Extracting Feature Sequences in Software Vulnerabilities Based on Closed Sequential Pattern Mining

Qunhui Wu
State Key Lab of Software Development Environment, Beijing University of Aeronautics and Astronautics, Beijing, P.R.China
Email: wuqunhui126com@126.com

Shilong Ma
State Key Lab of Software Development Environment, Beijing University of Aeronautics and Astronautics, Beijing, P.R.China

Hao Wang
NARI Group Corporation, State Grid Electric Power Research Institute, Beijing, P.R.China

Abstract—Feature Extraction is significant for determining security vulnerabilities in software. Mining closed sequential patterns provides complete and condensed information for non-redundant frequent sequences generation. In this paper, we discuss the feature interaction problem and propose an efficient algorithm to extract features in vulnerability sequences. Each closed sequential pattern represents a feature in software vulnerabilities. We explore how to efficiently maintain closed sequential patterns in vulnerability sequences. A compact structure WClosedTree is designed to keep closed sequential patterns, and its nice properties are carefully studied. Two main pruning strategies, backwards super pattern condition and equivalent position information condition, are developed to remove frequent but non-closed sequential patterns in WClosedTree. During the process of maintaining WClosedTree, the weight metric of each feature sequence is calculated to better meet the needs of decision makers. Thus, the proposed algorithm can efficiently extract features from vulnerability sequences. The experimental results show that the proposed algorithm significantly improves the runtime efficiency for mining closed sequential patterns, and the feature interaction framework implements feature extraction in software vulnerabilities.

Index Terms—feature extraction, closed sequential pattern, vulnerability sequence, software vulnerability

I. INTRODUCTION

With the ever-increasing number of hacker attacks and the spread of the worms on the Internet, information security now becomes the hot topic within both research and industry fields. The software vulnerabilities existing in the computer systems are the most important core issue of information security, which a malicious attacker can use to enhance competence, visit unauthorized resources, or even destroy sensitive data. So analyzing software vulnerabilities is critical for information security. Otherwise, feature extraction is an important element when analyzing software vulnerabilities [1, 2, 3, 4].

In order to make the complexity of modern software systems manageable their functionality is increasingly being decomposed into features. Feature extraction is an effective method for fault diagnosis. M. Pistoia [5] described static analysis methods to identify security vulnerabilities in software systems, which summarized security features, libraries and interfaces among various typical systems. A. Nhlabatsi [6] concluded the feature interaction problem and its possible implications for security requirements. In process planning (MASCAPP) systems [7], A. Nassehi discussed some simple types of feature interactions, though STEP-NC data are not considered in its final process plan. Moreover, feature interactions are considered in two different categories, geometric and technological interactions in Liu and Wang’s work [8]. Otherwise, the feature interaction method was presented in both 3D and feature interaction graph format in a developed software system [9]. G. C. Rui [10] generated hypothetical features for feature interaction detection, which is supported by sets of all possible events, predicates and inconsistent behaviors.

Vulnerability is the weakness in a software system that can be exploited to compromise system’s security. The mechanism of extracting the feature sequence of the software vulnerability can be described as a process. First, vulnerability models are loaded into memory and the data structures of these models are established. Second, operation sequences of vulnerability models are extracted through abstracting software vulnerabilities and filtering independent operations. Finally, feature sequences are extracted according to the information of operation sequences. In this paper, when extracting operation sequences of vulnerability models, each program module is defined as fundamental function units. Consequently, the operation sequences of program modules are
considered as the process of invoking software vulnerabilities. An operation sequence is denoted formally as a sequence of strings.

According to the above presentation, we introduce the method of mining closed sequential pattern to extract features of software vulnerabilities. So mining closed sequential patterns becomes a crucial task in feature sequences extraction. There exist many algorithms for mining closed sequential pattern, such as CloSpan [1], BIDE [12], CMP-Miner [13], BIDE-Margin [14], Stream_FCI [15], IWFP_wi and IWFP_fd [16] and so on. In CloSpan, it first generates a set of closed candidates stored in a hash indexed tree and then performs post-pruning on it. Because CloSpan needs to maintain the set of closed candidates, it will consume a large amount of memory for closure checking. In BIDE, this mines closed sequential patterns without maintaining candidates, which adopts forward and backward directional extension checking to perform closure checking and to prune the redundant patterns in the mining process. The CMP-Miner enumerates the frequent patterns by a frequent itemsets so as to avoid the time-consuming. IWFP_wi and IWFP_fd are effective for incremental and interactive WFP mining using a single database scan. When a database is updated or a minimum support threshold is changed, IWFP_wi and IWFP_fd are effective for incremental and interactive mining to utilize the current tree structure and to use the previous mining results. To our knowledge, this paper is the first one to apply the closed sequential pattern mining theory to feature extraction in software vulnerabilities.

In this paper, we propose an efficient algorithm called WCSMining (Weighted Closed Sequential pattern Mining) to extract features in vulnerability sequences where each closed sequential pattern represents a feature. To maintain closed sequential patterns incrementally, WClosedTree is proposed to keep the closed sequential patterns and other auxiliary information, and the nice properties of WClosedTree are investigated. Moreover, we apply two closure checking strategies to update node information during the mining process, and the weight metric of each feature sequence is calculated to meet the needs of decision makers more closely.

The remaining of the paper is organized as follows. Section II describes the problem definitions. The feature extraction model and our proposed algorithm are discussed in detail in Section III. Section IV illustrates experimental results and performance analysis. Finally, conclusion will be given in Section V.

II. PROBLEM DEFINITION

In order to better extract feature sequence, the software vulnerabilities are denoted formally as sequences of strings as follows: Let $L = \{l_1, l_2, ..., l_n\}$ be a set of literals called items. A sequence $S = \langle a_1, a_2, ..., a_n \rangle$ is an orderly list of items, such that each item $a_i \in L$. Let the sequences database $SDB$ be a set of operation sequences.

In a sequence $S$, each item $a_i$ represents a fundamental function operation of software vulnerability. Sequence $S$ stands for an operation sequence of vulnerability model, and the sequences database $SDB = \{S_1, S_2, ..., S_m\}$ is a set of all operation sequences. Each operation sequence in sequences database $SDB$ includes a sequence identifier (Sid) and a vulnerability sequence.

We say that the sequence $S_1 = \langle a_1, a_2, ..., a_n \rangle$ supports the sequence $S_2 = \langle b_1, b_2, ..., b_m \rangle$ if there exist integers $1 = i_1 < i_2 < ... < i_m = m$, such that $a_{i_1} = b_1, a_{i_2} = b_2, ..., a_{i_m} = b_m$, then $S_1$ is called a sub-sequence of the sequence $S_2$ or $S_2$ is super-sequence of sequence $S_1$. The support of sequence $S_1$ is denoted by $sup(S_1)$, it is the number of sequence in $SDB$ that contain $S_1$. Given a support threshold $min-sup$, a sequence $S_1$ is a frequent sequence in $SDB$, then $sup(S_1) \geq min-sup$. If a sequence $S_1$ is frequent and there exists no proper super-sequence with the same support, we call it a frequent closed sequence.

Definition 1. Item position ($IPos$). The positional information of each item $a_i$ in vulnerability sequence $S = \langle a_1, a_2, ..., a_n \rangle$ is denoted as $IPos (Sid, itemID)$, where $Sid$ is the sequence identifier, $itemID$ is the position identifier.

Definition 2. Backwards super pattern. Given two sequences $S_1$ and $S_2$, if $S_2$ is a super-sequence of $S_1$, i.e. $S_1$ is generated before $S_2$, otherwise $S_1$ and $S_2$ have the same hash key and the last item, then $S_2$ is backwards super pattern of $S_1$, where hash key is composed of the support, the total number of Sids and the total number of itemIDs.

Definition 3. Equivalent position information. Given two sequences $S_1$ and $S_2$, except for the same hash key and the last item, and there exists no inclusion relation between $S_1$ and $S_2$, then $S_1$ and $S_2$ have the equivalent position information.

Definition 4. The weight for item $a_i$ is defined as

$$weight(a_i) = \sum_{j=1}^{md} weight(S_j) * a_i \cdot \log \frac{\sum_{j=1}^{md} weight(S_j) * a_i}{\sum_{j=1}^{md} weight(S_j) * |S_j|}.$$ 

The details for these entries are as follows:

- $\{a_i\}$ is the number of item $a_i$ in sequences $S_j$.
- $\{a_i\}$ is a set of items, each item $a_i$ of sequences $S_j \in \{a_1, a_2, ..., a_n\}$.
- $weight(S_j)$ is the weight of sequence $S_j$, where $weight(S_j) = \sum_{a_i} weight(a_i)$. The greater the value of $weight(S_j)$ is, the greater the representation degree is for $S_j$.
- $S_{id}^total$ is the total number of sequences $S_j$ in sequences database $SDB$.
These vulnerability sequences are deposited in the database. We define 4, the corresponding weights for items are obtained in the same way. Otherwise, according to (1, 0), position information for other items can be obtained.

Example 1. Assume that software vulnerabilities have been denoted formally as sequences of strings, given sequence identifier Sid to each vulnerability sequence, these vulnerability sequences are deposited in the sequences database SDB.

Definition 5. Each node of multi-branches tree WCmedTree is defined as a tuple, \( Tnode = [a_i, sup(S_i), weight(S_i)] \). The details for these entries are as follows:
- \( a_i \) is an item which constitutes prefix of closed sequence.
- \( sup(S_i) \) is denoted as the corresponding closed sequence in the frequency of sequences database SDB.
- \( weight(S_i) \) is the weight of closed sequential pattern, where \( S_i \) is denoted as the path from root node to the current node, if \( sup(S_i) \) is greater than 0, \( S_i \) is a weighted closed sequential pattern.

Table I. Sequence database SDB of software vulnerabilities

<table>
<thead>
<tr>
<th>Sid</th>
<th>Vulnerability sequence</th>
<th>Weight of sequence ( S_{Sid} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>C A A B C D</td>
<td>0.5</td>
</tr>
<tr>
<td>S2</td>
<td>A B C B D</td>
<td>0.2</td>
</tr>
<tr>
<td>S3</td>
<td>C A B C</td>
<td>0.2</td>
</tr>
<tr>
<td>S4</td>
<td>A B B C A</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Table II. Position information of items in sequence database SDB

<table>
<thead>
<tr>
<th>Item</th>
<th>Position information</th>
<th>Weight of item</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>(1, 1) (1, 2) (2, 0) (3, 1) (4, 0) (4, 4)</td>
<td>0.467</td>
</tr>
<tr>
<td>B</td>
<td>(1, 3) (2, 1) (2, 3) (3, 2) (4, 1) (4, 2)</td>
<td>0.559</td>
</tr>
<tr>
<td>C</td>
<td>(1, 0) (1, 4) (2, 2) (3, 0) (3, 3) (4, 3)</td>
<td>0.465</td>
</tr>
<tr>
<td>D</td>
<td>(1, 5) (2, 4)</td>
<td>0.259</td>
</tr>
</tbody>
</table>

Figure 1. WCmedTree with 1-frequencies.

In this section, we first discuss the method WCmedMining (Weighted Closed Sequential pattern Mining) which is used to extract features in vulnerability sequences. Then an example is given to demonstrate the execution process of our method.

A. The Description of Algorithm

For the process of mining weighted closed sequential patterns in our method, firstly, the weighted measure factor and the positional information are introduced to calculate each item in vulnerability sequences. After obtaining the frequent sequences of length one (1-frequencies), frequent sequences of length two (2-frequencies) and corresponding position information can be generated simply through matching the position information of 1-frequencies with each other. Then 2-frequencies are implemented to closure-check whether conditions for pruning strategies are not met. Finally, we utilize the above procedure to generate weighted closed sequences, whose lengths are gradually increased.

Algorithm: WCmedMining

Input: \( SDB = \{S_1, S_2, ..., S_n\} \); the vulnerability sequences database \( weight(S_i) \); weight of sequence \( S_i \).
- \( min-sup \): the threshold for the minimal support
- \( WCmedTree \): the null tree for weighted closed sequential tree

Output: the weighted closed sequential patterns

1: scan sequences database \( SDS \) once, calculate the weight of each item \( a_i \) in \( SDS \) according to definition 4;
2: calculate the position information \( IPos \) of each item \( a_i \);
3: obtain frequent sequences of length one (1-frequencies), and insert 1-frequencies into \( WCmedTree \);
4: for (each 1-frequency \( <a_i> \) of 1-frequencies) {
5: match \( IPos \) of \( <a_i> \) with each \( IPos \) of 1-frequencies, and generate 2-frequencies and the corresponding \( IPos \);
6: perform the closure check, and insert 2-frequencies into \( WCmedTree \);
7: }
8: set null sets \( P \) and \( P' \);
9: for (each 2-frequency \( <a_i, a_j> \) of 2-frequencies) {
10: for (each 1-frequency \( <a_i> \) of 2-frequencies \( <a_i, a_j> \)) {
11: if (exist backwards super pattern \( <a_i, a_j> \) exist same positional information)) {
12: set \( sup(<a_i>) = 0 \) in \( WCmedTree \);
13: else
14: add \( <a_i, a_j> \) to set \( P \);
15: }
16: }
17: if \( (P = null) \) {
18: Exit;
19: else {
20: for (each sequence \( S_i \) of \( P \)) {
21: match \( S_i \) with each of the 1-frequencies, and generate sequence set \( P_S \);
22: for (each sequence \( S_j \) of \( P_S \)) {
23: if \( (S_j \) is frequent)
III. The whole set of 2-corresponding weights are generated and shown in Table I.

B. The Example of Algorithm

Fig. 2 shows the algorithm to compute the complete set of weighted closed sequences. In our algorithm, the sequence information is completely reserved by position information. Sequence connection \( IPo_i \otimes IPo_j \) is proposed to match the position information between \( IPo_i \) and \( IPo_j \), i.e., if \( Sid_i = Sid_j \) and \( itemID_i \leq itemID_j \), then the position information of \( IPo_i \otimes IPo_j = (Sid_i, itemID_i) \) and \( IPo_j = (Sid_j, itemID_j) \) denotes as position information of item \( a_i \) and \( a_j \) respectively. For example, position information of item \( A \) is \( (1,1) \), position information of item \( B \) is \( (1,3) \), then position information of sequence \( \{ A, B \} \) is \( IPo S = IPo A \otimes IPo B = (1,3) \). The process of performing the sequence connects in such a way that the position information for sequences is simply calculated. Backwards super pattern and same positional information are utilized to implement the closure check, if two sequences have the same positional information, they also have the same projection database. We adopt the WClosedTree structure to store closed sequences discovered so far, and detection of backwards super pattern technology to accelerate the closure check. Otherwise, the measurement of weight for each closed sequence ensures that feature sequences with comparatively great values are more valuable to future speculation.

### Table III. Position Information of 2-Fresequences

<table>
<thead>
<tr>
<th>2-Fresequence</th>
<th>Position Information</th>
<th>Weight of 2-Fresequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;A, A&gt;)</td>
<td>(1,2) (4,4)</td>
<td>0.934</td>
</tr>
<tr>
<td>(&lt;A, B&gt;)</td>
<td>(1,3) (2,3) (3,2) (4,2)</td>
<td>1.026</td>
</tr>
<tr>
<td>(&lt;A, C&gt;)</td>
<td>(1,4) (3,2) (3,3) (4,3)</td>
<td>0.932</td>
</tr>
<tr>
<td>(&lt;A, D&gt;)</td>
<td>(1,5) (2,4)</td>
<td>0.726</td>
</tr>
<tr>
<td>(&lt;B, B&gt;)</td>
<td>(2,3) (4,2)</td>
<td>1.118</td>
</tr>
<tr>
<td>(&lt;B, C&gt;)</td>
<td>(1,4) (2,2) (3,3) (4,3)</td>
<td>1.024</td>
</tr>
<tr>
<td>(&lt;B, D&gt;)</td>
<td>(1,5) (2,4)</td>
<td>0.818</td>
</tr>
<tr>
<td>(&lt;C, A&gt;)</td>
<td>(1,2) (3,1) (4,4) (5,3)</td>
<td>0.934</td>
</tr>
<tr>
<td>(&lt;C, B&gt;)</td>
<td>(1,3) (2,3) (3,2) (4,2)</td>
<td>1.024</td>
</tr>
<tr>
<td>(&lt;C, C&gt;)</td>
<td>(1,4) (3,3)</td>
<td>0.930</td>
</tr>
<tr>
<td>(&lt;C, D&gt;)</td>
<td>(1,5) (2,4)</td>
<td>0.724</td>
</tr>
</tbody>
</table>

![Figure 3. WClosedTree with 2-fresequences.](image-url)
0.465)\} in \textit{WClosedTree} are updated with set support to 0, which is denoted as the corresponding sequences are not closed, as shown in Fig. 4.

Fig. 5 shows the final \textit{WClosedTree} structure, the complete set of weighted closed sequential patterns is obtained, that is \{\langle A, A \rangle: (2, 0.934); \langle A, B, B \rangle: (2, 1.585); \langle A, B, C \rangle: (4, 1.491); \langle A, B, C, D \rangle: (2, 1.750); \langle C, A \rangle: (3, 0.932); \langle C, A, B, C \rangle: (2, 1.956); \langle C, B \rangle: (3, 1.024)\} through implementing depth first search \textit{WClosedTree} structure and identifying the sequences whose supports are not 0.

\textbf{C. The Process to Extract Feature Sequences in Software Vulnerabilities}

The process of extracting feature sequences in software vulnerabilities is shown in Fig. 6. The function of each module is explained as follows:

\textbf{a. Obtain software vulnerability models}

The existing software vulnerability models are collected and loaded in memory, which sent to next step for further treatment.

\textbf{b. Perform preprocessing}

The irrespective functions and program sentences are filtrated, e.g., the path of database, the list of the calling functions, a variety of string information and so on.

\textbf{c. Extract operation sequences}

After pretreatment processing, the data structures for vulnerability models are established. We utilized the
control flow extraction algorithm to process the above models, and the corresponding control flow graphs are converted. Moreover, we extract function relationships in creating medium file, the call graphs can be constructed for vulnerability models. The operation sequences are obtained by carrying on the control flow graphs and calling graphs depth first path traversal. Otherwise, each operation sequence is denoted formally as sequence of strings, which is given a unique identification Sid in sequences database SDB.

d. Generate feature sequences of software vulnerabilities

We adopt WCSMining algorithm proposed in this paper to mine the closed sequential patterns, and each closed sequence as a feature sequence of software vulnerability. In addition, the frequencies for vulnerability models are diverse, i.e. some of software vulnerabilities appear frequently and some appear rarely, we introduce the weight to meet the needs of decision makers more closely.

e. Incremental update operation sequences of software vulnerabilities

When the new operation sequences of software vulnerabilities generate, the new operation sequences are combined to incremental update sequences database SDB, based on the discussion of existing operation sequences.

IV. EMPIRICAL RESULTS

In this section, we perform a thorough evaluation of WCSMining on various kinds of datasets, compared with two closed sequence mining algorithms CloSpan and BIDE. CloSpan and BIDE were provided as source code. We perform all the experiments on a 2.8GHz processor computer with 2GB memory and running on Windows XP professional. In addition, all the proposed techniques are implemented in Java.

A. Datasets

Synthetic datasets, the datasets were produced from the well-known IBM synthetic dataset generator (http://www.almaden.ibm.com/cs/quest), which has been used in the evaluation of almost all sequence mining algorithms in data mining literature. The generator simulates a set of user product buying sequences. A user sequence is a list of transactions in the purchase order. The parameters and their value ranges are shown in Table IV. For example, D100C10T5N20 means there are 100K sequences, each sequence has about 10 transactions, each transaction has about five items and the dataset has 20K distinct items.

Gazelle is a sparse dataset, but it contains some very long frequent closed sequences with low support threshold. This dataset was originally provided by Blue Martini Company. It contains totally 29369 customers’ Web click-stream data. For each customer there is a corresponding series of page views, and we treat each page view as an event. This dataset contains 29369 sequences (i.e., customers), 87546 events (i.e., page views), and 1423 distinct items (i.e., web pages). The average length and the maximal length of this dataset is 3 and 651, respectively.

In order to test all algorithms, two datasets are employed. One is the Synthetic datasets; the other is the Gazelle dataset. The sequences in Synthetic datasets are generated by the input parameters, and Gazelle dataset is randomly selected. When testing WCSMining, we use D10C10T2.5N10, D10C15T2.5N10, D10C5-20T2.5N10 and D10C6T1-10N10 as the Synthetic databases.

B. Comparison of WCSMining with other Algorithms

Fig. 7 shows the running time of WCSMining, BIDE and CloSpan when min-sup is varied from 0.02% to 0.1% on D10C10T2.5N10.

On average, closed sequence mining algorithms are faster than algorithms which mine the complete set of frequent sequences from scratch. In comparison with the closed sequence mining algorithms BIDE and CloSpan, WCSMining is about 4 or more times faster than others. When min-sup is low, the gap between WCSMining and other algorithms is much more obvious. For instance, with min-sup = 0.02%, WCSMining completes in 9.27s, while BIDE and CloSpan completes in 44.29s and 102.55s, respectively. It is because at the extremely low support, there are too many non-closed patterns generated; WCSMining can successfully prune the non-closed sequences. Another main reason is that WCSMining starts its performance from calculating position information, using the nice properties of WClosedTree to extend the nodes and to update the support states of only a few nodes in the WClosedTree.
To test whether the algorithms perform unstably in different runs, we implemented the three algorithms 15 times, and the average running time is illustrated in Fig. 8. It can be shown that the running time of the three algorithms displays little variations in different runs.

The running time of all the algorithms on dataset D10C15T2.5N10 is given in Fig. 9. Fig.9 displays the same trend as Fig.7. The only difference between dataset D10C15T2.5N10 from D10C10T2.5N10 is the average number of transactions per customer. For this reason, the running time of each algorithm in Fig.9 is slightly higher than the running time of each algorithm in Fig.7.

Fig. 10 and Fig. 11 depict the comparison results among CloSpan, BIDE and WCSMining for Synthetic datasets D10C5-20T2.5N10 (the number of transactions per customer is increased from 5 to 20) and D10C6T1-10N10 (the number of items per transaction is increased from 1 to 10), where min-sup is fixed at 0.05%. When the number of transactions per customer or the number of items per transaction is increased, the average length of sequences is also increased. Thus, the datasets become much denser and there are more closed sequential patterns. This results in the significant increase of the running time of all the three algorithms. We can see from the figures that WCSMining still outperforms BIDE and CloSpan by a large margin.

In Fig. 12, the running time of the three algorithms on the real dataset Gazelle is illustrated. The support threshold is varying from 0.04% to 0.02%. The average length of this dataset is shorter than that of Synthetic datasets, so it is sparser than Synthetic datasets. From Fig.12 we can see that when the support is greater than 0.025%, WCSMining show better performance than BIDE and CloSpan, especially when we continue to lower the support threshold, WCSMining algorithms will outperform a lot the two algorithms. On this sparse dataset, an explosive number of non-closed sequences are generated, BIDE and CloSpan cannot prune the redundancy sequences in a relatively short time.

Figure 8. Mean running time on D10C10T2.5N10.

Figure 9. Running time on D10C15T2.5N10.

Figure 10. Varying the number of transactions per customer.

Figure 11. Varying the number of items per transaction.

Figure 12. Running time on Gazelle.
Overall, the WCSMining algorithm runs faster than BIDE and CloSpan algorithms in five different types of experimental datasets, and there are different levels of improving performance. In other words, the overall performance of WCSMining algorithm is significantly reduces the time consumption.

C. Representing Feature Extraction in Software Vulnerabilities

A feature extraction system has been developed based on the above mentioned methodology. Java is the main program development environment. The existing software vulnerability models have been used as the underlying modeling kernel. The filtrate task is considered indispensable as the pretreatment processing to be carried out. Operation sequences are obtained from the data structure for vulnerability models. Alongside the complete closed sequential patterns as feature sequences generated, WCSMining also offers the weight of each feature sequence for displaying weightiness. In a word, the method of mining closed sequential patterns is introduced to extract feature sequences in software vulnerabilities is unprecedented and effective.

V. CONCLUSION

The rapid development of Internet and the extensive applications of software in many key domains bring a new challenge in software security, so feature extraction of software vulnerability has aroused a deep concern increasingly. In this paper, closed sequences mining technology is introduced to extract feature of software vulnerability. Aiming at the problem of varying frequencies for software vulnerabilities, we present an effective algorithm WCSMining for mining weighted closed sequences as feature sequences. WCSMining utilizes position data to reserve sequence information in order to reduce the search space and degraded the time complexity. Moreover, WClosedTree is defined to keep the closed sequential patterns and other auxiliary information, and two main pruning strategies are developed to remove frequent but non-closed sequential patterns in WClosedTree. In addition, WCSMining accelerates the process of extracting features; this method overcomes the disadvantages of including a great deal of repetition works, and improves the efficiency of the whole extractive process.

ACKNOWLEDGMENT

The authors thank Jiadong Ren, who is a PhD candidate supervisor of Yanshan University for his careful review and detailed guidance. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES


Quanhui Wu She is a doctor candidate at state key lab of software development environment, school of computer science and engineering, Beijing University of Aeronautics and Astronautics. She was born in 1984. Her main research interests include data mining and software engineering.
**Shilong Ma** He is a doctor supervisor at school of computer science and engineering, Beijing University of Aeronautics and Astronautics. He was born in 1953. His main research interests include calculating model on the network environment, dynamic statistic behavior of logic and computation, calculating model of massive data process, grid computing technology and application.

**Hao Wang** He is a master, who now works at NARI group corporation, State Grid Electric Power Research Institute. He was born in 1978. His main research interests include information security and software engineering.