# Mining Weighted Frequent Patterns from Path Traversals on Weighted Graph

*Seong Dae Lee,[†] and Hyu Chan Park[††]*

[†, ††] Department of Computer Engineering, Korea Maritime University, Korea
[††] Corresponding author

**Summary**
A lot of real world problems can be modeled as traversals on graph, and mining from such traversals has been found useful in several applications. However, previous works considered only traversals on unweighted graph. This paper generalizes this to the case where vertices of graph are given weights to reflect their importance. Under such weight settings, traditional mining algorithms can not be adopted directly any more. To cope with the problem, this paper proposes new algorithms to discover weighted frequent patterns from the traversals. Specifically, we devise support bound paradigms for candidate generation and pruning during the mining process.

***Key words:***
*Data mining, Graph, Traversal, Weighted frequent pattern*.

## 1. Introduction

Graph and traversal on it are widely used to model several classes of real world problems. For example, 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. Furthermore, user navigations on the Web site can be modeled as traversals on the graph. Once a graph and its traversals are given, valuable information can be discovered. Most common form of the information may be frequent patterns, i.e., the sub-traversals that are contained in a large ratio of traversals. However, previous works have not considered any weight on the graph [1, 2, 3].

This paper extends previous works by considering weights attached to the vertices of graph. Such vertex weight may reflect the importance of vertex. For example, each Web page may have different importance which reflects the value of its content. With the weight setting, the mining algorithm can not be relied on the well-known Apriori paradigm any more. The reason why Apriori paradigm works is due to the downward closure property, which says all the subsets of a frequent pattern must be frequent. With the weight setting, however, it is not necessarily true that all the subpatterns of a weighted frequent pattern are weighted frequent. Therefore, we adopt the notion of support bound [4]. On top of the notion, we propose a mining algorithm for the discovery of weighed frequent patterns.

This paper is organized as follows. In Section 2, we review previous works related with the traversal pattern mining and weighted mining. Section 3 and 4 propose an algorithm for the discovery of the weighted frequent patterns from traversals on weighted graph. Section 5 includes two methods for the estimation of weight and support bound used in this paper. In Section 6, we experiment and analyze the algorithm on synthetic data. Finally, Section 7 contains the conclusion and future works.

## 2. Related Works

The main stream of data mining, which is related to our work, can be divided into two categories, i.e. the traversal pattern mining and the weighted mining. For the traversal pattern mining, there have been few works. Chen et al. [1] proposed the problem of traversal pattern mining, and then proposed algorithms with hashing and pruning techniques. However, they did not consider graph structure, on which the traversals occur. Nanopoulos et al. [2, 3] proposed the problem of mining patterns from graph traversals. They defined new criteria for the support and subpath containment, and then proposed algorithms with a trie structure. They considered the graph, on which traversals occur. Although the above works dealt with the mining of traversal patterns, to the best of our knowledge, there is no work which considers the notion of weight as our one.

For the weighted mining, most of previous works are related to the mining of association rules and its sub-problem, the discovery of frequent itemsets. Cai et al. [4] generalized the discovery of frequent itemsets to the case where each item is given an associated weight. They introduced new criteria to handle the weights in the process of finding frequent itemsets, such as the weighted support for the measurement of support, and the support bound for the pruning of candidates. Wang et al. [5] extended the problem by allowing weights to be associated with items in each transaction. Their approach ignores the weights when finding frequent itemsets, but considers

Manuscript received April 5, 2007
Manuscript revised April 25, 2007

during the association rule generation. Tao et al. [6] proposed an improved model of weighted support measurement and the weighted downward closure property. Yun et al. [7] also considered weighted items in the process of frequent itemsets, and the length-decreasing support constraints for a new measurement of support. Although the above works take the notion of weight into account as examined in this paper, they can not be adapted directly to our work because they only concerned on the mining from items, but not from traversals.

## 3. Weighted Frequent Patterns

**Definition 1.** A *weighted directed graph* is a finite set of vertices and edges, in which each edge joins one ordered pair of vertices, and each vertex is associated with a weight value. A *base graph* is a weighted directed graph, on which traversals occur.

For example, the following base graph has 6 vertices and 8 edges, in which each vertex is associated with a weight.

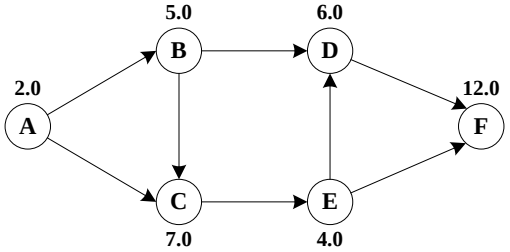


Fig. 1 Example of base graph

**Definition 2.** A *traversal* is a sequence of consecutive vertices along a sequence of edges on a base graph. We assume that every traversal is path, which has no repeated vertices and edges. The *length* of a traversal is the number of vertices in the traversal. The *weight* of a traversal is the sum of vertex weights in the traversal. A *traversal database* is a set of traversals.

We restrict any traversal to be a path, because repeated vertices or edges in a traversal may not contain useful information in many cases, such as backward movements. If a traversal has repeated vertices or edges, it can be separated into several paths, such as maximal forward references [1]. The following traversal database has totally 6 traversals, each of which has an identifier and a sequence of consecutive vertices.

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

Fig. 2 Example of traversal database

**Definition 3.** A *subtraversal* is any subsequence of consecutive vertices in a traversal. If a pattern $P$ is a subtraversal of a traversal $T$, then we say that $P$ is *contained* in $T$, and vice versa $T$ *contains* $P$.

There is a well known property on such subtraversal [2, 3] as follows.

**Property 1.** Given a traversal of length $k$, there are only two subtraversals of length $k$-$1$.

For example, given a traversal of length 4, <B, C, E, F>, there are only two subtraversals of length 3, <B, C, E> and <C, E, F>. Note that non-consecutive sequences, such as <B, C, F>, are not subtraversals.

**Definition 4.** The *support count* of a pattern $P$, $scount(P)$, is the number of traversals containing the pattern. The *support* of a pattern $P$, $support(P)$, is the fraction of traversals containing the pattern. Given a traversal database $D$, let $|D|$ be the number of traversals.

$$support(P) = \frac{scount(P)}{|D|} \tag{1}$$

There is a well known property on such support count and support as follows.

**Property 2.** The support count and the support of a pattern decrease monotonically as the length of the pattern increases. In other word, given a $k$-pattern $P$ and any $l$-pattern containing $P$, denoted by $(P, l)$, where $l > k$, then $scount(P) \geq scount(P, l)$ and $support(P) \geq support(P, l)$.

Given a base graph with a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$, in which each vertex $v_j$ is assigned with a weight $w_j \geq 0$, we will define the weighted support of a pattern.

**Definition 5.** The *weighted support* of a pattern $P$, $wsupport(P)$, is

$$wsupport(P) = \left( \sum_{v_j \in P} w_j \right) \left( support(P) \right) \tag{2}$$

**Definition 6.** A pattern $P$ is said to be *weighted frequent* when the weighted support is greater than or equal to a given minimum weighted support (*minwsup*) threshold,

$$wsupport(P) \geq minwsup \tag{3}$$

For example, given a base graph and traversal database of Fig. 1 and 2, and *minwsup* of 5.0, then the pattern <B, C, E> is weighted frequent since (5.0 + 7.0 + 4.0) × 2/6 = 5.3 ≥ 5.0, but the pattern <B, C> is not since (5.0 + 7.0) × 2/6 = 4.0 < 5.0.

From equation (1), (2) and (3), a pattern $P$ is weighted frequent when its support count satisfies:

$$scount(P) \geq \frac{minwsup \times |D|}{\sum_{v_j \in P} w_j} \tag{4}$$

We can consider the right hand side of (4) as the lower bound of the support count for a pattern $P$ to be weighted frequent. Such lower bound, called support bound, is given by

$$sbound(P) = \left\lceil \frac{minwsup \times |D|}{\sum_{v_j \in P} w_j} \right\rceil \tag{5}$$

We take the ceiling of the value since the function *sbound(P)* is an integer. From Equation (4) and (5), we can say a pattern $P$ is weighted frequent when the support count is greater than or equal to the support bound.

$$scount(P) \geq sbound(P) \tag{6}$$

Note that *sbound(P)* can be calculated from base graph without referring traversal database. On the contrary, *scount(P)* can be obtained by referring traversal database.

The problem concerned in this paper is stated as follows. Given a weighted directed graph (base graph) and a set of path traversals on the graph (traversal database), find all weighted frequent patterns.

## 4. A Framework for Mining Weighted Frequent Patterns

We propose a framework for the mining of weighted frequent patterns. An efficient algorithm for mining large itemsets has been Apriori algorithm. The reason why Apriori algorithm works is due to the downward closure property, which says all the subsets of a large itemset must be also large. For the weighted setting, however, it is not necessarily true for all the subpatterns of a weighted frequent pattern being weighted frequent. For example, although a pattern <B, C> is a subpattern of the weighted frequent pattern <B, C, E>, it is not weighted frequent. Therefore, we can not directly adopt Apriori algorithm. Instead, we will extend the notion of support bound, which can be applied to the pruning and candidate generation.

### 4.1 Pruning by Support Bound

One of the cornerstones to improve the mining performance is to devise a pruning method which can reduce the number of candidates as many as possible. We must prune such candidates that have no possibility to become weighted frequent in the future. On the contrary, we must keep such candidates that have a possibility to become weighted frequent in the future. Main concern is how to decide such possibility.

**Definition 7.** A pattern $P$ is said to be *feasible* when it has a possibility to become weighted frequent in the future if extended to longer patterns. In other words, when some future patterns containing $P$ will be possibly weighted frequent.

Now, the pruning problem is converted to the feasibility problem. For the decision of such feasibility, we will first devise the weight bound of a pattern. Let the maximum possible length of weighted frequent patterns be $u$, which may be the length of longest traversal in the traversal database. Given a $k$-pattern $P$, suppose $l$-pattern containing $P$, denoted by *(P, l),* where $k < l \leq u$. For the additional $(l - k)$ vertices, if we can estimate upper bounds of the weights as $w_{r1}, w_{r2}, \ldots, w_{rl-k}$, then the upper bound of the weight of the $l$-pattern *(P, l)* is given by

$$wbound(P,l) = \sum_{v_j \in P} w_j + \sum_{j=1}^{l-k} w_{rj} \tag{7}$$

We call this upper bound as *l-weight bound* of $P$. The first sum is the sum of the weights for the $k$-pattern $P$. The second one is the sum of the $(l - k)$ estimated weights, which can be estimated in several ways. We will propose two estimation methods in the following section.

From (5) and (7), we can derive the lower bound of the support count for $l$-pattern containing $P$ to be weighted frequent. Such lower bound, called *l-support bound* of *P,* is given by

$$sbound(P,l) = \left\lceil \frac{minwsup \times |D|}{wbound(P,l)} \right\rceil \tag{8}$$

**Lemma 1.** A pattern $P$ is feasible if $scount(P) \geq sbound(P, l)$ for some $k < l \leq u$, but not feasible if $scount(P) < sbound(P, l)$ for all $k < l \leq u$.

**Proof.** Let $l_i$ be anyone out of $l$. If $scount(P) \geq sbound(P, l_i)$, then because $scount(P) \geq scount(P, l_i)$ by Property 2, there is a possibility to be $scount(P, l_i) \geq sbound(P, l_i)$. It means that $(P, l_i)$ will possibly be weighted frequent. On the contrary, if $scount(P) < sbound(P, l_i)$, then because $scount(P) \geq scount(P, l_i)$ by Property 2, $scount(P, l_i) < sbound(P, l_i)$. It means that $(P, l_i)$ will definitely not be weighted frequent.

If a pattern $P$ is feasible then some $l$-patterns containing $P$ will be possibly weighted frequent. In other word, $P$ has a possibility to be subpatterns of some weighted frequent $l$-patterns. Therefore, $P$ must be kept to be extended to longer patterns for possible weighted frequent patterns in the coming passes. On the contrary, if a pattern $P$ is not feasible, then all $l$-patterns containing $P$ will not be weighted frequent. In other word, $P$ certainly has no possibility to be subpattern of any weighted frequent $l$-patterns. Therefore, $P$ must be pruned.

For example, referring to Fig. 1 and Fig. 2, given a 2-pattern <B, C>, suppose 3-pattern <B, C, –>. For the additional vertex '–', we can estimate a possible upper bound of the weight as 12.0, which is the greatest weight among the remaining vertices besides B and C. Therefore, the 3-support bound of <B, C> is

$$sbound(< B, C >, 3) = \left\lceil \frac{5.0 \times 6}{(5.0 + 7.0) + (12.0)} \right\rceil = 2$$

It means if the support count of <B, C> is greater than or equal to 2, some 3-patterns will be possibly weighted frequent. In other word, <B, C> has a possibility to be subpatterns of some weighted frequent 3-patterns. Because the support count of the pattern <B, C> is actually 2, the pattern must be extended to 3-patterns for possible weighted frequent patterns.

**Corollary 1.** A pattern $P$ is feasible if $scount(P) \geq sbound(P)$.

**Proof.** From Equation (5), (7) and (8), $sbound(P) \geq sbound(P, l)$ for all $k < l \leq u$. Therefore, $scount(P) \geq sbound(P, l)$ for all $k < l \leq u$, which means $P$ is feasible by Lemma 1.

In this case, we don't need to estimate $sbound(P, l)$ to decide the feasibility of $P$. On the contrary, in the case of $scount(P) < sbound(P)$, we can not decide the feasibility, and therefore we need to estimate $sbound(P, l)$ to decide the feasibility by Lemma 1.

According to Lemma 1 along with Corollary 1, we can devise a pruning algorithm, called 'pruning by support bounds', as follows.

**Algorithm.** ***Pruning-SB***

```
for each pattern P in candidates set C_k {
  if (scount(P) ≥ sbound(P))
    continue;   // P is feasible. keep
  for each l from k+1 to u {
    estimate sbound(P, l);
    if (scount(P) ≥ sbound(P, l))
      break;   // P is feasible. keep
  }
  if (l > u)
    C_k = C_k – {P};   // P is not feasible. prune
}
```

We can devise another pruning algorithm by using the minimum of $l$-support bounds.

**Definition 8.** The *max l-weight bound, wbound(P, +),* and the *min l-support bound* of a pattern $P$, *sbound(P, +),* are defined as follows.

$$wbound(P, +) = max(wbound(P, l)),$$
$$sbound(P, +) = min(sbound(P, l)), k < l \leq u.$$

**Corollary 2.** A pattern $P$ is feasible if $scount(P) \geq sbound(P, +)$, but not feasible if $scount(P) < sbound(P, +)$.

**Proof.** If $scount(P) \geq sbound(P, +)$, then there is at least one $l_i$ such that $scount(P) \geq sbound(P, l_i)$, where $sbound(P, l_i) = sbound(P, +)$. On the contrary, if $scount(P) < sbound(P, +)$, then $scount(P) < sbound(P, l)$ for all $k < l \leq u$.

According to Corollary 2 along with Corollary 1, we can devise another pruning algorithm, called 'pruning by min support bound', as follows.

**Algorithm.** ***Pruning-MSB***

```
for each pattern P in candidates set C_k {
  if (scount(P) ≥ sbound(P))
    continue;   // P is feasible. keep
  estimate sbound(P, +);
  if (scount(P) ≥ sbound(P, +))
    continue;   // P is feasible. keep
  C_k = C_k – {P};   // P is not feasible. prune
}
```

## 4.2 Candidate Generation

We will devise candidate generation algorithms by defining downward closure properties between feasible patterns. If there is a downward closure property between

feasible patterns, new candidates can be generated from current feasible patterns.

**Definition 9.** We say that there is *partial downward closure property* when the *(k-1)*-subpattern $<p_1, p_2, ..., p_{k-1}>$ of a feasible *k*-pattern $<p_1, p_2, ..., p_k>$ is also feasible. We say that there is *full downward closure property* when two *(k-1)*-subpatterns $<p_1, p_2, ..., p_{k-1}>$ and $<p_2, p_3, ..., p_k>$ of a feasible *k*-pattern $<p_1, p_2, ..., p_k>$ are also feasible.

Note that there are only two *(k-1)*-subpatterns of a *k*-pattern by Property 1. When there is the partial downward closure property, we can generate candidate *(k+1)*-patterns, $C_{k+1}$, from feasible *k*-patterns, $C_k$, as follows.

**Algorithm. *Gen-PDC***

```
for each P = <p1, p2, ..., pk> in Ck {
   for each edge <pk, v> in G
      if v is not already in P  // not repeated vertex
         P is extended to P' = <p1, p2, ..., pk, v>;
}
```

When there is the full downward closure property, we can generate $C_{k+1}$ in a similar way.

**Algorithm. *Gen-FDC***

```
for each P = <p1, p2, ..., pk> in Ck {
   for each edge <pk, v> in G
      if (v is not already in P) and (Q = <p2, ..., pk, v> is in Ck)
         P is extended to P' = <p1, p2, ..., pk, v>;
}
```

This algorithm will generate less number of candidates than Algorithm *Gen-PDC*.

When there is the full downward closure property, $C_{k+1}$ can be alternatively obtained by self-joining $C_k$. That is, two *k*-patterns $P = <p_1, p_2, ..., p_k>$ and $Q = <q_1, q_2, ..., q_k>$ will be joined if $p_2 = q_1, p_3 = q_2, ... p_k = q_{k-1}$, and $p_1 != q_k$. This results in a new candidate pattern $<p_1, p_2, ..., p_k, q_k>$. For example, the join of $<A\ B\ C>$ and $<B\ C\ D>$ results in $<A\ B\ C\ D>$. This method need not refer to the base graph *G*, besides for $C_2$ generation. For $C_2$ generation, each generated 2-pattern must be excluded if there is no corresponding edge in *G*.

**Algorithm. *Gen-SQL***

```
select P.p1, P.p2, ..., P.pk, Q.qk
from Ck P, Ck Q
where P.p2 = Q.q1, P.p3 = Q.q2, ... P.pk = Q.qk-1,
   and P.p1 != Q.qk.
```

This method need not refer to the base graph *G*, besides for $C_2$ generation. For $C_2$ generation, each generated 2-pattern must be excluded if there is no corresponding edge in *G*.

## 4.3 Mining Algorithm

By combing the pruning and candidate generation algorithms as a whole, we can devise an algorithm for mining weighted frequent patterns. Fig. 3 shows the algorithm proposed in this paper, which performs in a level-wise manner.

**Algorithm. *Mining weighted frequent patterns***

*Inputs:* Base graph *G*, Traversal database *D*, Minimum weighted support *minwsup*
*Output:* List of weighted frequent patterns $L_k$

```
{
   // 1. maximum length of weighted frequent patterns
   u = max(length(t)), t ∈ D;

   // 2. initialize candidate patterns of length 1
   C1 = V(G);

   for (k = 1; k ≤ u and Ck ≠ ∅; k++) {

      // 3. obtain support counts
      for each traversal t ∈ D {
         for each pattern p ∈ Ck
            If p is contained in t, then p.scount++;
      }

      // 4. determine weighted frequent patterns
      Lk = {p | p ∈ Ck, p.weightedSupport ≥ minwsup};
                  (equivalently, p.scount ≥ p.sbound)

      if (k < u) {
         // 5. prune candidates
         Ck = pruneCandidates(Ck, G, u);

         // 6. generate new candidates for next pass
         Ck+1 = genCandidates(Ck, G);
      }
   }
}
```

Fig.3 Algorithm for mining weighted frequent patterns

In the algorithm, each step is outlined as follows. Step 1 is to find out the maximum possible length of weighted frequent patterns, which is limited by the maximum length of traversals. Step 2 initializes candidate patterns of length 1 with the vertices of base graph. In Step 3, traversal database is scanned to obtain the support counts of candidate patterns. Step 4 is to determine weighted

frequent patterns if the weighted support is greater than or equal to the specified minimum weighted support. Equivalently, if the support count is greater or equal to the support bound. In Step 5, the subroutine *pruneCandidates($C_k$, G, u)* is to prune candidate patterns by checking their feasibility. The algorithm *Pruning-SB* or *Pruning-MSB* can be used according to their efficiency. The remaining patterns are feasible patterns. In Step 6, the subroutine *genCandidates($C_k$, G)* generates new candidate patterns of length $k+1$ from the feasible patterns of length $k$ for the next pass. The algorithm *Gen-PDC, Gen-FDC* or *Gen-SQL* can be used according to its applicability and efficiency.

## 5. Estimations of Support Bound

We propose two methods for the estimation of weight and support bound.

### 5.1 Estimation by All Vertices

Given a $k$-pattern $P$, suppose $l$-pattern containing $P$, where $k < l \le u$. Let $V$ be the set of all vertices in the base graph. Among the remaining vertices $(V - P)$, let the vertices with the $(l - k)$ greatest weights be $v_{r1}, v_{r2}, \ldots, v_{rl-k}$. Then, the $l$-weight bound, *wbound(P, l),* and the $l$-support bound, *sbound(P, l)*, of $P$ are defined same as Equation (7) and (8), respectively.

For example, refer to Fig. 1 and Fig. 2, the 3-support bound for the pattern <A> is

$$sbound(< A >,3) = \left\lceil \frac{5.0 \times 6}{(2.0) + (12.0 + 7.0)} \right\rceil = 2$$

**Corollary 3.** *wbound(P, l)* increases monotonically, and accordingly *sbound(P, l)* decreases monotonically as $l$ increases.

Let the upper limit of the length of possible weighted frequent patterns be known as $u$. By Corollary 3, the min support bound of $P$ is the $u$-support bound of $P$,

$$sbound(P, +) = sbound(P, u) \quad (9)$$

By Equation (9) along with Corollary 2, if $scount(P) \ge sbound(P, u)$, then $P$ is feasible. On the contrary, if $scount(P) < sbound(P, u)$, then $P$ is not feasible. This means that we do not need to calculate $l$-support bounds of $P$ for $k < l < u$. Therefore, the pruning algorithm *Pruning-MSB* is more efficient than *Pruning-SB*.

**Corollary 4.** For any $p_i$ in $P=<p_1, p_2, \ldots, p_k>$, $wbound(P-\{p_i\}, l) \ge wbound(P, l)$, and accordingly $sbound(P-\{p_i\}, l) \le sbound(P, l)$.

**Proof.** $wbound(P-\{p_i\}, l)$ is the sum of the vertex weights of $P$ excluding $p_i$ and $(l-k+1)$ greatest vertex weights among the vertices including $p_i$. This sum is always greater than or equal to the sum of the vertex weights of $P$ and $(l-k)$ greatest vertex weights.

**Lemma 2.** There is the full downward closure property among feasible patterns. That is, if a $k$-pattern $P=<p_1, p_2, \ldots, p_k>$ is feasible, then the two $(k-1)$-subpatterns $P_a=<p_1, p_2, \ldots, p_{k-1}>$ and $P_b=<p_2, p_3, \ldots, p_k>$ are also feasible.

**Proof.** The *if* condition means $scount(P) \ge sbound(P, u)$. For $P_a$, $scount(P_a) \ge scount(P)$ by Property 2, and $sbound(P_a, u) \le sbound(P, u)$ by Corollary 4. Therefore $scount(P_a) \ge sbound(P_a, u)$, which implies $P_a$ is feasible. This is similar for $P_b$.

Therefore, the candidate generation algorithm *Gen-FDC* or *Gen-SQL* can be applied.

**Example.**

From the Fig. 1 and 2, we will show how the weighted frequent patterns are generated from the traversal database. Suppose the *minwsup* (minimum weighted support threshold) is 5.0.

1. In the upperLimit() subroutine, the algorithm will scan the length of traversals, and returns the maximum length, which is 4 in this example. The maximum length is the upper limit of the length of weighted frequent patterns.

2. During the initialization step, the candidate patterns of length 1 are generated with all vertices of the base graph.

$C_1$ = {<A>, <B>, <C>, <D>, <E>, <F>}

3. The algorithm repeats as followings.

| Pattern *(P)* | *scount (P)* | *sbound (P)* | *wbound(P,4) / sbound(P,4)* | Weight frequent | Feasible |
|---|---|---|---|---|---|
| <A> | 4 | 15 | 27 / 2 | | ✓ |
| <B> | 3 | 6 | 30 / 1 | | ✓ |
| <C> | 4 | 5 | 30 / 1 | | ✓ |
| <D> | 1 | 5 | 30 / 1 | | ✓ |
| <E> | 3 | 8 | 29 / 2 | | ✓ |
| <F> | 1 | 3 | 30 / 1 | | ✓ |

Candidates for next pass are generated by *Gen-B* or *Gen-C*.

| Pattern *(P)* | *scount (P)* | *sbound (P)* | *wbound(P,4) / sbound(P,4)* | WF | F |
|---|---|---|---|---|---|
| <A, B> | 1 | 5 | 26 / 2 | | |
| <A, C> | 2 | 4 | 27 / 2 | | ✓ |
| <B, C> | 2 | 3 | 30 / 1 | | ✓ |
| <B, D> | 0 | - | - | | |
| <C, E> | 3 | 3 | - | ✓ | ✓ |
| <D, F> | 0 | - | - | | |
| <E, D> | 1 | 3 | 29 / 2 | | |
| <E, F> | 1 | 2 | 29 / 2 | | |

In the above table, '-' denotes 'not need'.

| Pattern *(P)* | *scount (P)* | *sbound (P)* | *wbound(P,4) / sbound(P,4)* | WF | F |
|---|---|---|---|---|---|
| <A, C, E> | 1 | 3 | 25 / 2 | | |
| <B, C, E> | 2 | 2 | - | ✓ | ✓ |

The weighted frequent patterns are {<C, E>, <B, C, E>}.

## 5.2 Estimation by Reachable Vertices

To prune unnecessary candidates as many as possible, the support bounds need to be estimated as high as possible. It means that we must estimate the weight bounds as low as possible. The previous method, however, has a tendency to over-estimate the weight bounds. This tendency is mainly due to the non-consideration of the topology of base graph. Specifically, the vertices with greatest weights are chosen one after one, even though they can not be reached from the corresponding pattern.

**Definition 10.** Given a base graph $G$, *k-reachable vertices* from a vertex $v$ is all the vertices reachable from $v$ within the distance $k$.

Such $k$-reachable vertices can be regarded as the vertices within the radius $k$ from $v$. Therefore, $k$-reachable vertices include all the $(k-1)$-reachable vertices.

Given a $k$-pattern $P$, let $R(P, l)$, $k < l \le u$, be the $(l-k)$-reachable vertices from the head vertex of $P$, but not in $P$ and not through the vertices in $P$. They can be obtained by a level wise manner.

**Algorithm. *R(P, l)***

$T$ = {head vertex of $P$} for $l = k+1$, $N$ otherwise;
$N = \varnothing$;
for each vertex $v$ in $T$
  for each edge <$v$, $w$> in $G$
    if $w$ is not in $P$ and $R(P, l-1)$ and $N$, then append $w$ to $N$;
$R(P, l) = R(P, l-1) \cup N$

For example, from Fig. 1, $R(<A>, 2)$ is {B, C}, and $R(<A>, 3)$ is {B, C, D, E}.

Among the vertices in $R(P, l)$, let the vertices with the $(l - k)$ greatest weights be $v_{r1}, v_{r2}, \ldots, v_{rl-k}$. Then, the $l$-weight bound, *wbound(P, l),* and the $l$-support bound, *sbound(P, l)*, of $P$ are obtained by Equation (7) and (8), respectively.

For example, refer to Fig. 1 and Fig. 2, the 3-support bound for the pattern <A> is

$$sbound(<A>,3) = \left\lceil \frac{5.0 \times 6}{(2.0) + (7.0 + 6.0)} \right\rceil = 2$$

**Corollary 3'.** *wbound(P, l)* increases monotonically, and accordingly *sbound(P, l)* decreases monotonically as $l$ increases.

By Corollary 3', the min support bound of $P$ is the $u$-support bound of $P$,

$$sbound(P, +) = sbound(P, u) \qquad (9')$$

In spite of Equation (9'), however, the pruning algorithm *Pruning-SB* may be more efficient than *Pruning-MSB* because the pruning can be decided before $u$ due to the level wise characteristic of the algorithm *R(P, l)*.

**Corollary 4'.** For $p_k$ in $P=<p_1, p_2, \ldots, p_k>$, $wbound(P-\{p_k\}, l) \ge wbound(P, l)$, and accordingly $sbound(P-\{p_k\}, l) \le sbound(P, l)$.

**Proof.** $wbound(P-\{p_k\}, l)$ is the sum of the vertex weights of $P$ excluding $p_k$ and $(l-k+1)$ greatest vertex weights among the vertices of $R_{l-k+1}(P-\{p_k\})$ which includes all the vertices of $R_{l-k}(P)$ and $p_k$. This sum is always greater than or equal to the sum of the vertex weights of $P$ and $(l-k)$ greatest vertex weights among the vertices of $R_{l-k}(P)$.

**Lemma 2'.** There is the partial downward closure property among feasible patterns. That is, if a $k$-pattern $P=<p_1, p_2, \ldots, p_k>$ is feasible, then the $(k-1)$-subpattern $P_a=<p_1, p_2, \ldots, p_{k-1}>$ is also feasible.

**Proof.** The *if* condition means $scount(P) \ge sbound(P, u)$. For $P_a$, $scount(P_a) \ge scount(P)$ by Property 2, and $sbound(P_a, u) \le sbound(P, u)$ by Corollary 4'. Therefore $scount(P_a) \ge sbound(P_a, u)$, which implies $P_a$ is feasible.

Therefore, the candidate generation algorithm *Gen-PDC* can be only applied.

**Example.**

| Pattern *(P)* | *scount (P)* | *sbound (P)* | *wbound(P,l) / sbound(P,l)* | | | WF | F |
|---|---|---|---|---|---|---|---|
| | | | $l = 2$ | $l = 3$ | $l = 4$ | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| <A> | 4 | 15 | 9 / 4 | - | - | | ✓ |
| <B> | 3 | 6 | 12 / 3 | - | - | | ✓ |
| <C> | 4 | 5 | 11 / 3 | - | - | | ✓ |
| <D> | 1 | 5 | 18 / 2 | × | × | | |
| <E> | 3 | 8 | 16 / 2 | - | - | | ✓ |
| <F> | 1 | 3 | × | × | × | | |

In the above table, '×' denotes 'not applicable'.
Candidates for next pass are generated by *Gen-A*.

| Pattern *(P)* | *scount (P)* | *sbound (P)* | *wbound(P,l) / sbound(P,l)* | | WF | F |
|---|---|---|---|---|---|---|
| | | | $l = 3$ | $l = 4$ | | |
| <A, B> | 1 | 5 | 14 / 3 | 26 / 2 | | |
| <A, C> | 2 | 4 | 13 / 3 | 27 / 2 | | ✓ |
| <B, C> | 2 | 3 | 16 / 2 | - | | ✓ |
| <B, D> | 0 | - | - | - | | |
| <C, E> | 3 | 3 | - | - | ✓ | ✓ |
| <E, D> | 1 | 3 | 22 / 2 | × | | |
| <E, F> | 1 | 2 | × | × | | |

| Pattern *(P)* | *scount (P)* | *sbound (P)* | *wbound(P,l) / sbound(P,l)* | WF | F |
|---|---|---|---|---|---|
| | | | $l = 4$ | | |
| <A, C, E> | 1 | 3 | 25 / 2 | | |
| <B, C, E> | 2 | 2 | - | ✓ | ✓ |
| <C, E, D> | 1 | 2 | 29 / 2 | | |
| <C, E, F> | 1 | 2 | × | | |

| Pattern *(P)* | *scount (P)* | *sbound (P)* | WF | F |
|---|---|---|---|---|
| <B, C, E, D> | 0 | - | | |
| <B, C, E, F> | 1 | 2 | | |

The weighted frequent patterns are {<C, E>, <B, C, E>}.

## 6. Experimental Results

This section presents experimental results of the mining algorithm, and compares two estimation algorithms, *All vertices* and *Reachable vertices*, using synthetic dataset. The algorithms are implemented with C++ language running under Microsoft Visual C++ 6.0. All experiments are performed on 3.0 GHz Pentium Ⅳ PC machine with 1 GB memory. The operating system is Windows XP Professional and the database server is Microsoft SQL Server 2000 for managing base graphs and traversals on them.

During the experiment, base graph is generated synthetically according to the parameters, i.e., number of vertices and average number of edges per vertex. And then, we assigned distinctive weight to each vertex of the base graph. All the experiments use a base graph with 100 vertices and 300 edges, i.e., 3 average edges per vertex. The number of traversals is 10,000 and the minimum weighted support is 2.0. We generated six sets of traversals, in each of which the maximum length of traversals varies from 5 to 10.

Fig. 4 shows the trend of the number of feasible patterns with respect to the max length of traversals. We measured the number of feasible patterns when the length of candidate patterns is (max length of traversals – 1). As shown in the figure, the number of feasible patterns for *Reachable vertices* is smaller than that of *All vertices*. The difference of the number of feasible patterns between two estimation algorithms becomes smaller as the max length of traversals increases.

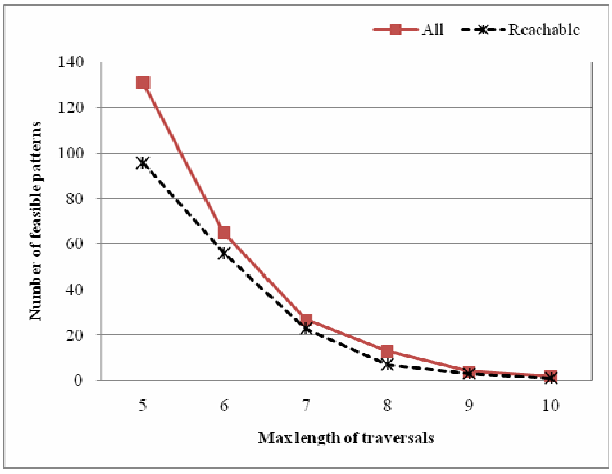


Fig. 4. Number of feasible patterns w.r.t diferrent max length of patterns

Fig. 5 shows the trend of the execution time with respect to the max length of traversals. As shown in Fig. 5, when the max length of traversals is short, *Reachable vertices* is more efficient than *All vertices*. When the max length of traversals increases, however *Reachable vertices* is less efficient. The performance difference becomes larger when the max length of traversals becomes longer. This is because *Reachable vertices* spends more time to find reachable vertices as the max length of traversals increases.

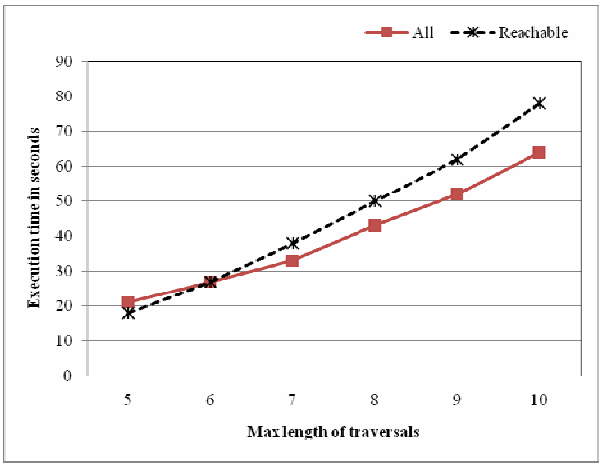


Fig. 5. Execution time w.r.t diferrent max length of patterns

## 7. Conclusions

This paper extended the mining problem to the discovering of weighted frequent patterns from path traversals on weighted graph. Differently from previous approaches, vertices of graph are attached with weights which reflect their importance. With this weight setting, we presented the mining algorithm which takes the weights into account in the measurement of support. This algorithm is based on the notion of support bound. We also proposed two methods for the estimation of support bound, and then experimented on them. We are currently working on the applications such as Web mining.

## References

[1] M.S. Chen, J.S. Park and P.S. Yu, "Efficient Data Mining for Path Traversal Patterns", IEEE Trans. on Knowledge and Data Engineering, vol. 10, no. 2, pp. 209-221, Mar. 1998.

[2] A. Nanopoulos and Y. Manolopoulos, "Finding Generalized Path Patterns for Web Log Data Mining", Proc. of East-European Conf. on Advanced Databases and Information Systems (ADBIS), Sep. 2000.

[3] A. Nanopoulos and Y. Manolopoulos, "Mining Patterns from Graph Traversals", Data and Knowledge Engineering, vol. 37, no. 3, pp. 243-266, Jun. 2001.

[4] C.H. Cai, W.C. Ada, W.C. Fu, C.H. Cheng and W.W. Kwong, "Mining Association Rules with Weighted Items", Proc. of International Database Engineering and Applications Symposium (IDEAS), UK, Jul. 1998.

[5] W. Wang, J. Yang and P.S. Yu, "Efficient Mining of Weighted Association Rules (WAR)", Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD), USA, Aug. 2000.

[6] F. Tao, F. Murtagh and M. Farid, "Weighted Association Rule Mining using Weighted Support and Significance Framework", Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD), USA, Aug. 2003.

[7] U. Yun and J.J. Leggett, "WLPMiner: Weighted Frequent Pattern Mining with Length-Decreasing Support Constraints", Proc. of Pacific-Asia International Conference on Knowledge Discovery and Data Mining (PAKDD), Vietnam, May 2005.

**Seong Dae Lee** received the M.S. degree in Computer Engineering from Korea Maritime University in 2001. He is working for a Ph.D. His research interests include Database, UML, and Data Mining.

**Hyu Chan Park** received the M.S. and Ph.D. in Computer Engineering from Korea Advanced Institute of Science and Technology in 1987 and 1995. He has been an Associate Professor in Korea Maritime University since 1997. His research interests include Database, Marine GIS, and Data Mining.