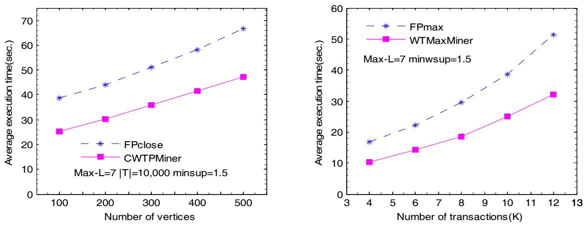
TABLE I. EXPERIMENTAL PARAMETERS

Parameter	Meaning or parameter	Value in experiment
V_n	Number of nodes per set	100 to 500
E_{max}	Max number of edges per vertex	$1\leq E_{max} \leq 4$
w_i	Weight of vertex	$0\leq w_i \leq 10$
minwsup	Min weighted support threshold	1 to 5
|*T*|	Number of traversals per set	4k to 12k
num_TSet	Number of traversal set	8
Max_L	Max length of pattern per traversal set	5,6,7,8,9,10



(a) (b)

Figure 11. Runtime comparison w.r.t. different *minwsup* & *Max_L*



(a). Number of vertices (b). Number of traversal transaction

Figure 12. Scale-up property test of WTMaxMiner

minwsup is lowered, the performance difference between two algorithms becomes larger. In all case of *minwsup*, *WTMaxMiner* outperforms algorithm *FPmax*. This is because *WTMaxMiner* carries out the maximal frequent pattern mining with weight constraints, and can reduced effectively search space, but *FPmax* only do the maximal frequent pattern mining without weight constraints, its search space is larger than that of algorithm *WTMaxMiner*. Our own experimental evaluation (not presented in this paper) showed that *WTMaxMiner* is equally effective for other value of *Max_L* as well. Figure 11(b) shows that *WTMaxMiner* is faster than FPmax. The performance difference between two algorithms becomes larger when *Max_L* becomes longer.

C. *Scalability Study*

To evaluate how the performance of *WTMaxMiner* scales with the size of the database, we performed an experiment in which we respectively varied the number of vertices from 100 to 500 and the number of traversal transactions $|T|$ from 4 to 12k for the synthetic datasets. Due to space considerations, we only illustrate the results of *Max_L*=7, and the experimental results (not presented in this paper) of other value of *Max_L* has a similar performance to it. Fig. 12 shows the experimental results. From these results we can see *WTMaxMiner* approximately scales linearly with the size of the vertices and traversal transactions. As shown in Fig. 12, *WTMaxMiner* has much better scale-up properties than *FPmax* with respect to the numbers of vertices & the number of traversal transactions. Fig. 12(a) shows, although itself runtime also increases, *WTMaxMiner* runs faster than *FPmax* along with increase of number of vertices. In Fig. 12(b), *WTMaxMiner* also has a better scalability in terns of number of traversal transactions and runs faster than *FPmax*. The reason for that is *WTMaxMiner* has a weight constraint to reduce the search space than FPmax which has not weight constraint.

VI. CONCLUSIONS AND FUTURE WOEKS

This paper explores the problems of discovering maximal frequent patterns with weight constraint from weighted traversals on graph. Differently from previous approaches, vertices of directed graph are attached with weights which reflect their importance. With the weight setting, a transformable model between *EWDG* and *VWDG* is proposed. Based on the model, we present algorithm named *WTMaxMiner*. In this algorithm, we use *divide-and-conquer* paradigm with a bottom-up pattern-growth method and incorporates the closure property with weight constrain to reduce effectively search space. Experimental results on synthetic datasets show that the algorithms is effective and scalable to the problem of mining maximal weighted frequent patterns based on *WDG* traversals. Many opportunities exist to apply WDG traversals-based maximal frequent pattern mining. How to scale the model and algorithms proposed in this paper to a larger scale, can we deeply optimize the algorithm, and how to efficiently put it into practice are still worthy doing further explorations for researches.

REFERENCES

[1] Chen, M.S., Park, J.S., Yu, P.S., “Efficient data mining for path traversal patterns,” IEEE Trans. on Knowledge and Data Engineering, Los Alamitos, CA, USA, vol. 10, pp. 209–221, April 1998.

[2] Nanopoulos, A., Manolopoulos, Y., “Mining patterns from graph traversals”, Data and Knowledge Engineering, Netherlands, vol. 37, pp. 243–266, June 2001.

[3] YEN S. J., Chen A. L P., “A graph-based approach for discovering various types of association rules”, IEEE Trans. on Knowledge and Data Engineer, Los Alamitos, CA, USA, vol.13, pp. 839-845, Oct. 2001.

[4] R. Agrawal, R. Srikant, “Fast algorithms for mining association rules”, In: proceedings of VLDB’94, Santigo, Chile, pp. 487-499, Sept. 1994.

[5] N. Pasquier, Y. Bastide, et.al., “Discovering frequent closed itemsets for association rules”, Proceedings of ICDT'99, Jerusalem, Israel, pp. 398-416, Jan. 1999.

[6] M. J. Zaki,C.-J. Hsiao, “Charm: an efficient algorithm for closed itemsets mining”, Proceedings of 2002 SIAM Int. Conf. on Data Mining, Arlington,VA, pp. 12-28, April 2002.

[7] J. Wang, J. Pei, J. Han, “Closet+: searching for the best strategies for mining frequent closed itemsets”, Proceedings of SIGKDD’03, Washington, D.C., pp. 236-245, Aug. 2003.

[8] R. J. Bayardo, “Efficiently mining long patterns from databases”, Proceeding of Special Interest Group on Management of Data, Seattle, WA, pp. 85-93, June 1998.

[9] D. Burdick, M. Calimlim, and J. Gehrke, “Mafia: a maximal frequent itemset algorithm for transactional databases”, Proceedings of ICDE’01, Heidelberg, Germany, pp. 443-452, April, 2001.

[10] K. Gouda, M.J. Zaki, “Efficiently Mining Maximal Frequent Itemsets”, In Proceedings of ICDM’01, San Jose pp. 163-170, Nov. 2001.

[11] Yuejin Yan, Zhoujunli, Tao Wang, Yuexin Chen and Huowang Chen, “Mining maximal frequent itemsets using combined FP-Tree”, Proceedings of the 17th Australian Computer Society (ACS) Australian Joint Conference on Artificial Intelligence (AI'2004), Cairns, Australia, pp. 475-487, Dec., 2004.

[12] Gösta Grahne and Jianfei Zhu, “High performance mining of maximal frequent itemsets”, In Proceedings of SIAM’03 Workshop on High Performance Data Mining: Pervasive and Data Stream Mining, pp. 135-143, May 2003.

[13] J. Han, J. Pei, and Y. Yin, “Mining frequent patterns without candidate generation”, Proceedings of SIGMOD’00, Dallas, pp. 1-12, May 2000.

[14] W. Wang, J. Yang, P.S. Yu, “Efficient mining of Weighted Association Rules (WAR)”, Proceedings of SIGKDD’00, Boston, Massachusetts, pp. 270-274, Aug.2000.

[15] F. Tao, F. Murtagh, M. Farid, “Weighted association rule mining using weighted support and significance framework”, Proceedings of ACM SIGKDD’03, Washington DC, USA. pp. 661–666, Aug. 2003.

[16] Cai, C.H., Ada, W.C., Fu, W.C et. al., “Mining association rules with weighted items”, In Proceedings of IDEAS’98, Cardiff, U, pp. 68–77, Aug. 1998.

[17] U. Yun, J.J. Leggett, “WLPMiner: weighted frequent pattern mining with length-decreasing support constraints”, Proceedings of PAKDD‘05, Hanoi, pp. 555-567, May, 2005.

[18] F. Bonchi, C. Lunnhese, “On closed constrained frequent pattern mining”, Proceedings of ICDM‘04, Brighton, UK, pp. 35-42, Nov., 2004.