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

Runian GENG1,2, Xiangjun DONG2
2. School of Information Science and Technology
Shandong Institute of Light Industry
Jinan, China
gengrmn@163.com, d-xj@163.com,

Ping ZHANG1, Wenbo XU1
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 (Weighted Traversals-based Maximal Frequent Patterns Miner) to discover maximal frequent patterns from weighted 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...
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 (Weighted Directed Graph), 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 WDGs. One is \( VWDG \) (Vertex-WDG) which assigns weights to each vertex in the graph, and the other is \( EWDG \) (Edge-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.

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.

\[
support(P) = \frac{\text{sup\_count} (P)}{|T|}.
\]  

Definition 5 (Maximal frequent traversal pattern) Given a threshold minimum support \( min\_sup \), a traversal pattern \( Y \) is a maximal frequent traversal pattern if \( P \) \( \supset \) \( Y \), \( \frac{sup\_count} {w\_support} \) \( P \geq min\_sup \).

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 \( P=<p_1,p_2,\ldots,p_k> \), the weight of each item in \( P \), denoted as \( w(p_i) \) \( (i=1,2,\ldots,k) \), then the weight of \( P \) is represented as follows.

\[
weight(P) = \frac{\sum w(p_i)}{|P|}
\]  

Definition 8 (Weighted Support) The weighted support of a pattern \( P \), called \( w\_support \) \( (P) \), is

\[
\text{w\_support} (P) = \text{Weight} (P) (\text{support} (P))
\]  

A \( P \) is said to be weighted frequent when its weighted support is no less than a given minimum weighted support threshold called \( min\_w\_sup \), i.e.,

\[
\text{w\_support} (P) \geq \text{min\_w\_sup}
\]  

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 \( WDGs \) 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
weight value, and their linking directions refer to the source EWDG.

Figure 3 describes this change method: Fig. 3 (a), (c) and (e) are EWDGs, and Fig. 3(b), (d), (g) and (b) are VWDGs. In Fig. 3(b) or (d), the nodes named ‘bd’, ‘de’, ‘ec’ and ‘bc’ respectively represent 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 (b) 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 represent 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,…,n), must satisfy: \( \min(W) \leq w_i \leq \max(W) \), there, \( W = \{w_1,w_2,…,w_n\} \). To let weighted patterns satisfy the anti-monotone property (i.e., if \( wsupport(P) \leq \minwsup \Rightarrow \wsupport(P') \leq \minwsup \)), we revise weight \( (p) \) as two following representations.

\[
\text{weight}(p) = \min(W) \quad \text{or} \quad \text{weight}(p) = \max(W)
\]

That is, the weight of pattern is revised as:

\[
\text{weight}(P) = \frac{\sum w(p)}{|P|} = \frac{\sum \min(W)}{|P|} \quad \text{(5)}
\]

or

\[
\text{weight}(P) = \frac{\sum w(p)}{|P|} = \frac{\sum \max(W)}{|P|} \quad \text{(6)}
\]

Thus, the anti-monotone property can be used in mining weighted frequent patterns since \( \wsupport(P') \leq \wsupport(P) \). 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:

\[
\sum_{i=1}^{k} \text{weight}(p) \geq \text{sup}(P) \Rightarrow \minwsup .
\]

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].

\[
\text{(I)} \quad Cl(\text{FTh}_s\{sup(X) \geq \minsup\}) \cap \text{FTh}_s(C_{an}).
\]

\[
\text{(II)} \quad Cl(\text{FTh}_s\{sup(X) \geq \minsup\} \cup C_{an}).
\]

There, \( C_{an} \) 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. \( \text{MFI}_c\text{CFI} \)), 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.
and exploits a divide-and-conquer strategy with a pattern growth method, to mine the maximal weighted 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 frequent patterns 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 $\text{minwsup} = 1.5$ (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 WTMMaxMiner

Because we use a revised weight support, so we only generates approximate weighted frequent patterns 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 $S_{\text{c}}$ is compared with maximal weighted frequent patterns in the MFI-tree. If there is no superset of the pattern $S_{\text{c}}$ in MFI-tree, the pattern $S_{\text{c}}$ 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 correct 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 $\text{wsupport} (P) = \text{support} (P) \times \text{max} (W) \geq \text{minwsup}$. (2) Toward newly found (local) candidate approximate weighted frequent patterns, we check their real $\text{wsupport}$, and those whose real $\text{wsupport} \geq \text{minwsup}$ 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 $\text{sup_count}$ 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 $\text{sup_count}$ referring to the path on $G$. If corresponding $\text{sup_count}$ of some pattern $P$ is changed, we must check if $P$ satisfies the condition: $\text{wsupport} (P) = \text{support} (P) \times \text{Weight} (P) \geq \text{minwsup}$. Those patterns whose real $\text{wsupport}$ is less than $\text{minwsup}$ 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 the non-maximal local patterns and remain the 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, WTMMaxMiner mines maximal weighted frequent patterns by adapting the divide-and-conquer approach. For the traversal transaction database shown in Fig. 1(b) ($\text{minwsup} = 1.5$), 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
WTMaxMiner (Weighted Tree-based Maximal Frequent Patterns Minier)

**Inputs**: (1) A traversal transaction database TDB based on directed graph G; (2) A minimum support threshold minsup and (3) Weights of each node on graph G.

**Outputs**: 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 B.
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 MFI-tree using the B.
4. Initialize Max | Max the result set of closed weighted frequent patterns.
5. Call WTMaxM(G,M,FMI-tree).

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

```plaintext
public (1) T an FP-tree, Max the result set of maximal weighted frequent patterns, and (3) MFI-tree an building with real closed 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, MWF.
3. for (Vp∈MWF) { if sp is a certain presentation of P with the original orders of vertices according to original TDB then, lsi is the first vertex in sp.
4. for (Vp∈MWF) { if (yi|weight(P(1)yi)*minsup) then, lj is a certain presentation of P with the original orders of vertices according to original TDB P on G, lsi is the vertex in sp.
5. } end of line 4 for
6. } end of line 3 for
7. for (yi|weight(P(1)yi)*minsup) then, lj is a certain presentation of P with the original orders of vertices according to original TDB P on G, lsi is the vertex in sp.
8. end of line 12 if
9. } end of line 1 if
10. if (yi|weight(P(1)yi)*minsup) then, lj is a certain presentation of P with the original orders of vertices according to original TDB P on G, lsi is the vertex in sp.
11. } end of line 1 if
12. if (yi|weight(P(1)yi)*minsup) then, lj is a certain presentation of P with the original orders of vertices according to original TDB P on G, lsi is the vertex in sp.
13. } end of line 1 if
14. } end of line 4 if
15. } end of line 3 for
16. } end of line 12 if
17. } end of line 1 if
18. } end of line 12 if
19. } end of line 4 if
20. } end of line 3 for
21. } end of line 1 if
22. } end of line 12 if
23. } end of line 1 if
24. } end of line 12 if
25. } end of line 4 if
26. } end of line 3 for
27. } end of line 1 if
28. } end of line 12 if
29. } end of line 4 if
30. } end of line 3 for
31. } end of line 1 if
32. } end of line 12 if
33. } end of line 4 if
```

**Figure 8. Algorithm WTMaxMiner**

**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 Fig. 7(b), there the node ‘x’ 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 FPMX [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 to it. All the reported runtimes are in seconds.

**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 Gaussian 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 minsup (minsup=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 minsup. The characteristics of these databases are summarized in Table I.

**B. Effectiveness Comparison of WTMaxMiner and FPMX**

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

![Graph showing the comparison between WTMaxMiner and FPMX](image-url)

**Figure 10. Weight distribution**

**Figure 11. Runtime comparison w.r.t. different minsup & Max L**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning or parameter</th>
<th>Value in experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nv</td>
<td>Number of nodes per set</td>
<td>100 to 300</td>
</tr>
<tr>
<td>wv</td>
<td>Weight of vertex</td>
<td>1.0 to 500</td>
</tr>
<tr>
<td>minsup</td>
<td>Min weighted support threshold</td>
<td>0.5 ≤ σ ≤ 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Number of traversals per set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Number of traversals per set</td>
</tr>
<tr>
<td>Max L</td>
<td>Max length of pattern per traversal</td>
<td>5,6,7,8,9,10</td>
</tr>
</tbody>
</table>
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 [7] 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 terms 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 from FPmax which has not weight constraint.

VI. CONCLUSIONS AND FUTURE WORKS

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 WWDG 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