

Received March 14, 2020, accepted April 21, 2020, date of publication April 29, 2020, date of current version May 15, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2991328

A Constant Time Complexity Spam Detection Algorithm for Boosting Throughput on Rule-Based Filtering Systems

TIAN XIA 

Computer and Information Engineering Department, Shanghai Polytechnic University, Shanghai 201209, China

e-mail: xiatian@sspu.edu.cn

This work was supported in part by the Key Disciplines of Computer Science and Technology of Shanghai Polytechnic University under Grant No. xxkzd1604.


ABSTRACT Along with the barbarous growth of spams, anti-spam technologies including rule-based approaches and machine-learning thrive rapidly as well. In antispam industry, the rule-based systems (RBS) becomes the most prominent methods for fighting spam due to its capability to enrich and update rules remotely. However, the antispam filtering throughput is always a great challenge of RBS. Especially, the explosively spreading of obfuscated words leads to frequent rule update and extensive rule vocabulary expansion. These incremental obfuscated words make the filtering speed slow down and the throughput decrease. This paper addresses the challenging throughput issue and proposes a constant time complexity rule-based spam detection algorithm. The algorithm has a constant processing speed, which is independent of rule and its vocabulary size. A new special data structure, namely, Hash Forest, and a rule encoding method are developed to make constant time complexity possible. Instead of traversing each spam term in rules, the proposed algorithm manages to detect spam terms by checking a very small portion of all terms. The experiment results show effectiveness of proposed algorithm.

INDEX TERMS Constant time complexity, hash forest, rule-based filtering, spam detection, throughput.

I. INTRODUCTION

The widespread use of Internet had grown explosively since the first establishment of Internet in 1969. Internet had connected each individual together via computers and mobile devices. Along with it, the scale of data is overwhelmingly increased as well [1], especially after the wide use of social networks, personal communication tools, emails and short messages (SMS). This easy-communication circumstance also encouraged the numerous emerge of spams. Such kind of activities turned into one of the most profitable businesses for spammers and criminals.

Spams first spread explosively but mainly in emails in the first decade of 21st century, indicated by the statistic results provided in [2], [3]. Spam e-mails grew exponentially from 8% in 2001 up to 90% during 2009 [2]. Fortunately, the trend turned downward to around 8.2% of junk e-mail volumes worldwide because of the withdraw of rogue Internet Service Providers such as 3FN, Bredolab, Rustock, and Grum.

The associate editor coordinating the review of this manuscript and approving it for publication was Choon Ki Ahn .

Then, spams spread widely into the fast growing SMS service, because it has reached more than 6 billion users globally with approximately 9.5 trillion SMS sent globally in the year 2009. At first, spam SMS generally less pervasively than email spam [4]. Then, mobile spam has steadily increased from 2008 to 2012 and recently account for half of all North America mobile phone traffic in 2019 [5]. The increased use of SMS service has sustained great profits close to 117.2 billion dollars in 2017 [6]. The great interests have attracted malicious spammers to spread unsolicited, commercial, bulk electronic message [7]. Such SMS may sometimes convey undemand adverts, viruses, malware or other annoying contents targeted at consumers, businesses or government organizations. Recently, security has become main threatening concern because spam SMS often attract users to reveal critical personal information by promising free gifts, cheap product offers, credit cards or debt relief services.

Anti-spam technologies have been developed much along with the growth of spams. Compared with machine learning approaches boosting in antispam research, the rule-based systems (RBS) have been much predominant in the filtering

industry since it integrates various classification methods and are able to update filters online remotely. Recent years, motivated by the benefits provided by SaaS (Software as a Service), RBS are running on cyber security companies, such as Symantec Cloud, McAfee Cloud Security, or Kasper-sky Hosted Security [8]. The most popular one is SpamAssassin [9].

However, such cloud-based spam-filtering services concentrate filtering tasks and demand much high throughput capabilities. In addition, the variety of spam always challenges the limited RBS filtering throughput. The overwhelmingly emerged obfuscated words, as shown in TABLE 1, are the most difficult and challenging issue. They are possible to get through RBS spam filters as they always contain unusual symbols or blanks. However, they are still able to be associate to spam words in mind.

TABLE 1. Obfuscated words in spam SMS messages.

Spam Terms	Obfuscated words
Cash	cAsh, c@sh, c@sh,cAsh, c@sh
Debt	De bt, d@bt, dëbt
Important, Credit Card	I M POR TAN T, C re it Ca rd
Discount, Pay by Loan	Di'scountq, Pa'y by loankh

Obfuscated words directly lead to rules spam term vocabulary explosion and rule quantity increasing extraordinarily. Consequently, the computation time for detecting and filtering grows dramatically, making the running RBS slow down significantly and even impractical.

To overcome the drawback, some researchers studied scheduling strategies [10], [11], some researchers provided simulation tools of filters' throughput for research [12] and some companies relied on upgrading hardware equipment.

However, an inconstant time complexity algorithm stretches the computation time longer while rules quantity and its spam term vocabulary increase. If the time complexity of filtering algorithms of RBS can reduce to constant, the throughput issue can be solved since the expansion of rule and its vocabulary size will not slow down filtering speed ever.

RBS rules include a Boolean expression and a rank. The Boolean expression is a combination of spam terms and logical operators. An RBS is capable to detect spam SMS rapidly based on the Boolean expressions and return spam SMS ID and matching rule ID pairs. In our research, several creative methods manage to complete the filtering process within an extremely short and constant time. We first studied on the rule representation in computer memory. The rule data are organized in a speed-optimized structure, namely, Hash Forest, which is capable to avoid accessing each spam term while detecting spams. Then, we studied on the method to calculate Boolean operators automatically through the spam detection process. Currently, we support the use of && (*AND operator*), || (*OR operator*) and () (*bracket operator*) in rule

Boolean expressions. Other operators will be supported in our future research.

This research was motivated by the cooperation with a communication service company. This company provides SMS service in Shanghai and serval provinces near around. Their SMS includes verification codes SMS, e-commercial SMS, express delivery SMS, government SMS, promote sale SMS of malls or grocery stores, etc. The company sends more than 80 million business short messages per day on average and about 150 million per day on peak. The project was carried out to increase filtering speed to meet its overwhelming SMS sending throughput requirement. This project successfully addressed the throughput issue and decreased the time complexity of the spam detection algorithm to constant $O(1)$. About 150 million SMS can be filtered in less than one hour. Currently, the company is running 110k rules, including 10k black rules and 100k white rules. With the rise of rule number, the algorithm maintains its constant spam detection speed.

The main contributions are following.

1) Propose a new rule data structure, namely, Hash Forest. It rearranges spam terms into search routes branches. Hash Forest helps the algorithm detect spam term by checking a very small portion of them.

2) Propose an encoding method for rule Boolean expressions. The encoding method helps the filter calculate operators in expressions automatically. This process has constant time complexity.

3) Develop a spam detection algorithm that only processes symbols (i.e. letters in English) instead of whole spam terms. Since symbols in languages are limited, the time complexity is greatly reduced.

4) Support sequential matching. A rule contains a logical expression. The logical brackets in an expression can be detected one by one from beginning to the end with constant time complexity.

The rest of the paper is organized as follows. Section II discusses the related work. Section III presents the proposed Constant Time Complexity Spam Detection algorithm. Section IV outlines the experiment's results to show the constant computational complexity. The conclusion is drawn and future work is discussed in Section V.

II. RELATED WORK

Along with the barbarous growth of spams, antispam technologies also thrive prosperously. The antispam technologies include content filtering systems and pattern detecting systems. Both have advantages and limitations.

A. THE CONTENT FILTERING TECHNOLOGIES

The content filtering systems utilize statistical machine learning approaches. Many models have been applied to obtain better spam detection results, such as Support Vector Machine (SVM) [13], Bayesian methods [14], Decision Tree [15], etc.

SVM creates a multiclass, SVM-based classifier from a set of binary SVM classifier. SVM becomes popular because it is robust for many circumstances. SVM trains a decision

equation from a high-dimensional feature space, which leads to high accuracy. In addition, it has been developed into several types, including one-against-rest SVM (OAR-SVM), one-against-one SVM (OAO-SVM), directed acyclic graph SVM (DAG-SVM), adaptive directed acyclic graph SVM (ADAG-SVM), and error-correcting output code SVM (ECOC-SVM) [16].

Bayesian methods, such as Naive Bayes, are regarded as the effective and important machine learning algorithms in information retrieval. Teraguchi *et al.* [17] proposed a Bayesian algorithm to defeat spammers. To enhance its accuracy, Bayesian methods are often hybrid with other algorithms. Ebadati and Ahmadzadeh [18] proposed a GA-Naive Bayes for spam email detection with GA algorithm for feature section. Arifin *et al.* [19] focused on spam detection for SMS by Naive Bayes Classifier and FP-Growth since FP-Growth is utilized for mining frequent patterns.

In addition, other machine learning methods, such as boosting trees [20], Neural Networks [21], are also applied in spam detection and outperform Naive Bayes and Decision trees.

Machine learning approaches usually require a beginning training for spam filter and training again if rules are updated. Then, the filter may be required to restart to load the updated models. Therefore, although many progresses have been made in research, their deployments in antispam industry are not often reported.

B. RULE-BASED SYSTEMS

Rule-based systems, one of the most prominent methods for fighting spam in industry, combine Pattern Detection technologies, such as operation research, graph theory, data analysis, clustering. It has been deployed and spread in the antispam industry because they are capable to update rules online remotely.

Anti-spam RBS filter spam based on a set of pares of rules and its scores. Whenever a rule is matched through its logical test, along with Boolean TRUE returns, its score is accumulated into a global counter. A spam is determined by whether the global counter comes up to a preconfigured threshold.

1) THE SpamAssassin FRAMEWORK

SpamAssassin [9], a successful forerunner of typical RBS, was developed from two rudimentary rule engines filter.plx [22] and Spamometer [23]. In a long time, SpamAssassin play a vital role [24] and has been adopted by antispam industry companies (such as Symantec or McAfee) [10].

SpamAssassin is integrated with the most popular Mail Transfer Agent (MTA) packages (e.g. Postfix, Exim or QMail). It is designed to listen to TCP port 783. All received packages are analyzed and several features are extracted as inputs of rules filter. Then, spam emails are determined by the accumulation of a global counter. SpamAssassin supports Naive Bayes classifier, Sender Policy Framework (SPF) verifier as build-in modules and third party filter as plugins.

However, SpamAssassin was reported to have a low filtering throughput [25]. Therefore, a new-generation RBS middleware has been developed from SpamAssassin, such as Wirebrush4SPAM, to relieve its heavy throughput pressure.

2) THROUGHPUT IMPROVEMENTS

Throughput capability is always challenging essential features of RBS. Fortunately, achievements have been made to cope with performance issues [11].

The smart filter evaluation (SFE), introduced in Wirebrush4SPAM [25], is able to terminate a rule execution at a proper point based on evaluation. The termination is aimed to save computational recourses.

Learning After Report (LAR) [25] in Wirebrush4SPAM generates a new thread to process auto-learning tasks which is inherited from SpamAssassin.

Identification of Bayes Useless Information (IBUI) in [26] refines Naive Bayes databases by removing unhelpful tokens if it has an over-threshold Inverse Document Frequency (IDF) value.

Per Rule Parallelization (PRP) [25] in Wirebrush4SPAM upgrades parallel computing of SpamAssassin. It has a concurrent rule execution scheme capable to take advantage of parallelizing computation regardless of the number of pending classifications.

Sufficient Condition Rules First (SCRF) [9] in a plugin of SpamAssassin is able to terminate filter execution as soon as enough matching rules are found.

The improvements discussed above mainly focus on simplifying the filtering stages, omitting unnecessary process and parallel computing. However, despite of the progress made by these technologies, they did not report to decrease the time complexity of filtering algorithms to an acceptable level. Therefore, along with the explosion of term vocabulary in rules, the filter speed will still keep slowing down.

This context provides a solution for throughput issue in RBS by downgrading the computational complexity to constant. That is, the speed of filtering algorithm is irrelevant to rule size or rule term vocabulary.

III. THE CONSTANT TIME COMPLEXITY SPAM DETECTION ALGORITHM

A. PROBLEM FORMULATION

The method presented in this paper is able to scan candidate SMS to detect their all matching rules and return their rule IDs. The filtering speed is independent of the increasing rule amounts and rule term vocabulary.

Typical Boolean expressions of an English rule, a Chinese rule and a mixed-language rule are shown in TABLE 2.

Apparently, rules always contain symbols including letters, Chinese characters, numbers and some special symbols. Therefore, a typical Boolean expression can be represented as a unified formula shown in (1):

$$\begin{aligned} & (T_{11}||T_{12}||\cdots||T_{1n}) \&\& \\ & (T_{21}||T_{22}||\cdots||T_{2n}) \&\& \\ & \dots \\ & (T_{m1}||T_{m2}||\cdots||T_{mn}) \end{aligned} \quad (1)$$

TABLE 2. Typical Boolean expression of rules.

<i>English Rule</i>
(Corp. Company Co. Discount IMPORTANT Debt)&& (£ \$ € ¥ € 円 %)&& (Credit Loan Check Ca\$h)&& (Contact MP Reply)
<i>Chinese Rule</i>
(圈地 小区 区域 车载)&& (基站 短信 设备)&& (发送 群发 免费 费用)&& (13 14 15 18)
<i>Mixed-language Rule</i>
(营 春晚 歌手 星光)&& (笔记本 奖金 元)&& (WWW www COM com 码)

where T_{ij} represents a certain spam term in the Boolean expression.

For a single term T_{ij} , let $S_1^{ij}, S_2^{ij}, S_3^{ij}, \dots, S_k^{ij}, \dots, S_K^{ij}$ represent the symbols in T_{ij} . Here, symbols are letters in alphabet language, like English, or hieroglyphic characters in pictographic language. In this paper, we say symbols instead of letters because there are different symbols in different languages. Please note that this approach can also apply to other language, regardless alphabet language, like English, or pictograph language, like Chinese, even multiple language mixture.

Filter speed is the main challenge of RBS throughput. The paper focuses this issue and reduces the SMS filtering time to constant. For a given set of SMS, the overall speed of the proposed algorithm is only related to spam SMS amount and remains stable while rule amount and its vocabulary are expanding.

B. THE HASH FOREST

The Hash Forest, a special data structure of all spam terms, forms the foundation of the constant time complexity of term detection in the proposed spam detection algorithm.

1) PROCESSING SYMBOLS INSTEAD OF WORDS

There are numerous words in each language whereas symbols are limited. Processing a relatively small set of symbols make it possible to deploy $O(1)$ search algorithms, like Hash search, on symbols searching.

Apparently, if traversing each word to detect term, as shown in FIGURE 1, time complexity is uncontrollable.

However, the detecting direction can be changed to vertical if symbols are processed. The detection process is to find a path getting through each symbols of a term. By this means, it only checks a small portion of all terms. An example of searching for term $C@sh$ is shown in FIGURE 2.

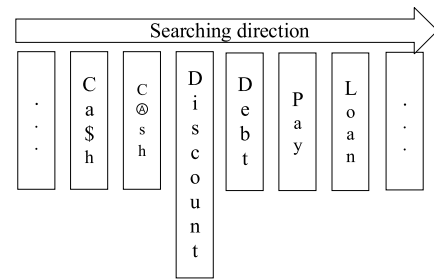


FIGURE 1. Term by term searching direction.

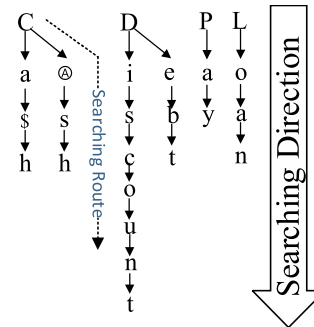


FIGURE 2. Searching route of the term $C@sh$.

Apparently, other terms, such as Debt, Discount, Pay Loan, are left untouched.

Since word length is limited, the searching time for term detection is limited below a constant.

Therefore, to achieve constant time complexity, Boolean rule expressions are first represented in a unified formula and their terms are represented symbol by symbol. Symbols from all languages come from a limited set. Processing symbols instead of terms takes the first step towards the goal: constant time complexity. English language has 52 symbols, including all letters and their capitals. Also, taking special symbols in spam terms in consideration, approximately 200 or more symbols will be added to the set. Therefore, symbol size of spam terms is expected to be limited under 300 in English. Even for Chinese, 3000 frequently used Chinese characters can cover 99% Chinese documents and 1000 can cover 90%. Therefore, in Chinese, not more than 2000 characters are expected in terms of Boolean expressions.

C. THE DATA STRUCTURE OF THE HASH FOREST

The Hash Forest is a data structure to represent all terms in all Boolean expressions.

As shown in FIGURE 2, many spam terms have the first letter in common. This happens more likely in English than in Chinese. In the Hash Forest, these common symbols are merged together to form the root of each tree in the forest.

In addition, the second successive symbol of terms may also be the same. They are merged to form the branch nodes. A root node and a branch node represent a bunch of terms with the first and second symbols in common.

Moreover, in turn, the rest common symbols follow the same process to merge together to form branch tree nodes

in next levels. When no more common symbols are found, the trees continue growing according to symbols of each term until reaching the last symbol which is the leaf node.

For example, as shown in FIGURE 3, suppose there are only 3 terms in rules. They are *Cash*, *C@sh*, *Ca\$h*. They all begin with the letter *C*. So, they will form a single tree with root node *C*. Then, *Cash* and *Ca\$h* have the second letter *a* in common. Therefore, the both terms will continue to merge together with a branch node *a*. The rest uncommon letters link one after another until the last one.

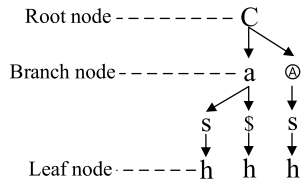


FIGURE 3. A 3 terms in a single-tree hash forest example.

More generally, for instance, suppose there are two terms T_{ij} and T_{pq} . They contain a few common symbols from the beginning. let $S_1^{ij}, S_2^{ij}, S_3^{ij}, \dots, S_k^{ij}$ represents the symbols in T_{ij} and $S_1^{pq}, S_2^{pq}, S_3^{pq}, \dots, S_r^{pq}$ represents the symbols in T_{pq} . Also, S_1^{ij}, S_1^{pq} and S_2^{ij}, S_2^{pq} are the same respectively. They form the tree as shown in FIGURE 4.

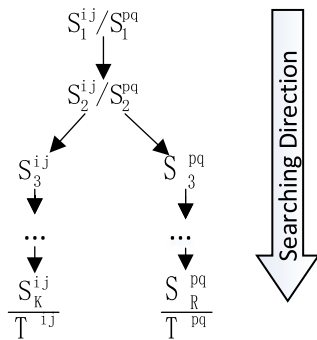


FIGURE 4. A two terms in a single-tree hash forest example.

All terms in rule Boolean expressions are divided symbol by symbol and same divided symbols are merged. Normally, given enough terms, their symbols should be able to form a multiple-tree, namely, Hash Forest. Each different first symbol is the root of each tree. Suppose a black solid circle in FIGURE 5 represents a symbol. The common symbols are merged together. Such example of a multiple-tree Hash Forest is shown in FIGURE 5.

1) SPAM TERM DETECTION VIA THE HASH FOREST

The Hash Forest is designed to speed up the existence detection of spam terms in SMS. It rearranges all spam terms together and builds searching routes for each term. Such searching process avoids checking the majority of terms in each rule. The detection algorithm has $O(1)$ time complexity.

To detect symbols in spam terms, a Hash search runs on each level of the Hash Forest. If the Hash search finds the matching symbol, another Hash search is performed on the

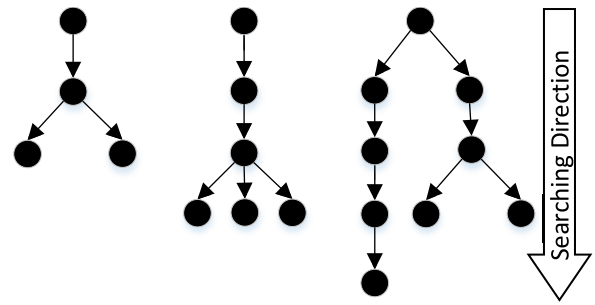


FIGURE 5. A multiple-tree hash forest examples.

next level of the matching node. And so on. When a level only has one node, Hash search is substituted by a matching check. If all Hash searches and matching checks return a matching node until leaf node, a detected term is reported. By this way, Hash searches are able to filter out all other un-matching terms by checking a few symbols of the matching term. Such process only checks a very small portion of term vocabulary like route finding in the Hash Forest.

For instance, as illustrated in Figure 6, if trying to detect *ca\$h* in a term set $\{ca$h, cost, loan, debt, credit, \dots\}$, the first Hash search will locate the tree of symbol *c* as the beginning of this searching route. Then, the second Hash Search is performed on the nodes of level 2, tree *c*. It locates the branch of symbol *a*. Then, the searching route continues by directly getting through the remaining symbols by a few matching checks since no branches exist in deeper levels. If the searching route reaches the leaf node *h*, it indicates that spam term *cash* is detected. Apparently, it is unnecessary to traverse into other trees of other symbols, such as *b* for *debt*, *l* for *loan*, etc. The dash line in FIGURE 6 is the searching route.

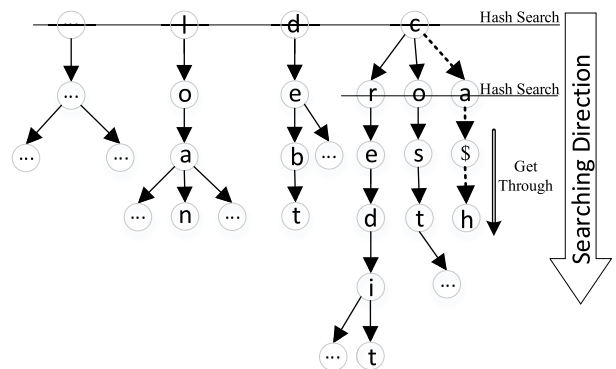


FIGURE 6. An example of spam term detection and its searching route in the hash forest.

2) TIME COMPLEXITY ANALYSIS OF TERM DETECTION

As mentioned above, existence detection of a term is achieved by a few Hash searches in the Hash Forest.

A Hash table (Hash map) is a data structure that implements an associative array abstract data type, a structure that can map keys to values. A hash table uses a hash function to compute a key, also called index or a hash code, and map with an array of buckets or slots, from which the desired value

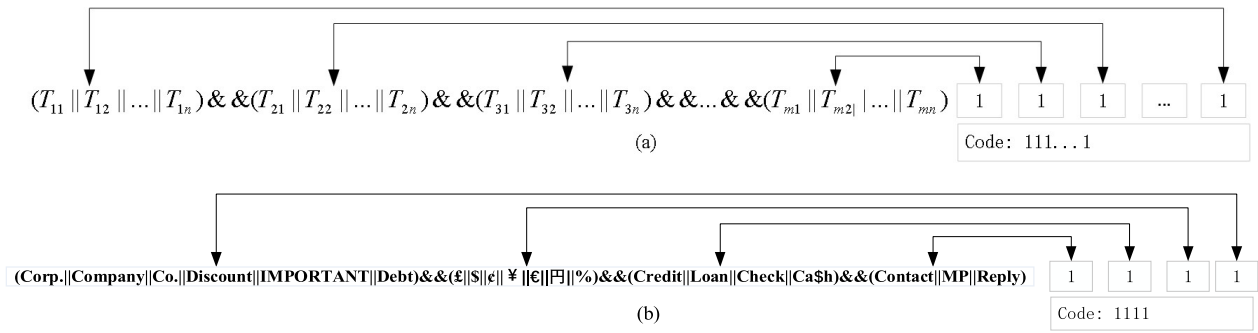


FIGURE 7. A general format (a) and an Example (b) of Boolean expression encoding.

can be found. Hash search gets $O(1)$ search time on average and $O(n)$ in worst case [27]. The worst case happens when all values are mapped to the same bucket. Such scenario is unlikely to happen because the nodes amounts of each level in the Hash Forest is quite limited. Therefore, the Hash search time on each level has $O(1)$ time complexity.

In addition, obviously, the times of Hash searches needed when searching through the Hash Forest depend on the average depth of the Hash Forest. The average depth, then, is determined by the average length of terms. Actually, for English, the average word length is 4.7 characters [28]. This means the average depth of the Hash Forest is 4.7 in English and the average times of Hash searches are 4.7. Furthermore, since the algorithm also accelerate the search speed by getting through the remaining symbols together when there is only one node on these levels, the actual average times of Hash searches are lower than 4.7.

On summary, a searching through the Hash Forest only performs a few Hash searches which time complexity is $O(1)$ on average. Therefore, the overall time complexity for detecting terms should be $O(1)$.

D. RULE IDENTIFICATION

The Rule identification algorithm is designed to calculate the logical operators rapidly and return ID of all matching Boolean expressions of rules.

1) THE ENCODING OF THE BOOLEAN EXPRESSIONS

Take a look at the typical logical rule expression in (1) again.

$$\begin{aligned} &(T_{11} || T_{12} || \dots || T_{1n}) \&\& \\ &(T_{21} || T_{22} || \dots || T_{2n}) \&\& \\ &\dots \\ &(T_{m1} || T_{m2} || \dots || T_{mn}) \end{aligned}$$

where T_{ij} represents a spam term in logical rule expression.

Apparently, expression always contains a few brackets. Also, an identification of any term in a bracket indicates Boolean true of the bracket. The rule is identified matching only when all pair of brackets in a rule Boolean expression equal TRUE. The aim of expression encoding is to make the code equals 0 while all brackets equal true and the rule is matching.

Therefore, the Boolean expressions are encoded at bracket level as shown in FIGURE 7(a). Each bracket is encoded as one Boolean value. To do so, a code of any expression is first initialized as a “1...1” string. Each character “1” represents a bracket in an expression. Once a term in a bracket is detected, the bracket is TRUE and the character “1” responding to the bracket is set to “0” at once. Also, if all brackets are true, the code is “0...0” and its numerical value equals 0. At this time, a rule matching is reported. Such codes can represent and calculate $\&\&$ (AND operator), $||$ (OR operator) and $()$ (bracket operator). Furthermore, this process is $O(1)$ time complexity.

FIGURE 7(b) shows a specific example of Boolean expression encoding. In the example, the rule expression has 4 brackets. It is encoded as a four-character string “1111”. Each character “1” represents a bracket from left to right respectively. Also, a detection of any term in each bracket indicates the logical Boolean TRUE of the bracket. For instance, if *Company* is detected, the bracket $(Corp.||Company||Co.||Discount||IMPORTANT||Debt)$ is TRUE and its corresponding character, or called bit, is set to “0”. At this time, the string is “0111”. With more terms are detected, more brackets are TRUE and more characters “1” are set to “0”. When the string is “0000”, its numerical value equals 0. The matching rule is reported for logging.

2) HASH FOREST WITH EXPRESSION CODE EXTENSIONS LIST

Expression code extensions lists are the lists of rule expression ID and term position pairs. The term position value logs that bracket the term exists in the rule.

For example, as shown in FIGURE 7(b), the term *Contact* exist in the 4th bracket of the rule expression. The expression code expressions should be $[<rule\ expression\ ID>, 4]$. The filter will use the position 4 to set the 4th bit of the code to 0 when detecting term *Contact*.

Usually, a term may exist in more than one rules. Therefore, a list of rule expression ID and term position pairs links to the leaf node of the term indicating all rules in the list contains the term.

While initializing the filter, all rule Boolean expressions are traversed once to form the Hash Forest.

Also, the initializing preprocess simultaneously collects term position data, generates the lists of expression ID and term position ID pairs and links them to the Hash Forest.

For example, suppose a rule with ID 0001 is $(ca\$h|credit)\&\&(cost)\&\&(loan|debt)$. The position number of each term depends on the number of bracket where it exists. Their code extensions link to the Hash Forest, as shown in FIGURE 8. The position of term $ca\$h$ and $credit$ is 1 as they exist in the first bracket. The term $cost$ is in position 2 because it is in the second bracket. Thus, $loan$ and $debt$ has the position 3.

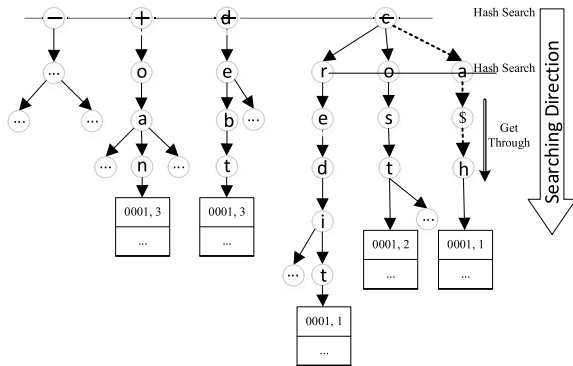


FIGURE 8. An example rule in hash forest with code extensions list.

A more general example of Hash Forest with Code Extensions Lists is shown in FIGURE 9. Note the black solid circles represent symbols of terms.

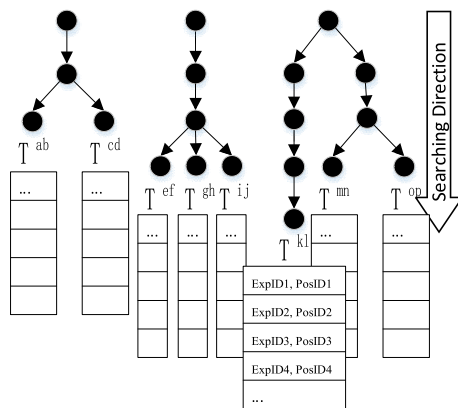


FIGURE 9. Hash forest with code extensions list.

3) TIME COMPLEXITY ANALYSIS OF RULE IDENTIFICATION

Once a term is detected, all rules containing the term are matching candidates. Their codes are set by bit based on the code extensions list of the term. The corresponding bit of code is set to 0. When the algorithm ends, the only remaining work is to find all matching expression by checking straight-zero codes.

Furthermore, the codes are stored as a String. Each bit of codes can be accessed by a char array, such as `String[]`. So, the time complexity of setting codes by bit is $O(1)$. Once a term is detected, the codes of corresponding rules are set. If each

bit of a code is set to 0, it indicates that all brackets in the rule is TRUE and the rule is matched. At this time, the numerical value of the String equals 0, a matching event is reported and the rule ID is logged in the matching rules list.

The time complexity of these steps is all irrelevant to either rule size or rule term vocabulary. The overall time complexity of rule identification is $O(1)$.

E. THE SPAM SMS DETECTION ALGORITHM

This part will take SMS spam filtering as an example to illustrate how a SMS message is traversed and how the logical rule expressions are identified during the SMS traversing process.

A SMS message is traversed only once. Since SMS message always has a limit length, in this respect, the time complexity of processing a single SMS message is also constant.

FIGURE 10 shows the traversing process of a SMS message.

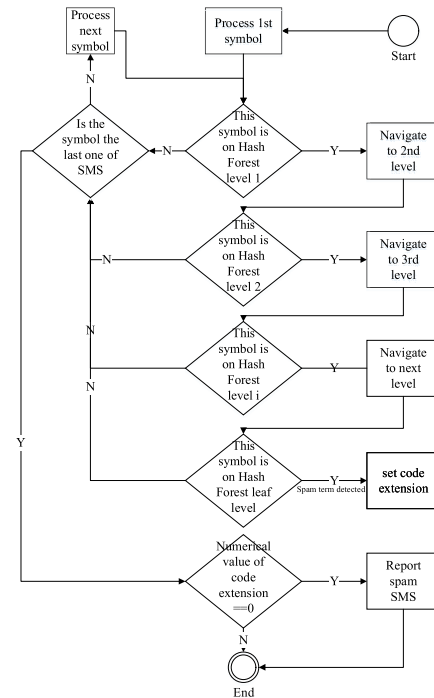


FIGURE 10. The SMS traversing process.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

A. EXPERIMENTAL RESULTS

The experiment is based on production environmental data of the SMS service company cooperated with us mentioned in Introduction.

The experiment is designed to validate the constant time complexity of the proposed spam detection algorithm. The filter processes a given set of SMS and filters certain spam SMS in it. Then, we will check if the processing time remains stable while adding more rules with term vocabulary expanding.

Algorithm 1 Single SMS Filtering

- Step 1.** Start processing from the 1st symbol of the SMS message.
- Step 2.** Perform a Hash search on 1st level of the Hash Forest. If not found, go to Step 7.
- Step 3.** Perform a Hash search on 2nd level of the Hash Forest. If not found, go to Step 7.
- Step 4.** Perform a Hash search on i^{th} level of the Hash Forest. If not found, go to Step 7.
- Step 5.** Perform a Hash search on the leaf level of the Hash Forest. If not found, go to Step 7.
- Step 6.** Term detected and set code extensions by bit. If the numerical value of the code is 0, a matching rule is reported.
- Step 7.** If the current symbol is not the last one, process the next symbol and go to step 2.
- Step 8.** Check if any matching rules are reported. If so, report spam SMS with the rules' IDs.

The filter speed is too fast to notice the time difference for a small size of SMS set. To make the processing time visible, the given set of SMS has a size of 500,000.

To verify the overall spam detection time, the experiment is designed to perform on 9 different sizes of rules. The first batch has 120 rules and they filter out 28533 spam SMS. Then, 10 rules are added each time. The 2nd one contains 130 in total. And so on, the 9th group contains 200 rules. The term vocabulary expands while adding more rules.

The experiment is run on a MacBook Pro 2018 and the results are shown in TABLE 3.

TABLE 3. Experimental results.

Size of Rules	1st Time(s)	2nd Time(s)	Term Vocabulary
120	25.812	25.467	1437
130	25.399	25.230	1512
140	25.581	25.251	1615
150	25.623	25.051	1736
160	25.297	25.482	1864
170	26.036	25.890	1989
180	25.403	25.254	2109
190	25.666	25.045	2225
200	25.977	25.162	2346

Please note that 500,000 SMS are filtered below 26 seconds. The filtering speed of the algorithm running on a laptop is 70,000,000 SMS per hour. Actually, our cooperated SMS service company reports a nearly 10 times faster speed on their server.

As shown in FIGURE 11, the time consumed to filter the 50,000 spam remains stable while rules consistently increase. In this procedure, term vocabulary expands much larger than its original. Therefore, the experiment results validate the constant time complexity of the spam detection algorithm proposed in this paper.

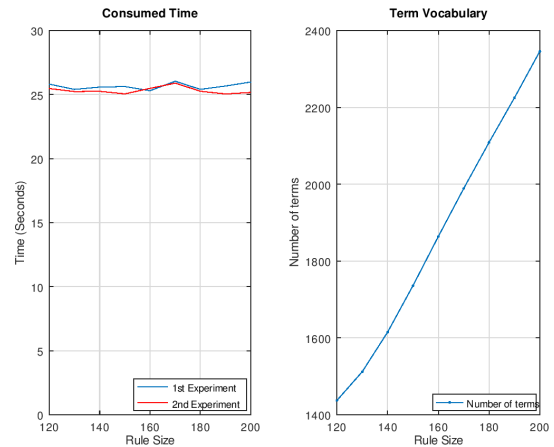


FIGURE 11. The experimental results: consumed time (seconds) and expanding term vocabulary.

In addition, the methods and algorithms in this paper have also been implemented in a SMS service company in anti-spam industry. 80 million SMS are sent per day on average and 150 million are sent on peak days. The rules keep raising. According to our latest contact, the company is currently holding 110k rules, including 10k black rules and 100k white rules. The processing speed remains stable regardless the increasing size of rules and its vocabulary. Before implementing the proposed algorithm, the company had 6 powerful servers to parallel compute their old filtering program. Now, only one virtual server is dedicated to filter spam SMS and its CPU occupation is low.

B. AN ADDITIONAL FEATURE: SEQUENTIAL MATCHING

Sequential Matching is an additional feature of the method presented in this paper. In addition to Boolean expression matching, the SMS service company also demands each pair of brackets of an expression must be satisfied one after another sequentially.

Take the rule $(ca\$h||credit)\&\&(cost)\&\&(loan||debt)$ for instance, the second bracket is TRUE not only when term *cost* is detected but also when the first bracket is already TRUE. Also, the third bracket is TRUE not only when term *loan* or *debt* is detected but also when the first and second brackets are already TRUE.

This feature can be achieved by setting a criterion to expression code setting process. That is, a bit of a code can only be modified unless the preceding bits have been set to 0. This can be done by checking the numerical value of the preceding bits.

V. CONCLUSION AND FUTURE WORK

In this paper, a constant time complexity spam detection algorithm was put forward to boost throughput on rule-based filtering systems and the computational complexity of the proposed algorithm was analyzed. Since each step in the proposed algorithm, including detecting a term and calculating logical operators of expressions, has $O(1)$ time complexity,

the overall time complexity for spam detection is $O(1)$. That is, the speed of the spam detection algorithm presented in this paper is independent of rule size and rule term vocabulary. The experiment results validated the $O(1)$ time complexity as the spam detection rules and its terms increasing.

We are now working on upgrading the algorithm to make it much more flexible. Actually, the ! (NOT operator) can be supported by a simple modification. More operators such as '<=', '<', '>=' or '>' will be supported in the future.

In addition, more technics will be engaged in our future research to work on incremental rules and their vocabulary, including one-pass compressing techniques [29] and rough set [30].

REFERENCES

- [1] Internet World Stats. (Jun. 30, 2019). *Internet Usage Statistics The Internet Big Picture*. Accessed: Feb. 21, 2020. [Online]. Available: <https://www.internetworldstats.com/stats.htm>
- [2] Symantec Corporation. (Jan. 12, 2010). *2000-2009 The Spam Explosion*. Accessed: Feb. 21, 2020. [Online]. Available: <https://www.symantec.com/connect/blogs/2000-2009-spam-explosions>
- [3] V. V. Arutyunov, "Spam: Its past, present, and future," *Sci. Tech. Inf. Process.*, vol. 40, no. 4, pp. 205–211, Apr. 2014.
- [4] S. Corporation. (Feb. 2019). *Internet Security Threat Report 2019 Volume 24*. Accessed: Feb. 21, 2020. [Online]. Available: <https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf>
- [5] H. Shaban. (Sep. 19, 2019). *Nearly Half of Cellphone Calls Will be Scams by 2019, Report Says*. The Washington Post. Accessed: Feb. 21, 2020. [Online]. Available: <https://www.washingtonpost.com/technology/2018/09/19/nearly-half-cellphone-calls-will-be-scams-by-report-says/>
- [6] O. Abayomi-Alli, S. Misra, A. Abayomi-Alli, and M. Odusami, "A review of soft techniques for SMS spam classification: Methods, approaches and applications," *Eng. Appl. Artif. Intell.*, vol. 86, pp. 197–212, Nov. 2019.
- [7] S. M. Abdulhamid, M. S. A. Latiff, H. Chiroma, O. Osho, G. Abdul-Salaam, A. I. Abubakar, T. Herawan, "A review on mobile SMS spam filtering techniques," *IEEE Access*, vol. 5, pp. 15650–15666, 2017.
- [8] E. Mathisen, "Security challenges and solutions in cloud computing," in *Proc. 5th IEEE Int. Conf. Digit. Ecosystems Technol. (IEEE DEST)*, Daejeon, South Korea, Jun. 2011, pp. 208–212.
- [9] The Apache Spam Assassin Group. (2003). *The First Enterprise Open-Source Spam Filter*. Accessed: Dec. 1, 2019. [Online]. Available: <http://spamassassin.apache.org/>
- [10] D. Ruano-Ordás, J. Fdez-Glez, F. Fdez-Riverola, and J. R. Méndez, "Effective scheduling strategies for boosting performance on rule-based spam filtering frameworks," *J. Syst. Softw.*, vol. 86, no. 12, pp. 3151–3161, Dec. 2013.
- [11] D. Ruano-Ordás, J. Fdez-Glez, F. Fdez-Riverola, and J. R. Méndez, "Using new scheduling heuristics based on resource consumption information for increasing throughput on rule-based spam filtering systems," *Softw., Pract. Exper.*, vol. 46, no. 8, pp. 1035–1051, Jul. 2015.
- [12] S. Mostinckx, T. Van Cutsem, S. Timmermont, E. G. Boix, É. Tanter, and W. De Meuter, "RuleSIM: A toolkit for simulating the operation and improving throughput of rule-based spam filter," *Softw., Pract. Exper.*, vol. 46, no. 8, pp. 1091–1108, Aug. 2016.
- [13] H. Drucker, D. Wu, and V. N. Vapnik, "Support vector machines for spam categorization," *IEEE Trans. Neural Netw.*, vol. 10, no. 5, pp. 1048–1054, Sep. 1999.
- [14] V. P. Deshpande, R. F. Erbacher, and C. Harris, "An evaluation of naive Bayesian anti-spam filtering techniques," in *Proc. IEEE SMC Inf. Assurance Secur. Workshop*, Jun. 2007, pp. 9–17.
- [15] M. Z. Gashti, "Detection of spam email by combining harmony search algorithm and decision tree," *Eng. Technology Appl. Sci. Res.*, vol. 7, pp. 1713–1718, Jun. 2017.
- [16] A. A. Aburomman and M. B. Ibne Reaz, "A novel weighted support vector machines multiclass classifier based on differential evolution for intrusion detection systems," *Inf. Sci.*, vol. 414, pp. 225–246, Nov. 2017.
- [17] T. Teraguchi, K. Nakamura, S. Tanaka, and K. Kitano, "Detection method of blog spam based on categorization and time series information," in *Proc. 26th Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA)*, 2012, pp. 801–808.
- [18] O. M. E. Ebadati and F. Ahmadzadeh, "Classification spam email with elimination of unsuitable features with hybrid of GA-naive Bayes," *J. Inf. Knowl. Manage.*, vol. 18, no. 1, Mar. 2019, Art. no. 1950008.
- [19] D. Delvia Arifin, Shaufiah, and M. A. Bijaksana, "Enhancing spam detection on mobile phone short message service (SMS) performance using FP-growth and naive bayes classifier," in *Proc. IEEE Asia Pacific Conf. Wireless Mobile (APWiMob)*, Sep. 2016, pp. 80–84.
- [20] X. Carreras and L. Marquez, "Boosting trees for anti-spam email filtering," in *Proc. 4th Int. Conf. Recent Adv. Natural Lang. Process.*, Sep. 2001, pp. 58–64.
- [21] S. Madisetty and M. S. Desarkar, "A neural network-based ensemble approach for spam detection in Twitter," *IEEE Trans. Comput. Social Syst.*, vol. 5, no. 4, pp. 973–984, Dec. 2018.
- [22] M. Jevtovic. (Jan. 2, 1998). *Filter.Plx: A Context/Keyword Based Spam Filter, Internet Archive WayBack Machine*. Accessed: Feb. 21, 2020. [Online]. Available: <http://web.archive.org/web/19981207030000/antispam.schmooze.net/filter/File:filter.plx>
- [23] G. Robinson. (1997). *The Spamometer, Spamometer*. Accessed: Feb. 21, 2020. [Online]. Available: <http://jon.es/spamometer/File:spamometer.tar.gz>
- [24] A. Garg, R. Battiti, and R. G. Cascella, "May I borrow your filter? Exchanging filters to combat spam in a community," in *Proc. 20th Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, vol. 1, Apr. 2006, pp. 1–5.
- [25] N. Pérez-Díaz, D. Ruano-Ordás, F. Fdez-Riverola, and J. R. Méndez, "Wirebrush4SPAM: A novel framework for improving efficiency on spam filtering services," *Softw., Pract. Exper.*, vol. 43, no. 11, pp. 1299–1318, Jun. 2012.
- [26] J. R. Méndez, M. Reboiro-Jato, F. Díaz, E. Díaz, and F. Fdez-Riverola, "Grindstone4Spam: An optimization toolkit for boosting e-mail classification," *J. Syst. Softw.*, vol. 85, no. 12, pp. 2909–2920, Dec. 2012.
- [27] B. Goi, M. U. Siddiqi, and H.-T. Chuah, "Computational complexity and implementation aspects of the incremental hash function," *IEEE Trans. Consum. Electron.*, vol. 49, no. 4, pp. 1249–1255, Nov. 2003.
- [28] M. Mayzner and E. Shrlidcu. *English Letter Frequency Counts*. Accessed: Dec. 1, 2019. [Online]. Available: <http://norvig.com/mayzner.html>
- [29] C. Hou and Z.-H. Zhou, "One-pass learning with incremental and decremental features," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 11, pp. 2776–2792, Nov. 2018.
- [30] P. K. Roy, J. P. Singh, and S. Banerjee, "Deep learning to filter SMS spam," *Future Gener. Comput. Syst.*, vol. 102, pp. 524–533, Jan. 2020.



TIAN XIA received the M.S. and Ph.D. degrees in computer application from East China Normal University, Shanghai, China, in 2004 and 2007, respectively.

Since 2007, he has been working as the Director of the Digital Media Technology Department, Shanghai Polytechnic University, and promoted to an Associate Professor, in 2009. He also cooperated with companies on Computing Advertisements, Big Data Mining, and Anti-spam technologies. His research interests include machine learning, natural language processing, and deep learning.

• • •