ICIC Express Letters
Volume **11**, Number **1**, January **2017**

ICIC International ©2017 ISSN 1881-803X
pp. **213–219**

MINING WEIGHTED PATTERNS FROM SOFTWARE DYNAMIC CALL GRAPH

Haitao He[1,2], Jiandi Wang[1,2,*], Hao Wang[1,2], Ruiling Zhang[1,2] and Jiadong Ren[1,2]

[1]College of Information Science and Engineering
[2]The Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province
Yanshan University
No. 438, West Hebei Ave., Qinhuangdao 066004, P. R. China
{ haitao; jdren }@ysu.edu.cn; *Corresponding author: wjdysu@sina.com

Received June 2016; accepted September 2016

Abstract. *As scale of software system and complexity of its structure are increasing, it is inevitable to study the behavioral characteristics of software to guarantee software safety and reliability. Finding representative patterns can help program maintainers detect the exception and improve work efficiency. In this paper, an efficient method called PSWP-Miner (projected-database based software weighted pattern mining) is proposed to mine weighted patterns from software dynamic call graph. In the algorithm, we firstly define the weight of each event in the software dynamic call graph. Secondly, a novel structure called sequence identifier table (SIT) is put forward. It stores patterns and corresponding sequences identifiers information which contributes to reducing search space when constructing projected database. Moreover, an effective upper bound model is proposed to get more accurate weight of the pattern and pruning strategy is designed in the algorithm to reduce number of candidates in the recursive mining process. At last, we conduct experiments on both real and synthetic datasets. The experiment result demonstrates impressive performance.*

Keywords: Software dynamic call graph, Weighted pattern, Software behavior

1. **Introduction.** Like hardware failure, software failure can also lead to severe and even fatal consequences [1]. Software behavior learning becomes a popular research in software trustworthiness [2]. The function calling sequences in a scenario are represented as a labeled, directed acyclic graph [3]. As a result, it is desirable to mine representative software behavior patterns from software dynamic call graph which are important to software maintenance and failure detection.

Software systems are composed of many interacting elements. A natural way to abstract over software systems is to model them as graphs. [4] proposed a generative model of software dependency graphs which synthesizes graphs whose degree distribution is close to the empirical ones observed in real software systems. This model gave us novel insights on the potential hidden rules of software evolution. [5] analyzed the dynamic function call graph for the purpose of software fault defection. Such a software behavior graph reflects the invocation structure of a particular program execution and we can learn software behavior by analyzing the software call graph.

Park [6] found traversal patterns from graph traversals based on the graph structure of the model. Therefore, the problem of finding representative patterns from graphs is converted into finding frequent patterns from sequential database. We can use sequence pattern mining algorithm to address software reliability issue. Lo et al. [7] proposed a method to classify software behaviors based on past history or runs. Li et al. [8] came up with a method which used the pattern position distribution as features to detect software failure occurring through misused software patterns. One of the limitations of the above

approaches for mining frequent patterns is that all items were treated uniformly. In many cases, the item has different importance.

The most commonly used software quality assurance method is software testing. Customarily, testing a software system involves a large set of test cases. The large number of program execution traces that we thus obtain gives rise to a representative pattern mining problem. Based on the above issues, a new algorithm PSWP-Miner has been designed to mine weighted patterns from software dynamic call graph. In the method we propose a new data structure called SIT (sequence identifier table) which is used to store the pattern and the sequence identifier. By this structure, the mining procedure only needs to scan the sequences in SIT without scanning the whole database, so the mining efficiency can be improved. Moreover a new upper-bound model and projection based pruning strategy are integrated into the algorithm. We evaluate the weight of the pattern through the upper-bound model to avoid unnecessary pattern calculation. As a result, the candidates number and memory consumption are reduced greatly. At last, we evaluate our algorithm on a set of experiments.

The remaining paper is organized as follows. Section 2 gives the definitions. The proposed algorithm PSWP-Miner with the sequence identifier table and pruning strategy is stated in Section 3. Sections 4 and 5 present the experiments and conclusion respectively.

2. **Problem Statement and Preliminaries.** The dynamic call graph G is a directed graph that represents calling relationships between program. Specifically, the nodes in G represent functions. We can traverse G by Depth-First way to get function call path S from root to leaf. These paths constitute a database $SDB = \{S_1, S_2, \ldots, S_n\}$.

Definition 2.1. *The weight of a node in G is denoted as $w(v, G)$. The weight value presents the importance of the function in G. It can be defined as follows.*

$$w(v, G) = \sum_{i=1}^{n} \frac{w(v_i v)}{w_{out}(v_i)} w(v_i, G) \tag{1}$$

$w(v_i v)$ is the weight of edge $v_i v$. $w_{out}(v_i)$ is the outdegree and it is defined as follows.

$$w_{out}(v_i) = \sum_{j=1}^{m} w(v_i, u_j) \tag{2}$$

Definition 2.2. *Software Traverse Path is a sequence of consecutive nodes from the root node to leaf node in a dynamic call graph. It can be represented as $S =< f_{root}, f_{root+1}, \ldots, f_{leaf} >$.*

Definition 2.3. *A pattern $P =< e_1, e_2, \ldots, e_n >$ is considered as a subsequence of STP $S =< s_1, s_2, \ldots, s_m >$ if there exist integers $1 \leq i_1 < i_2 < i_3 < i_4 \ldots < i_n \leq m$ where $e_1 = s_{i_1}$, $e_2 = s_{i_2}, \ldots, e_n = s_{i_n}$.*

Definition 2.4. *The weight value of a pattern P in a sequence S is defined as follows.*

$$w_P = \frac{\sum_{i \in P \wedge P \subseteq S}^{|P|} w_i}{l_P} \tag{3}$$

where l_P is the number of items in pattern P.

Definition 2.5. *The actual weight support value of a pattern P is $awsup_p = w_P \times$ scount, where scount is the number of sequences containing the pattern P.*

Definition 2.6. *If $awsup_P$ of a pattern P is no less than minimum weighted support threshold (shorted as minwsup), P is called a weighted pattern.*

Definition 2.7. *The weighted upper bound of a pattern P $wubs_P$ is the sum of maximum weights of the sequence including P in a sequence database. That is,*

$$wubs_p = \sum_{p \subseteq Seq_y \wedge Seq_y \subseteq SDB} mws_y \tag{4}$$

Lemma 2.1. *The weighted upper bound of a pattern keeps the downward-closure property.*

Proof: Let p be a weighted upper-bound pattern and d_p be the set of sequences containing p in a sequence database. If p is a subsequence of $p^{'}$, then $p^{'}$ cannot exist in any sequence where p is absent. Therefore, the weighted upper bound $wubs_p$ of p is the maximum upper-bound of weight value of $p^{'}$. Accordingly, if $wubs_p$ is less than minimum weighted support threshold, then $p^{'}$ cannot be a weighted upper-bound pattern. □

Definition 2.8. *A pattern p is called weighted upper bound pattern if $wubs_p$ is no less than minwsup.*

Lemma 2.2. *The weighted sequential patterns WSP_s is the subset of weighted upper bound patterns $WUBP_s$.*

Proof: For a pattern p, if p is the weighted sequential pattern, $awsup_p$ is no less than minimum weighted threshold. The $wubs_p$ is the upper bound of p and $awsup_p$ is less than $wubs_p$. In this case, we can conclude that $wubs_p$ is bigger than minimum weighted threshold, so p is also the element of $WUBP_s$. □

3. **The PSWP-Miner Algorithm.** In this section, we describe the sequence identifier table, the pruning strategy and the PSWP-Miner at length.

3.1. **The sequence identifier table (SIT).** The SIT structure contains two fields: the pattern and the identifiers of sequences containing the pattern. By using the efficient structure, the proposed algorithm only needs to scan the sequences in SIT without scanning the whole database during the constructing projected database procedure. Take the item c in Figure 1(a) as an example. When constructing projected database of c, we only need to scan the Seq3, Seq4, Seq5 which contain item c.

3.2. **Pruning strategy.** The pattern not in weighted upper bound pattern set must not be weighted pattern, so this kind of pattern can be pruned to reduce the memory consumption of the algorithm. The proposed upper bound model holds downward-closure property, so if the pattern is not weighted upper bound pattern, its subpattern cannot be weighted frequent pattern. In this case, the weighted sequential pattern set is the subset of weighted upper bound pattern set, so the pattern not in weighted pattern set must not be weighted pattern.

3.3. **The PSWP-Miner Algorithm.** In PSWP-Miner procedure, it traverses the software dynamic call graph to get the sequence database D in DFS way. Next, it initializes SIT and select the max weight of each sequence in D. Then, it calculates $awsup_I$ and $wubs_I$ of each item I to construct the $WUBS_1$ which is the set of 1-length patterns and outputs the weighted item. At the same time, the sid of sequence containing I is put into SIT. After the prepared work, it begins to prune the item in the sequence not appearing in $WUBS_1$ according to the pruning strategy. When the revised sequence length is less than 2, it will be deleted from database since it cannot generate the 2-pattern. Then for each item I in $WUBS_1$, the algorithm constructs its projected database and calls the procedure Finding-WP.

In the Finding-WP procedure. It first calculates the $wubs_I$ of item I and produces the set $WUBS_P$ which is a set of patterns consisting of prefix p and item I. The pruning strategy is also used to prune the item not in $WUBS_P$. Then, for each pattern in $WUBS_P$, the algorithm constructs its projected database and calls Finding-WP recursively to find

the weighted pattern. However, in procedure Finding-WP, we do not use the SIT structure because of the high memory consumption. It is memory-consuming while the running time is not reduced a lot when integrating SIT in the Finding-WP procedure.

Algorithm 1: PSWP-Miner

Input: software dynamic call graph G, a threshold *minwsup*
Output: weighted frequent patterns
1. Obtain all paths by traversing the graph G in DFS way and store these paths in D;
2. Initialize index table SIT to null;
3. Scan D to select the max weight of each sequence in D;
4. **for** each item I $\in$ D **do**
5. calculate the $wubs_I$ and $awsup_I$ of I;
6. **if** $wubs_I > minutil$
7. $WUBS_1 \leftarrow$ I and put item I and its identifiers of sequences into SIT;
8. **if** $awsup_I > minutil$
9. Output I;
10. **for** each y-th sequence in D
11. Remove the items not in $WUBS_1$;
12. **if** $|Seq_y| < 2$
13. Remove Seq_y from D;
14. **for** each item I in $WUBS_1$
15. Construct PDB_I reference to SIT;
16. Call Finding-WP(I, $awsup_I$, PDB_I);

Procedure: Finding-WP
Input: a prefix p and its weight support $awsup_p$ and its projected database SDB_P;
Output: weighted frequent patterns with p as its prefix pattern;
1. **for** each distinct item I in PDB_P
2. calculate the $wubs_I$ and $awsup_I$ of I;
3. **if** $wubs_I > minwsup$
4. $\mathrm{p}' = \mathrm{p} \cup \mathrm{I}$ and $WUBS_P \leftarrow \mathrm{p}'$;
5. $awsup_{p'} = (awsup_p \times |p| + awsup_I) \div (|P| + 1)$;
6. **if** $awsup_{p'} > minwsup$
7. Output p′;
8. **for** each y-th sequence in PDB_P
9. Remove the items not in $WUBS_P$;
10. **for** each pattern p′ in $WUBS_P$
11. Construct $PDB_{p'}$ and call Finding-WP(p′, $awsup_{p'}$, $PDB_{p'}$);

Example 3.1. *As the instance in Figure 1(a). The minwsup is 0.5. And then we calculate the weighted upper bound value and actual weight support value for each item according to Figure 1(b). Item a, c, d, f, g, h, i will be put into the WUBS1 since their weighted frequent upper bounds are no less than minwsup. Item a, c, f, g are weighted patterns, while the item b and e are deleted from the sequences. The revised sequence Seq1 will be removed since there is only one item a in the sequence. In the next, we will construct the projected database for each item in WUBS1 and call the Finding-WP procedure to mine weighted patterns recursively. The projected database of item a is shown in Figure 1(c).*

SID	Sequences
Seq1	<a,b,e>
Seq2	<a,f,i>
Seq3	<a,c,f,i>
Seq4	<a,c,g,i>
Seq5	<a,c,h>
Seq6	<a,d,h,>

(a) The database

item	weight
a	0.1
b	0.15
c	0.2
d	0.25
e	0.3
f	0.35
g	0.4
h	0.5
i	0.6

(b) The weight of each item

item	*wubs*	*awsup*
c	1.2	0.4
d	0.5	0.25
f	1.2	0.7
h	1.0	1.0
i	1.8	1.8
g	0.6	0.4

(c) The projected database of item a

FIGURE 1. The instance

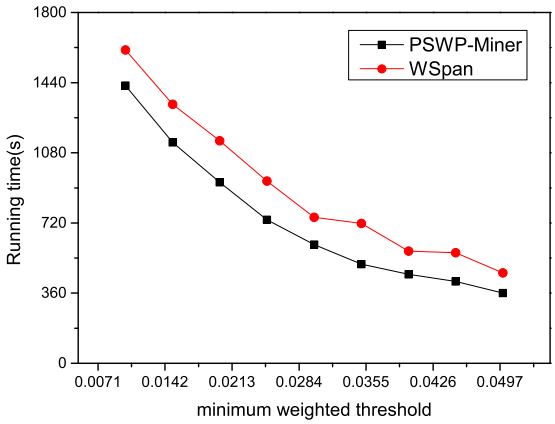

FIGURE 2. Running time on cflow

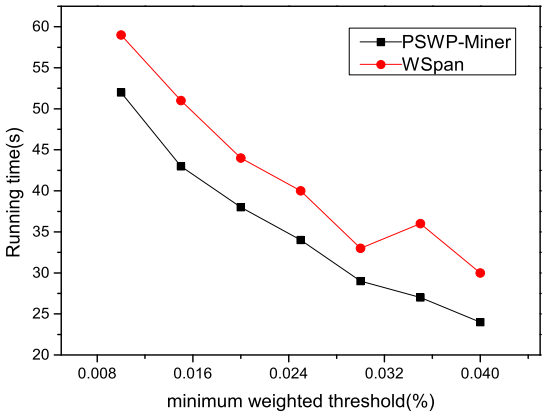

FIGURE 3. Running time on T1014D100K

4. **Experiment.** In this section, PSWP-Miner is compared with WSpan for the running time and pruning efficiency. We have done extensive experiments on cflow and T1014D100K [9]. In order to obtain relationships of function calls when software executing, we track process of software executing and map relationships of function calls to software dynamic call graph. We get experiment datasets cflow with the help of pvtrace, Gephi and Graphviz on Linux.

4.1. **Running time.** We test the executing time of algorithm PSWP-Miner and WSpan in various support thresholds shown in Figure 2 and Figure 3. In Figure 4 the number of traversal sequences varies from 20000 to 100000. The result shows that PSWP-Miner is faster than WSpan. This is mainly due to the fact that PSWP-Miner adopts SIT structure and the pruning strategy. The time consuming is greatly decreased.

Figure 5 shows the running time with SIT structure. We can see from the result that the running time is decreased after the algorithm adopts SIT structure. When constructing the projected database, the algorithm reference to the identifiers stored in the structure. As a result, excessive database scan can be avoided and the search space can be reduced by this structure.

4.2. **Pruning efficiency.** As Figure 6 and Figure 7 shown, the number of candidates number is decreasing by threshold decreasing and the difference between them is becoming large. It could be observed that the number of candidates generated by PSWP-Miner is always less than WSpan. The reason for this is that the PSWP-Miner adopts WUBS upper bound model to obtain accurate upper-bounds of sequence weight for sequences in the recursive mining process. The maximal weight in sequence is more suitable as

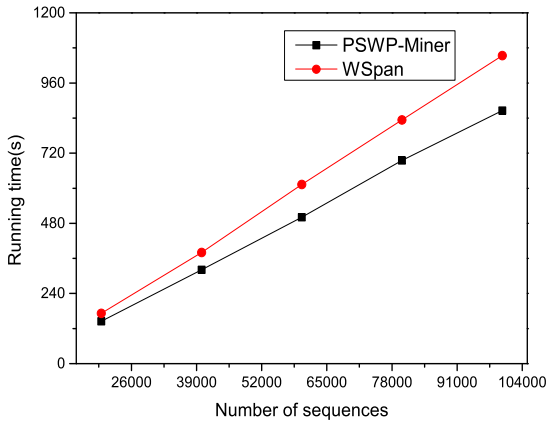

FIGURE 4. Running time on cflow with different data-size

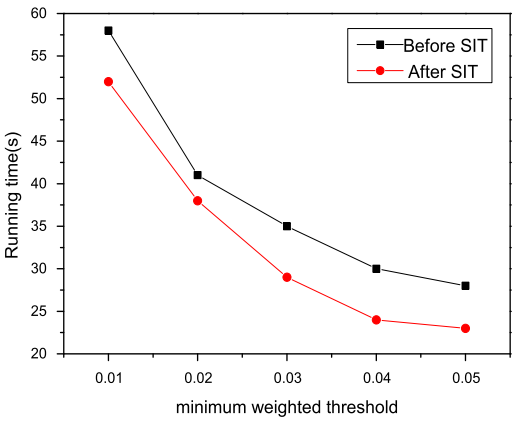

FIGURE 5. Running time on T1014D100K with SIT

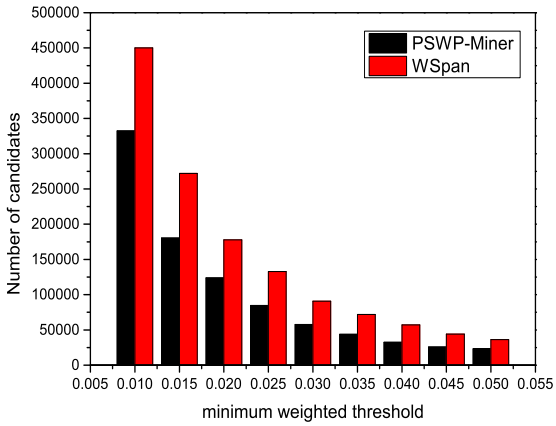

FIGURE 6. Number of candidates on cflow

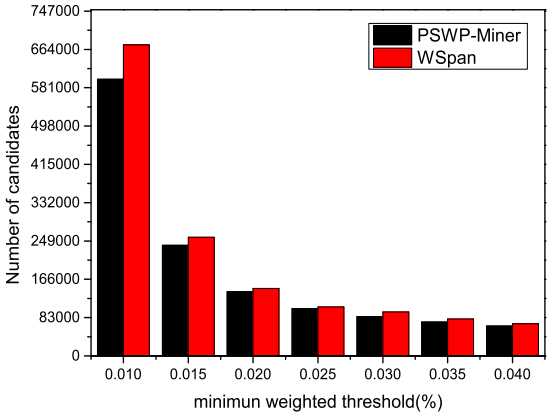

FIGURE 7. Number of candidates on T1014D100K

upper bound of any subsequence in a sequence when compared with WSpan algorithm. Accordingly, a large number of unweighted candidates could be pruned effectively.

5. **Conclusions.** In this paper, we propose an efficient algorithm PSWP-Miner for mining weighted patterns from software dynamic call graphs. A new structure SIT is integrated into the algorithm to reduce the search space when constructing projected database. Meanwhile, an effective upper bound model is designed to get more accurate weight of the pattern and pruning strategy is designed in the algorithm to reduce number of candidates when constructing projected database. We have demonstrated our proposed algorithm on the synthetic and real datasets and the experimental results show the number of weighted frequent upper bound patterns is obviously less than that required by the WSpan algorithm, and the proposed algorithm outperforms the WSpan algorithm in terms of execution efficiency. In the future, we will optimize the structure of the algorithm to reduce the memory consumption of the algorithm.

Acknowledgment. This work is supported by the National Natural Science Foundation of China under Grant No. 61572420, No. 61472341 and the Natural Science Foundation of Hebei Province P. R. China under Grant No. F2013203324, No. F2014203152 and No. F2015203326.

REFERENCES

[1] Y. Shi, M. Li, S. Arndt et al., Metric-based software reliability prediction approach and its application, *Empirical Software Engineering*, pp.1-55, 2016.

[2] B. Zhao, Y. Wang, Z. Shan et al., Software behavior model based on functional slicing, *International Conference on Intelligent Systems Research and Mechatronics Engineering*, 2015.

[3] T. T. Nguyen et al., Graph-based mining of multiple object usage patterns, *ACM SIGSOFT*, 2009.

[4] V. Musco, M. Monperrus and P. Preux, A generative model of software dependency graphs to better understand software evolution, *Eprint Arxiv*, 2014.

[5] W. Masri, Fault localization based on information flow coverage, *Software Testing Verification and Reliability*, vol.20, no.20, pp.121-147, 2009.

[6] H. C. Park, Mining weighted-frequent traversal patterns using graph topology, *International Journal of Computer Science and Network Security*, 2014.

[7] D. Lo, H. Cheng, J. Han et al., Classification of software behaviors for failure detection: A discriminative pattern mining approach, *ACM SIGKDD*, pp.557-566, 2009.

[8] C. Li, Z. Chen, H. Du et al., Using pattern position distribution for software failure detection, *International Journal of Computational Intelligence Systems*, vol.6, no.2, pp.234-243, 2013.

[9] *Frequent Itemset Mining Dataset Repository*, http://fimi.ua.ac.be/, 2012.