Efficiently Detecting Disguised Web Spambots (with Mismatches) in a Temporally Annotated Sequence

Hayam Alamro

Department of Informatics
King's College London, UK
Department of Information Systems
Princess Nourah bint Abdulrahman University
Riyadh, KSA
email: hayam.alamro@kcl.ac.uk

Costas S. Iliopoulos

Department of Informatics
King's College London, UK
email: costas.iliopoulos
@kcl.ac.uk

***Abstract*—Web spambots are becoming more advanced, utilizing techniques that can defeat existing spam detection algorithms. These techniques include performing a series of malicious actions with variable time delays, repeating the same series of malicious actions multiple times, and interleaving legitimate (decoy) and malicious actions. Existing methods that are based on string pattern matching are not able to detect spambots that use these techniques. In response, we define a new problem to detect spambots utilizing the aforementioned techniques and propose an efficient algorithm to solve it. Given a dictionary of temporally annotated sequences $\overline{S}$ modeling spambot actions, each associated with a time window, a long, temporally annotated sequence T modeling a user action log, and parameters f and k, our problem seeks to detect each degenerate sequence $\tilde{S}$ with c indeterminate action(s) in $\overline{S}$ that occurs in T at least f times within its associated time window, and with at most k mismatches. Our algorithm solves the problem exactly, it requires linear time and space, and it employs advanced data structures, bit masking and the Kangaroo method, to deal with the problem efficiently.**

Keywords–Web spambot; Indeterminate ; Disguised; Actions log.

I. INTRODUCTION

A spambot is a computer program designed to do repetitive actions on websites, servers or social media communities. These actions might be harmful, such as carrying out certain attacks on websites/ servers or may be used to deceive users such as involving irrelevant links to increase a website ranking in search engine results. Spambots can take different forms that are designed according to a spammer desire such as using web crawlers for planting unsolicited material or to collect email addresses from different sources like websites, discussion groups or newsgroups with the intent of building mailing lists for sending unsolicited or phishing emails. Usually, Spammers create fake accounts to target specific websites or domain specific users and start sending predefined designed actions which are known as predefined scripts. Therefore, websites administrators are looking for automated tools to curtail the actions of web spambots. Although there are attempts to prevent spamming using anti-spambots tools, the spammers try to adopt new forms of spambots by manipulating spambots' actions behaviour to appear as it were coming from a legitimate user to bypass the existing spam-filter tools. One of the main popular techniques used in web spambots is *content-based* which inject repetitive keywords in meta tags to promote a website in search engines, as well as *link-based* techniques that add links to a web page to increase its ranking score in search engines. There are several works for preventing the use of content-based or link-based techniques by web spambots [1]–[6]. However, they focus on identifying the content or links added by spambots, rather than detecting the spambot based on their actions. There are also techniques that analyze *spambot behavior* [5] [7]. These techniques utilize supervised machine learning to identify the source of spambot, rather than detecting the spambot. More relevant to our work are string pattern matching-based techniques that detect spambots based on their actions (i.e., based on how they interact with the website these spambots attack) [8] [9]. These techniques model the user log as a large string (sequence of elements corresponding to actions of users or spambots) and common/previous web spambot actions as a dictionary of strings. Then, they perform pattern matching of the strings from the dictionary to the large string. If a match is found, then they state that a web spambot has been detected. For example, the work by Hayati et.al [8] proposes a rule-based, on-the-fly web spambot detection technique, which identifies web spambots by performing string matching efficiently using tries. The work of [9] improves upon [8] by considering spambots that utilize decoy actions (i.e., injecting legitimate actions, typically performed by users, within their spam actions, to make spam detection difficult) and using approximate pattern matching based on the *FPT* algorithm to detect such spambots. However, both [8] and [9] are limited in that they consider consecutive spambot actions. This makes them inapplicable in real settings where a spambot needs to be detected from a log representing actions of both users and spambots, as well as settings where a spambot injects legitimate actions in some random fashion within a time window. The reason that the works of [8] and [9] are inapplicable in these settings is that they do not take into account temporal information of neither the sequence (i.e., the user log) nor the pattern (i.e., the spambot actions). Recently, Artificial Intelligence (AI) has been employed in security purposes to recognize cyber crimes and to reduce the time and money needed for manual tasks and threats monitoring at businesses. In our paper, we use one of the main approaches for AI-based threats detection which is based on monitoring behavior of a stream of data in a log file and try to detect the places of spambots. To the best of our knowledge, our

 ISBN: 978-1-61208-783-2

contribution is novel as no other work takes into account the temporal information of a sequence of actions in a user log, nor the estimated time window of a pattern of actions in a dictionary. These challenges made finding an effective solution to our problem a new challenge as there is no other work addressing the issue of time (the spambot can pause and speed up) or errors (deceptive or unimportant actions). It is worth mentioning that our methods are not comparable to others as they address different issues. The other challenge that we faced was when conducting our laboratory experiments for our algorithm as there was no publicly available data set modeling the real temporal annotated sequence of a user log. The only available is the public data sets (WEBSPAM-UK2006/7) which are a collection of assessments and set of host names and based on the spamicity measure to decide whether a web page is a spam, non-spam or undecided.

In this work, we focus on time annotated sequences, where each action is associated with a time stamp. Our goal is to detect one or more spambots, by finding frequent occurrences of indeterminate spambot actions within a time window that can also occur with mismatches. Our work makes the following specific contributions:

1. We introduce an efficient algorithm that can detect one or more sequences of indeterminate (non solid) actions in text T in linear time. We ignore the temporal factor in describing this algorithm to facilitate the process of clarification and focus on detecting disguised web spambots efficiently. It is worth mentioning that our algorithm can compute all occurrences of a sequence $\tilde{S}$ in text T in $O(m + logn + occ)$, where m is the length of the degenerate sequence $\tilde{S}$, n is the length of the text T and occ is the number of the occurrences of the sequence $\tilde{S}$ in text T.

2. We propose an efficient algorithm for solving (f, c, k, W)-Disguised Spambots Detection with indeterminate actions and mismatches. Our algorithm takes into account temporal information, because it considers time-annotated sequences and because it requires a match to occur within a time window. The latter requirement models the fact that spambots generally perform a series of disguised actions in a relatively short period of time. Our algorithm is a generalization of the previous problem and based on constructing a *generalized enhanced suffix array*, *bit masking* with help of *Kangaroo method* which help in locating indeterminate spambots with mismatches fast. Our proposed algorithm (f, c, k, W)-Disguised Spambots Detection with indeterminate actions can find all occurrences of each $\tilde{S}_i$ in $\overline{S}$, such that $\tilde{S}_i$ occurs in T at least f times within the window W_i of $\tilde{S}_i$ and with at most k mismatches according to Hamming distance.

The rest of the paper as follows. In Section II, detailed literature review, In section III we introduce notations and background concepts. In Section IV, we formally define the problems we address. In Section V, we formally detail our solutions and present our algorithms. In Section VI, we present experimental results. In Section VII, we conclude.

II. LITERATURE REVIEW

Web spam usually refers to the techniques that the spammers used to manipulate search engine ranking results to promote their sites either for advertising purposes, financial benefits or for misleading the user to a malicious content trap or to install malware on victim's machine. For these purposes, spammers can use different techniques such as *content-based* which is the most popular type of web spam, where the spammer tries to increase term frequencies on the target page to increase the score of the page. Another popular technique is through using *link-based*, where the spammer tries to add lots of links on the target page to manipulate the search engine results [10] [11]. Ghiam et al. in [11] classified spamming techniques to link-based, hiding, and content-based, and they discussed the methods used for web spam detection for each classified technique. Roul et al. in [10] proposed a method to detect web spam by using either content-based, link-based techniques or a combination of both. Gyongyi et al. in [12] proposed techniques to semi-automatically differ the good from spam page with the assistance of human expert whose his role is examining small seed set of pages to tell the algorithm which are 'good pages' and 'bad pages' roughly based on their connectivity to the seed ones. Also, Gyongyi et al. in [13] introduced the concept of spam mass and proposed a method for identifying pages that benefit from link spamming. Egele et al. [14] developed a classifier to distinguish spam sites from legitimate ones by inferring the main web page features as essential results, and based on those results, the classifier can remove spam links from search engine results. Furthermore, Ahmed et al. [15] presented a statistical approach to detect spam profiles on online social networks (OSNs). The work in [15] presented a generic statistical approach to identify spam profiles on online social networks. For that, they identified 14 generic statistical features that are common to both Facebook and Twitter and used three classification algorithms (naive Bayes, Jrip and J48) to evaluate features on both individual and combined data sets crawled from Facebook and Twitter networks. Prieto et al. [16] proposed a new spam detection system called Spam Analyzer And Detector (SAAD) after analyzing a set of existing web spam detection heuristics and limitations to come up with new heuristics. Prieto et al. in [16] tested their techniques using Webb Spam Corpus(2011) and WEBSPAM-UK2006/7, and they claimed that the performance of their proposed techniques is better than others system presented in their literature. On the other side, other contributions try to detect web spambot using supervised machining learning. In this regard, Dai et al. [17] used supervised learning techniques to combine historical features from archival copies of the web and use them to train classifiers with features extracted from current page content to improve spam classification. Araujo et al. [18] presented a classifier to detect web spam based on qualified link (QL) analysis and language model (ML) features. The classifier in [18] is evaluated using the public WEBSPAM-UK 2006 and 2007 data sets. The baseline of their experiments was using the precomputed content and link features in a combined way to detect web spam pages, then they combined the baseline with QL and ML based features which contributed to improving the detecting performance. Algur et al. [19] proposed a system which gives spamicity score of a web page based on mixed features of content and link-based. The proposed system in [19] adopts an unsupervised approach, unlike traditional supervised classifiers, and a threshold is determined by empirical analysis to act as an indicator for a web page to be spam or non-spam. Luckner et al. [20] created a web spam detector using features based on lexical items. For that, they created three web spam detectors and proposed new lexical-based features that are trained and tested using WEBSPAM-

UK data sets of 2006 and 2007 separately, then they trained the classifiers using WEBSPAM-UK 2006 data set but they use WEBSPAM-UK 2007 for testing. In the end, the authors based on the results of the first and second detectors as a reference for the third detector where they showed that the data from WEBSPAM-UK 2006 can be used to create classifiers that work stably both on the WEBSPAM-UK 2006 and 2007 data sets. Moreover, Goh et al. [21] exploited web weight properties to enhance the web spam detection performance on a web spam data set WEBSPAM-UK 2007. The overall performance in [21] outperformed the benchmark algorithms up to 30.5% improvement at the host level and $6 - 11\%$ improvement at the page level. At the level of online social networks (OSNs), the use of social media can be exploited negatively as the impact of OSNs has increased recently and has a major impact on public opinion. For example, one of the common ways to achieve media blackout is to employ large groups of automated accounts (bots) to influence the results of the political elections campaigns or spamming other users' accounts. Cresci et al. [22] proposed an online user behavior model represents a sequence of string characters corresponding to the user's online actions on Twitter. The authors in [22] adapt biological DNA techniques to online user behavioral actions which are represented using digital DNA to distinguish between genuine and spambot accounts. They make use of the assumption of the digital DNA fingerprinting techniques to detect social spambots by mining similar sequences, and for each account, they extract a DNA string that encodes its behavioral information from created data set of spambots and genuine accounts. After that, Cresci et al. [23] investigate the major characteristics among group of users in OSNs. The study in [23] is an analysis of the results obtained in DNA-inspired online behavioral modeling in [22] to measure the level of similarities between the real behavioral sequences of Twitter user accounts and synthetic accounts. The results in [23] show that the heterogeneity among legitimate behaviors is high and not random. Later, Cresci et al. in [24] envisage a change in the spambot detection approach from reaction to proaction to grasp the characteristics of the evolved spambots in OSNs using the logical DNA behavioral modeling technique, and they make use of digital DNA representation as a sequence of characters. The proactive scheme begins with modeling known spambot accounts with digital DNA, applying genetic algorithms to extract new generation of synthetic accounts, comparing the current state-of-art detection techniques to the new spambots, then design novel detection techniques.

III. Background and Main Terminologies

Let $T = a_0 a_2 \dots a_{n-1}$ be a string of length $|T| = n$ over an alphabet Σ of size $|\Sigma| = \sigma$. The empty string ε is the string of length 0. For $1 \leq i \leq j \leq n$, $T[i]$ denotes the ith symbol of T, and $T[i, j]$ the contiguous sequence of symbols (called *factor* or *substring*) $T[i]T[i+1] \dots T[j]$. A *substring* $T[i, j]$ is a suffix of T if $j = n$ and it is a prefix of T if $i = 1$. A string p is a *repeat* of T iff p has at least two occurrences in T. In addition p is said to be *right-maximal* in T iff there exist two positions $i < j$ such that $T[i, i+|p|-1] = T[j, j+|p|-1] = p$ and either $j + |p| = n + 1$ or $T[i, i + |p|] \neq T[j, j + |p|]$. A *degenerate or indeterminate string* , is defined as a sequence $\tilde{X} = \tilde{x_0}\tilde{x_1} \dots \tilde{x_{n-1}}$, where $\tilde{x_i} \subseteq \Sigma$ for all $0 \leq i \leq n-1$ and the alphabet Σ is a non-empty finite set of symbols of size $|\Sigma|$. A *degenerate symbol* $\tilde{x}$ over an alphabet Σ is a non-empty subset of Σ, i.e. $\tilde{x} \subseteq \Sigma$ and $\tilde{x} \neq \emptyset$. $|\tilde{x}|$ denotes the size of $\tilde{x}$ and we have $1 \leq \tilde{x} \leq |\Sigma|$. A degenerate string is built over the potential $2^{|\Sigma|} - 1$ non-empty subsets of letters belonging to Σ. If $|\tilde{x}| = 1$, that is $|\tilde{x}|$ repeats a single symbol of Σ, we say that $\tilde{x_i}$ is a *solid symbol* and i is a *solid position*. Otherwise, $\tilde{x_i}$ and i are said to be a *non-solid symbol* and *non-solid position* respectively. For example, $\tilde{X} = ab[ac]a[bcd]bac$ is a degenerate string of length 8 over the alphabet $\Sigma = \{a, b, c, d\}$. A string containing only solid symbols will be called a solid string. A *conservative degenerate string* is a degenerate string where its number of non-solid symbols is upper-bounded by a fixed position constant c [25], [26]. The previous example is a conservative degenerate string with $c = 2$.

A *suffix array* of T is the lexicographical sorted array of the suffixes of a string T i.e., the suffix array of T is an array $SA[1 \dots n]$ in which $SA[i]$ is the i^{th} suffix of T in ascending order [27]–[29]. The major advantages of *suffix arrays* over *suffix trees* is the space as the space needed using suffix trees becomes larger with larger alphabets such as Japanese characters, and it is useful in computing the frequency and location of a substring in a long sequence (corpus) [28]. $LCP(T_1, T_2)$ is the length of the longest common prefix between strings T_1 and T_2 and it is usually used with SA such that $LCP[i] = lcp(T_{SA[i]}, T_{SA[i-1]})$ For all $i \in [1..n]$ [27] [30].

IV. Problems definitions

The two main problems that the paper will address can be defined as follows.

Problem A: Disguised (Indeterminate) Actions

Some spambots might attempt to disguise their actions by varying certain actions. For example, a spambot takes the actions $ABCDEF$, then $ACCDEF$, then $ABDDEF$ etc. This can be described as $A[BC][CD]DEF$. They try to deceive by changing the second and third action. The action $[BC]$ and $[CD]$ are variations of the same sequence. We will call the symbols A, C, D, E, F solid, the symbols $[BC]$ $[CD]$ indeterminate or non-solid and the string $A[BC][CD]DEF$ degenerate string which is denoted by $\tilde{S}$. In fact, they can disguise any of the actions. In this case, we are not concern which actions will be disguised but we assume that the numbers of attempts to disguise is limited. Let us assume that the number of disguised actions is bounded by a constant c. For the moment, we will ignore the temporal factor of the disguises at this problem to facilitate the clarification of the discovery of the spambot actions, and we will consider the temporal factor in describing problem B as it is a generalization of problem A. Actually, we combine both temporal and fake actions discovery by apply both algorithms simultaneously. For now, let us consider the series of actions taking place on the server.

Definition IV.1. Given a sequence $T = a_1 \dots a_n$, find all occurrences of $\tilde{S} = s_1 s_2 \dots s_m$ in T, where s_i might be solid or indeterminate.

Problem B: Disguised Actions (with k Mismatches)

It is a generalization of *Problem A* with k errors such that the sequence of spambot actions $\tilde{S}$ is *degenerate* actions with errors, and the number of errors is bounded by a constant k. Our aim is to detect new suspicious spambots which are

resulting from changing other disguised actions by spammers such that using few mismatches in its spambot actions $\tilde{S}$ to appear like actions issued by a genuine user.

Definition IV.2. Given a sequence $T = a_1 \dots a_n$ and an action sequence $\tilde{S} = s_1 s_2 \dots s_m$, find all occurrences of $\tilde{S}$ in T where s_i might be solid or indeterminate with *hamming distance* between $\tilde{S}$ and T is no more than k mismatches.

V. ALGORITHMS

In the following, we discuss our algorithms for the two aforementioned problems which they include (preprocessing) as preliminary stage.

A. Preprocessing

Our algorithms require as input sequences temporally annotated actions. These temporally annotated sequences are produced from user logs consisting of a collection of *http* requests. Specifically, each request in a user log is mapped to a predefined index key in the sequence and the date-time stamp for the request in the user log is mapped to a time point in the sequence.

B. Problem A: Disguised (indeterminate) actions

In order to design an efficient algorithm for this problem, we need to use the following steps that will make the algorithm fast.

Step 1: For each *non-solid* s_j occurring in degenerate pattern $\tilde{P} = s_1 \dots s_m$, we substitute each s_j with '#' symbol, where '#' is not in Σ. Let $\hat{P}$ be the resulting pattern from substitution process and will be considered as a *solid* pattern, see (Figure 1 **Step1**) and (Table I).

TABLE I. CONVERTING $\tilde{P}$ TO $\hat{P}$

$\tilde{P}$	A	B	[GX]	C	[AD]	F
$\hat{P}$	A	B	$\#_1$	C	$\#_2$	F

Step 2: Compute the *suffix array* for the sequence of actions T. Since the *suffix array* is sorted array of all suffixes of a given string, we can apply *binary search* algorithm with the suffix array to find a pattern of spambot actions in a text of actions in $O(mlogn)$ time complexity, where m is the the length of the pattern P and n is the length of the text T. Our algorithm uses Manber and Myers algorithm which is described in [31], which uses a suffix array for on-line string searches and can answer a query of type "Is P a substring in T?" in time $O(m + logn)$. The algorithm in [31], uses a sorted *suffix array*, *binary search* against the suffix array of T and auxiliary data structures *Llcp, Rlcp* which they are precomputed arrays and hold information about the *longest common prefixes (lcps)* for two subranges *(L ... M)* and *(M ... R)* of binary search. Subsequently, the algorithm speeds up the comparison and permits no more than one comparison for each character in P to be compared with the text T. The method is generalised to $O(m+logn+occ)$ to find all occurrences of P by continuing on the adjacent suffixes to the first occurrence of P in *suffix array*, see (Figure 1 **Step2**).

Step 3: At this stage, we consider each *non-solid* position s_j in $\hat{P}$ which is represented by '#' as an allowed mismatch with the corresponding action a_i in T as '#' is not in Σ. To query whether that a_i belongs to the set of actions in '#', the algorithm uses a *bit masking* operation. For example, suppose we want to see whether the action 'X' in text T belongs to one of the set actions $[GX]$ in $\hat{P}$ which is represented by '$\#_1$', see Table I, we assume that each action in degenerate symbol represents bit '1' among other possible actions, and '0' otherwise. Furthermore, The current compared action a_i in T is always represented by bit '1'. Thereafter, the algorithm uses *And* bit wise operation between the two sets $[GX]$ and $[X]$ such that $[11] \bigwedge [01] = [01]$ which means that $[X] \in [GX]$. To do that, the algorithm uses the *suffix array* and *binary search* to find the pattern match, and for each '#' in the sequence is encountered, the algorithm consider it as an allowed mismatch and get into the *verification* process to check whether the action a_i in T is one of the actions in '#'. However, each *non-solid* position s_j in $\hat{P}$ is numbered sequentially starting from 1 up to the number of indeterminate symbols c. Consequently, we refer to that position for each pattern of spambots actions with the number of the pattern $\hat{P}_r$ and the number of $\#_l$, where $1 \le r \le \Sigma\hat{P}$ and $1 \le l \le \Sigma\# \in \hat{P}_r$, see (Figure 1 **Step3**).

Input: Action sequence T, spambots dictionary $\overline{S}$ where each spambot $\tilde{P} \in \overline{S}$
Output: all matching $\tilde{P}$ found in T

1: **procedure** LOCATE ALL LOCATIONS OF $\tilde{P}$ WITH INDETERMINATE ACTIONS IN T
2: ▷ **Step1: (Substitution)**
3: **for** each $\tilde{P}_r \in \overline{S}$ **do**
4: $\hat{P}_r \leftarrow \tilde{P}_r$
5: $m \leftarrow |\hat{P}_r|$
6: $l \leftarrow 1$
7: **for** $(j = 0\ to\ m-1)$ **do**
8: **if** $(\hat{P_r}[j]\ is\ non\ solid)$ **then**
9: $\hat{P_r}[j] \leftarrow \#_l$
10: $l \leftarrow l + 1$
11: **end if**
12: **end for**
13: **end for**
14: ▷ **Step2: (actions matching)**
15: Build the *suffix array SA* for the text of actions T
16: **for** each $\hat{P}_r \in \overline{S}$ **do**
17: Apply the *binary search* with *LCPs* arrays of **Manber and Myer** in [31]
18: For each current action a_i in SA compared to *non solid* symbol represented by '$\#_l$' in $\hat{P}_r$ ▷ **go to: step 3**
19: **end for**
20: ▷ **Step3: (verification process)**
21: $mask = 1 \bigwedge hashMatchTable[\hat{P}_r \#_l][ascii[a_i]]$
22: **if** $mask = 1$ **then**
23: continue
24: **else** ▷ not match
25: exit matching
26: **end if**
27: **end procedure**

Figure 1: Problem A: Locate spambots with indeterminate actions

Verification process: At this stage, the algorithm does a bit level masking operation using the logical operator 'And' between the current compared action a_i in T and the corresponding *non-solid* position in $\hat{P}$ which is represented by '$\#_l$'. As we mentioned before, the algorithm assumes each current compared action a_i in T is represented by a bit '1', and each '$\#_l$' of each pattern reveals its original set of actions by setting bit '1' at each action belongs to its set and '0' otherwise using a match table called *hashMatchTable*, see Table II. To access the corresponding column in *hashMatchTable* directly, the columns are indexed by the *(ascii code)* of each character

belongs to the actions alphabets in Σ and ordered from 65 to 90 which are the ascii code of capital letters (or 97 to 122 for small letters). Thus, the algorithm can apply the following formula ($1 \bigwedge hashMatchTable[\hat{P}_r\#_l][ascii[a_i]]$) to find whether that current comparing action a_i in T has a match with one of the actions in '$\#_l$' where '1' is the corresponding bit of a_i, see (Figure 1 **Step3**) and (Table II).

TABLE II. HASHMATCHTABLE OF THE PATTERN $\tilde{P}_1 = AB[GX]C[AD]F$ WHERE ITS COVERSION IS $\hat{P}_1 = AB\#_1C\#_2F$

asscii(a_i)	65	66	67	68	..	71	…	88	89	90
a_i	A	B	C	D	..	G	…	X	Y	Z
$\hat{P}_1\#_1$	0	0	0	0		1	…	1	0	0
$\hat{P}_1\#_2$	1	0	0	1		0	…	0	0	0
…	…	…	…	…		…	…	…	…	…
$\hat{P}_r\#_l$	…	…	…	…		…	…	…	…	…

Theorem 1. Algorithm (Figure 1) computes the occurrence of the pattern $\hat{P}$ in text T in $O(mlogn)$ time using *suffix array* with *binary search*. □

Theorem 2. Algorithm (Figure 1) can compute all occurrences of the pattern $\hat{P}$ in text T in $O(m + logn + occ)$ time using an *enhanced suffix array* with auxiliary data structure *LCP*, *binary search* and *bit masking*. □

C. Problem B: Disguised Actions (with k Mismatches)

The problem we solve is referred to as (f, c, k, W)-Disguised Spambots Detection which is a generalization of problem A and defined as follows:

Problem. (f, c, k, W)-Disguised Spambots Detection with indeterminate actions. Given a temporally annotated action sequence T (a_j, t_j), a dictionary $\overline{S}$ containing sequences $\hat{S}_i$ each has a c non-solid symbol (represented by $\#$), associated with a time window W_i, a minimum frequency threshold f, and a maximum Hamming distance threshold k, find all occurrences of each $\hat{S}_i \in \overline{S}$ in T, such that each $\hat{S}_i$ occurs: (I) at least f times within its associated time window W_i, and (II) with at most k mismatches according to Hamming distance.

The problem we introduce in our work considers spambots that perform indeterminate sequence of malicious actions multiple times. Thus, we require an indeterminate sequence which has c non-solid symbol(s) to appear at least f times and within a time window W_i, to attribute it to a spambot. In addition, we consider spambots that perform decoy actions, typically performed by real users. To take this into account, we consider mismatches. We assume that the dictionary and parameters are specified based on domain knowledge (e.g., from external sources or past experience).

1) Our algorithm for solving (f, c, k, W)-Disguised Spambots Detection: The algorithm is based on constructing a *generalized enhanced suffix array* data structure, *bit masking* with help of *Kangaroo method* [32], to find the *longest common subsequence* LCS between a sequence of actions in T and an action sequence $\hat{S}_i$ with at most k mismatches in linear time.

Definition V.1. The *Enhanced suffix array (ESA)* is a data structure consisting of a suffix array and additional tables which can be constructed in linear time and considered as an alternative way to construct a *suffix tree* which can solve pattern matching problems in optimal time and space [33], [34].

Definition V.2. The *Generalized enhanced suffix array (GESA)* is simply an enhanced suffix array for a set of strings, each one ending with a special character and usually is built to find the *longest common sequence* LCS of two strings or more. *GESA* is indexed as a pair of identifiers (i_1, i_2), one identifying the string number, and the other is the lexicographical order of the string suffix in the original concatenation strings [35].

To do so, we start with algorithm (Figure 2). First, our algorithm extracts the actions of the temporally annotated action sequence T into a sequence T_a such that it contains only the actions $a_0 \dots a_n$ from T (step 2). Then, we generalize the enhanced suffix array to a collection of texts T_a and set of action sequences $\overline{S_{\hat{S}_i}}$ separated by a special delimiter at the end of each sequence (step 5) as follows:

$$GESA(T_a, \overline{S_{\hat{S}_i}}) = T_a!_0\hat{S}_1!_1\hat{S}_2!_2 \dots \hat{S}_r!_r$$

Such that, $\hat{S}_1 \dots \hat{S}_r$ are set of spambots sequences that belong to dictionary $\overline{S_{\hat{S}_i}}$, and $!_0, \dots, !_r$ are special symbols not in Σ and smaller than any alphabetical letter in T_a and smaller than '#' with respect to an alphabetical order. We will refer to a collection of tables (GESA, GESA^R, LCS, T, $\overline{S_{\hat{S}_i}}$) to find disguised spambots within a time window t such that given a temporally annotated action sequence $T = (a_0, t_0), (a_1, t_1) \dots (a_n, t_n)$, an action sequence $\hat{S} = s_1 \dots s_m$

Input: Temporally annotated action sequence T, spambot dictionary $\overline{S}$, k, f
Output: All occurrences for each spambot $\hat{S}_i$ in dictionary $\overline{S}$

1: **procedure** DISGUISED SPAMBOTS DETECTION WITH K MISMATCHES
2: $T_a \leftarrow$ *all extracted action sequences with same their order from* T
3: $n \leftarrow |T_a|$
4: // Create $GESA$, where each index consists of a pair (i_1, i_2)
5: $GESA(T_a, \overline{S_{\hat{S}_i}}) \leftarrow T_a!_0\hat{S}_1!_1\hat{S}_2!_2 \dots \hat{S}_r!_r$
6: Create $GESA^R$ from $GESA$
7: Initialize $hashMatchTable[no.of\#][26]$
8: **for each** spambot sequence $\hat{S}_i \in \overline{S}$ **do** ▷ Start matching
9: $occ \leftarrow 0, occur[] \leftarrow empty, sus\_spam \leftarrow empty$
10: // Calculate LCS between $\hat{S}_i$ and T_a
11: $m \leftarrow GESA^R[i], (m_1, m_2) \leftarrow GESA[m].(i_1, i_2)$
12: // Find the closest T_a sequence suffix j which is identified by $i_1 = 0$ and closest to m either before or after m
13: $j \leftarrow m - 1$ ▷ $(j \leftarrow m + 1)$ in case closest j is after m
14: `Find_Occ:`
15: **while** $j \geq 0$ and $i_1 \neq 0$ **do** ▷ $(j < n)$ & $(i_1 \neq 0)$ in case closest j is after m
16: $j \leftarrow j - 1$ ▷ $j \leftarrow j + 1$ in case closest j is after m
17: **end while**
18: **if** $j \geq 0$ and $i_1 = 0$ **then** ▷ $((j < n)$ & $(i_1 = 0))$ in case closest j is after m
19: $(j_1, j_2) \leftarrow GESA[j].(i_1, i_2)$
20: Find_LCS(T_a, $\hat{S}_i$, j_2, m_2, n, k, occ, $occur[]$, $\overline{S}$, $sus\_spam, hashMatchTable$)
21: $j \leftarrow j - 1$ ▷ $j \leftarrow j + 1$ in case closest j is after m
22: **if** $j \geq 0$ **then** ▷ $j < n$ in case closest j is after m
23: // Find other occurrences of suspicious spambot from $\hat{S}_i$
24: **go to** `Find_Occ`
25: **else**
26: Output sus_spam,$occur[]$
27: **end if**
28: **end if**
29: **end for**
30: **end procedure**

Figure 2: Problem B: Disguised spambots detection with k mismatches

and an integer t, we will compute $j_1, j_2, \ldots, j_m$ such that $a_{j_i} = s_i,\ 1 \leq i \leq m$ and $\sum_{i=1}^{m} t_{j_i} < t$ or $t_{j_m} - t_{j_1} < t$ with Hamming distance between T_a and $\hat{S}$ no more than k mismatches. For example, suppose we have the following sequence actions:
$T_a = ABBABGCDFCBACAFAABGDFF$ and an indeterminate spambot sequence: $\hat{S} = B\#_1C\#_2F$, where $\#_1 = [GX]$ and $\#_2 = [AD]$ in the original sequence $\tilde{S}$. Hence, the $GESA(T_a, \hat{S}) = T_a!_0\hat{S}!_1$, where all sequences are concatenated in one string separated with a unique delimiter ! and the reference indexing of the GESA will consist of 29 index as shown in Figure 3. Our algorithm includes initializa-

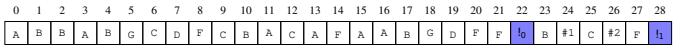

Figure 3: Concatenation strings of $T_a!_0\hat{S}!_1$

tion for *hashMatchTable* to do *bit masking* (see Figure 2, step 7). For each spambot sequence $\hat{S}_i$ in the spambots dictionary $\overline{S_{\hat{S}_i}}$, the algorithm calculates the *longest common sequence* LCS between $\hat{S}_i$ and T_a starting at position 0 in sequence $\hat{S}_i$ and position j in sequence T_a such that the common substring starting at these positions is maximal (see Figure 2, steps 8-24). Since the suffixes of these two sequences are represented in the lexicographical order at $GESA(T_a, \overline{S_{\hat{S}_i}})$, we need to look up the closest suffix j (which belongs to the other sequence in T_a) to the sequence $\hat{S}_i$. This can be achieved by using $GESA^R$ table which retains all the lexicographical ranks of the suffixes of the $GESA$ (see Figure 2, step 6). After locating the suffix index of the pattern $\hat{S}_i$, (see Figure 5, rank 10 in $GESA^R(i)$ column as an example), then, the closest suffix j will be the closest neighbour to that suffix and belongs to the sequence T_a (which is identified by an integer number i_1 and equal to 0). More precisely, the length of the longest common sequence at position $GESA(i)$ and matching substring of $GESA(j)$ is given as follows:

$$LCS(\hat{S}_i, T_a) = max(LCP(GESA(i_1, i_2), GESA(j_1, j_2)) = l_0$$

Where l_0 is the maximum length of the *longest common prefix* matching characters between $GESA(i_1, i_2)$ and $GESA(j_1, j_2)$ until the first mismatch occur (or one of the sequences terminates). Next, the second step in our algorithm involves finding the length of the longest common subsequence starting at the previous mismatch position l_0 which can be achieved using *Kangaroo method* (see Figure 4) as follows:

$$max(LCP(GESA(i_1, i_2 + l_0 + 1), GESA(j_1, j_2 + l_0 + 1)) = l_1$$

Where l_1 is the maximum length of the *longest common prefix* matching characters between $GESA(i_1, i_2 + l_0 + 1)$ and $GESA(j_1, j_2 + l_0 + 1)$ until the second mismatch occur (or one of the sequences terminates). Once our algorithm encounters '#' at the pattern, it will get into the verification process (see Figure 4, steps 27-32) that we described in problem A (using *bit masking* and *hashMatchTable*). Our algorithm will continue in using *Kangaroo method* to find other k mismatches until the number of mismatches is greater than k or one of the sequences terminates. Subsequently, to find other occurrences of the spambot $\hat{S}_i$ in T_a, Figure 2 continues finding the second closest suffix j that belongs to the sequence T_a simply from $GESA$, see Figure 5.

1: **function** FIND_LCS(T_a, $\hat{S}_i$, j_2, m_2, n,k, occ, ref : $occur[]$, $\overline{S}$, ref : $sus\_spam, hashMatchTable[][26]$) ▷ ref: to return parameter's value to algorithm Figure 3
2: $l \leftarrow 0$; $k\_mis \leftarrow 0$
3: **while** $k\_mis < k$ and $l < n$ and $l < |\hat{S}_i|$ **do**
4: **while** $T_a[j_2 + l] = \hat{S}_i[0 + l]$ **do**
5: $sus\_spam \leftarrow sus\_spam + T_a[j_2 + l]$
6: $l \leftarrow l + 1$
7: **end while**
8: **if** $\hat{S}_i[0 + l] =' \#'$ **then**
9: **go to** Verification process
10: **if** match **then**
11: $sus\_spam \leftarrow sus\_spam + T_a[j_2 + l]$
12: $l \leftarrow l + 1$
13: **else**
14: $k\_mis \leftarrow k\_mis + 1$
15: **end if**
16: **end if**
17: **end while**
18: **if** $|sus\_spam| = |\hat{S}_i|$ **then**
19: $a\_time \leftarrow 0$
20: **for** $pos = j_2$ to $|\hat{S}_i|$ **do**
21: $a\_time \leftarrow a\_time + T[t_{pos}]$
22: **end for**
23: **if** $a\_time \leq W_i$ **then**
24: $occ \leftarrow occ + 1$; $occur[occ] \leftarrow j_2$
25: **end if**
26: **end if**
27: ▷ **Verification process:**
28: **if** $hashMatchTable[\hat{S}_i[0 + l]][ascii[T_a[j_2 + l]]] = 1$ **then**
29: $match \leftarrow true$
30: **else**
31: $match \leftarrow false$
32: **end if**
33: **end function**

Figure 4: Problem B: LCS with *Kangaroo* method

i	$GESA[i]$	Suffix	$GESA^R[i]$
0	(1,28)	$!_1$	5
1	(0,22)	$!_0b\#_1c\#_2f!_0$	13
2	(1,24)	$\#_1c\#_2f!_1$	11
3	(1,26)	$\#_2f!_1$	6
4	(0,15)	$aabgdff!_0b\#_1c\#_2f!_1$	14
5	(0,0)	$abbabgcdfcbacafaabgdff!_0b\#_1c\#_2f!_1$	27
6	(0,3)	$abgcdfcbacafaabgdff!_0b\#_1c\#_2f!_1$	19
7	(0,16)	$abgdff!_0b\#_1c\#_2f!_1$	20
8	(0,11)	$acafaabgdff!_0b\#_1c\#_2f!_1$	25
9	(0,13)	$afaabgdff!_0b\#_1c\#_2f!_1$	18
10	**(1,23)**	$b\#_1c\#_2f!_1$	12
11	(0,2)	$babgcdfcbacafaabgdff!_0b\#_1c\#_2f!_1$	8
12	**(0,10)**	$bacafaabgdff!_0b\#_1c\#_2f!_1$	17
13	(0,1)	$bbabgcdfcbacafaabgdff!_0b\#_1c\#_2f!_1$	9
14	**(0,4)**	$bgcdfcbacafaabgdff!_0b\#_1c\#_2f!_1$	24
15	**(0,17)**	$bgdff!_0b\#_1c\#_2f!_1$	4
16	(1,25)	$c\#_2f!_1$	7
17	(0,12)	$caf\ aabgdff!_0b\#_1c\#_2f!_1$	15
18	(0,9)	$cbacafaabgdff!_0b\#_1c\#_2f!_1$	28
19	(0,6)	$cdfcbacafaabgdff!_0b\#_1c\#_2f!_1$	21
20	(0,7)	$dfcbacafaabgdff!_0b\#_1c\#_2f!_1$	26
21	(0,19)	$dff!_0b\#_1c\#_2f!_1$	23
22	(1,27)	$f!_1$	1
23	(0,21)	$f\ !_0b\#_1c\#_2f!_1$	**10**
24	(0,14)	$faabgdff!_0b\#_1c\#_2f!_1$	2
25	(0,8)	$fcbacafaabgdff!_0b\#_1c\#_2f!_1$	16
26	(0,20)	$f\ f!_0b\#_1c\#_2f!_1$	3
27	(0,5)	$gcdfcbacafaabgdff!_0b\#_1c\#_2f!_1$	22
28	(0,18)	$gdff!_0b\#_1c\#_2f!_1$	0

Figure 5: $GESA$ for the sequences T_a,$\hat{S}$ and illustration of occurrences of $\hat{S}$ in T_a at $i = 12, 14$ and 15 and $k = 2$, where $\#_1 = [GX]$ and $\#_2 = [AD]$

As we can see from Figure 5, there are three occurrences for spambot $\hat{S}$ in T_a with up to $k = 2$ mismatches. The first occurrence is illustrated using blue color at $i = 12$ with one mismatch ($k = 1$). The second occurrence is illustrated using violet color at $i = 14$ with zero mismatch ($k = 0$) and the last

 ISBN: 978-1-61208-783-2

occurrence is illustrated using green color at $i = 15$ with two mismatches ($k = 2$). All curved arrows represent the *Kangaroo* jumps, and the underlines represent mismatches places. Finally, at each occurrence of $\hat{S}_i$ in the sequence T_a, algorithm (Figure 4) checks its time window using the dictionary $\overline{S}$ and T such that it sums up each time t_i associated with its action a_i in T starting at the position j_2 in $GESA(j_1, j_2)$ until the length of the spambot $|\hat{S}_i|$ and compares it to its time window W_i. If the resultant time is less than or equal to W_i, algorithm (Figure 4) considers that the pattern sequence corresponds to a spambot and terminates (see Figure 4, steps 18-26), so that the control returns to algorithm (Figure 2). Once the frequency number hits a predefined threshold f, the sequence and its occurrences will be output by algorithm Figure 2.

VI. Experimental evaluation

We implemented our algorithm in C++ and ran it on an Intel i7 at 3.6 GHz. Our algorithm uses linear time and space. It constructs the suffix array by *almost pure Induced-Sorting* [36], to build our GESA for the concatenated sequences (T_a and data dictionary $\overline{S}$), and then it applies the *bit masking* and *Kangaroo* method to detect all uncertain actions. We use synthetic data to test our method, because we are not aware of publicly available datasets of real spambot behavior and modeling the real temporal annotated sequence of a user log.

Table III. Results for the number of detected desguised spambots and runtime. Dictionary size $|\overline{S}| = 100$, 200 and 500 (include 20, 50 and 100 disguised sequences respectively) with $f = 2$, and Hamming distance threshold $k = 0$, 1 and 2.

$\|\overline{S}\| : \|\#\|$	$\|T\|$	$\|T_{inj}\|$	Disguised actions			Time (min)		
			$k=0$	$k=1$	$k=2$	$k=0$	$k=1$	$k=2$
100 : 20	10,000	13,004	85	142	376	0.036	0.038	0.043
	25,000	28,100	97	152	302	0.223	0.225	0.237
	50,000	52,924	93	154	542	0.890	0.894	0.974
	100,000	103,028	90	161	758	3.705	3.751	3.795
200 : 50	10,000	16,060	206	313	976	0.137	0.148	0.160
	25,000	30,876	306	422	1726	0.555	0.582	0.586
	50,000	55,856	202	258	946	1.976	1.957	2.094
	100,000	105,878	203	337	1512	7.422	7.781	7.966
500 : 100	10,000	25,088	630	856	2141	2.294	2.255	2.367
	25,000	39,736	602	811	2488	2.440	2.575	2.582
	50,000	64,666	582	791	2854	6.621	6.779	6.929
	100,000	114,812	640	902	4110	21.515	21.744	22.207

However, the use of synthetic data does not affect our findings, because our method is guaranteed to detect all specified patterns in any temporally annotated sequence. Also note that we do not compare with existing works because no existing method can solve this problem. We used a random string generator to generate: (I) temporally annotated sequences with 26 distinct characters and sizes in $\{10000, 25000, 50000, 100000\}$, and (II) dictionaries with size in $\{100, 200, 500\}$. In every generated temporally annotated sequence T, we injected each sequence in $\overline{S}$ to random locations $f \in \{2, 5, 10\}$ times, to obtain the web-spambots user log T_{inj}. We also changed some sequences in $\overline{S}$, to simulate disguised and mismatch actions. Specifically, we replaced 20% of actions of selected sequences in $\overline{S}$ by '#' symbol which represents an indeterminate symbol that corresponds to one or more action in T_{inj}, to simulate disguised actions. Also, we changed one random element in 25% of randomly selected sequences, to simulate a mismatch, and two random elements in another 25% of randomly selected sequences, to simulate two mismatches. The window length (W_i) for each sequence in $\overline{S}$ was selected randomly in $[5, 125]$. Table III shows the impact of the dictionary size $|\overline{S}|$ on the disguised spambot actions (with mismatches) and runtime. From this table we observe the following:

- For the same ($|\overline{S}| : |\#|$) and $|T|$ and varying k, the number of detected actions increases, as our algorithm by construction detects all actions with at most k mismatches (so the actions for a larger k include those for all smaller k's). Interestingly, the sequences which have disguised actions (represented by '#' symbol) are detected as the normal sequences due to the use of *bit masking* operation and *hashMatchTable*. Also, it is noticeable that the runtime is hardly affected by k, due to the use of the *Kangaroo* method, which effectively speeds up the finding of occurrences of spambots. For example, the time for the disguised spambots detection at $k = 1$ and $k = 2$, when ($|\overline{S}| : |\#| = 500 : 100$) and ($|T| = 25,000$) is relatively the same, despite the big difference in the number detected spambots.
- For the same ($|\overline{S}| : |\#|$) and k and varying $|T|$, the number of detected actions does not directly depend on $|T|$; it depends on the size of the injected actions to T, which generally increases with ($|\overline{S}| : |\#|$). The runtime increases with $|T|$, since our algorithm always detects all actions in T.
- For the same $|T|$ and k and varying ($|\overline{S}| : |\#|$), the number of detected actions increase with ($|\overline{S}| : |\#|$) as our algorithm always detects all actions in the dictionary, and the runtime also increases with the dictionary size.

Table IV shows the impact of the frequency f on the disguised spambot actions and runtime. For this table, we observe the following:

Table IV. Results for the number of detected disguised spambots and runtime. Dictionary size ($|\overline{S}| : |\#| = 100 : 20$) with $f = 2$, 5 and 10, and Hamming distance threshold $k = 0$, 1 and 2.

f	$\|T\|$	$\|T_{inj}\|$	Disguised actions			Time (min)		
			$k=0$	$k=1$	$k=2$	$k=0$	$k=1$	$k=2$
2	10,000	13,004	85	142	376	0.036	0.038	0.043
	25,000	28,100	97	152	302	0.223	0.225	0.237
	50,000	52,924	93	154	542	0.890	0.894	0.974
	100,000	103,028	90	161	758	3.705	3.751	3.795
5	10,000	17,510	209	345	715	0.085	0.088	0.092
	25,000	32,750	243	376	497	0.345	0.353	0.357
	50,000	57,310	234	379	894	1.073	1.091	1.144
	100,000	107,570	224	382	1075	4.184	4.147	4.247
10	10,000	25,020	417	681	1264	0.179	0.191	0.203
	25,000	40,500	488	751	1149	0.514	0.532	0.539
	50,000	64,620	466	751	1475	0.528	1.494	1.456
	100,000	115,140	451	755	1609	4.713	4.948	5.045

- For the same f and $|T|$ and varying k, the detected actions for any k include all actions with at most k mismatches. Also, the runtime is hardly affected by k and disguised actions, due to the use of the *Kangaroo* method and *bit masking* operation.
- For the same f and k and varying $|T|$, the number of detected actions does not increase with $|T|$, as it depends on the size of injected actions which generally depends on ($|\overline{S}| : |\#|$), as explained above. The runtime increases because our algorithm always detects all actions in T.

 ISBN: 978-1-61208-783-2

- For the same $|T|$ and k and varying f, the number of detected actions increases as our algorithm always detects all injected actions. Interestingly though the algorithm scales well with f, especially when $|T|$ is large. This is due to the use of the Generalized enhanced suffix array (GESA) which finds all subsequent occurrences of a detected action occurrence directly from the adjacent suffixes to the first occurrence suffix.

VII. Conclusion

We have introduced two efficient algorithms that can detect spambots of malicious actions with variable time delays. One can detect one or more indeterminate sequences in a web user log using *Manber and Myers* algorithm and *bit masking* operation. The second proposed a generalized solution for solving (f, c, k, W)-Disguised Spambots Detection with indeterminate actions and mismatches. Our algorithm takes into account temporal information, because it considers time-annotated sequences and because it requires a match to occur within a time window. The problem seeks to find all occurrences of each conservative degenerate sequence corresponding to a spambot that occurs at least f times within a time window and with up to k mismatches. For this problem, we designed a linear time and space inexact matching algorithm, which employs the *generalized enhanced suffix array* data structure, *bit masking* and *Kangaroo* method to solve the problem efficiently.

References

[1] J. Yan and A. S. El Ahmad, "A low-cost attack on a microsoft captcha," in CCS. ACM, 2008, pp. 543–554.

[2] A. Zinman and J. S. Donath, "Is britney spears spam?" in CEAS, 2007.

[3] S. Webb, J. Caverlee, and C. Pu, "Social honeypots: Making friends with a spammer near you." in CEAS, 2008, pp. 1–10.

[4] P. Heymann, G. Koutrika, and H. Garcia-Molina, "Fighting spam on social web sites: A survey of approaches and future challenges," IEEE Internet Computing, vol. 11, no. 6, 2007, pp. 36–45.

[5] P. Hayati, K. Chai, V. Potdar, and A. Talevski, "Behaviour-based web spambot detection by utilising action time and action frequency," in International Conference on Computational Science and Its Applications, 2010, pp. 351–360.

[6] F. Benevenuto, T. Rodrigues, V. Almeida, J. Almeida, C. Zhang, and K. Ross, "Identifying video spammers in online social networks," in International workshop on Adversarial information retrieval on the web. ACM, 2008, pp. 45–52.

[7] A. H. Wang, "Detecting spam bots in online social networking sites: a machine learning approach," in CODASPY, 2010, pp. 335–342.

[8] P. Hayati, V. Potdar, A. Talevski, and W. Smyth, "Rule-based on-the-fly web spambot detection using action strings," in CEAS, 2010.

[9] V. Ghanaei, C. S. Iliopoulos, and S. P. Pissis, "Detection of web spambot in the presence of decoy actions," in IEEE International Conference on Big Data and Cloud Computing, 2014, pp. 277–279.

[10] R. K. Roul, S. R. Asthana, M. Shah, and D. Parikh, "Detecting spam web pages using content and link-based techniques," Sadhana, vol. 41, no. 2, 2016, pp. 193–202.

[11] S. Ghiam and A. N. Pour, "A survey on web spam detection methods: taxonomy," arXiv preprint arXiv:1210.3131, 2012.

[12] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen, "Combating web spam with trustrank," in Proceedings of the 30th international conference on very large data bases (VLDB), 2004.

[13] Z. Gyongyi, P. Berkhin, H. Garcia-Molina, and J. Pedersen, "Link spam detection based on mass estimation," in Proceedings of the 32nd international conference on Very large data bases. VLDB Endowment, 2006, pp. 439–450.

[14] M. Egele, C. Kolbitsch, and C. Platzer, "Removing web spam links from search engine results," Journal in Computer Virology, vol. 7, no. 1, 2011, pp. 51–62.

[15] F. Ahmed and M. Abulaish, "A generic statistical approach for spam detection in online social networks," Computer Communications, vol. 36, no. 10-11, 2013, pp. 1120–1129.

[16] V. M. Prieto, M. Álvarez, and F. Cacheda, "Saad, a content based web spam analyzer and detector," Journal of Systems and Software, vol. 86, no. 11, 2013, pp. 2906–2918.

[17] N. Dai, B. D. Davison, and X. Qi, "Looking into the past to better classify web spam," in Proceedings of the 5th international workshop on adversarial information retrieval on the web, 2009, pp. 1–8.

[18] L. Araujo and J. Martinez-Romo, "Web spam detection: new classification features based on qualified link analysis and language models," IEEE Transactions on Information Forensics and Security, vol. 5, no. 3, 2010, pp. 581–590.

[19] S. P. Algur and N. T. Pendari, "Hybrid spamicity score approach to web spam detection," in International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME-2012). IEEE, 2012, pp. 36–40.

[20] M. Luckner, M. Gad, and P. Sobkowiak, "Stable web spam detection using features based on lexical items," Computers & Security, vol. 46, 2014, pp. 79–93.

[21] K. L. Goh, R. K. Patchmuthu, and A. K. Singh, "Link-based web spam detection using weight properties," Journal of Intelligent Information Systems, vol. 43, no. 1, 2014, pp. 129–145.

[22] S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, and M. Tesconi, "Dna-inspired online behavioral modeling and its application to spambot detection," IEEE Intelligent Systems, vol. 31, no. 5, 2016, pp. 58–64.

[23] ——, "Exploiting digital dna for the analysis of similarities in twitter behaviours," in 2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA). IEEE, 2017, pp. 686–695.

[24] S. Cresci, M. Petrocchi, A. Spognardi, and S. Tognazzi, "From reaction to proaction: Unexplored ways to the detection of evolving spambots," in Companion Proceedings of the The Web Conference 2018, 2018, pp. 1469–1470.

[25] C. Iliopoulos, R. Kundu, and S. Pissis, "Efficient pattern matching in elastic-degenerate strings," arXiv preprint arXiv:1610.08111, 2016.

[26] M. Crochemore, C. S. Iliopoulos, R. Kundu, M. Mohamed, and F. Vayani, "Linear algorithm for conservative degenerate pattern matching," Engineering Applications of Artificial Intelligence, vol. 51, 2016, pp. 109–114.

[27] S. J. Puglisi, W. F. Smyth, and A. H. Turpin, "A taxonomy of suffix array construction algorithms," acm Computing Surveys (CSUR), vol. 39, no. 2, 2007, pp. 4–es.

[28] M. Yamamoto and K. W. Church, "Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus," Comput. Linguist., vol. 27, no. 1, Mar. 2001, pp. 1–30.

[29] J. Kärkkäinen, P. Sanders, and S. Burkhardt, "Linear work suffix array construction," JACM, vol. 53, no. 6, 2006, pp. 918–936.

[30] T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park, "Linear-time longest-common-prefix computation in suffix arrays and its applications," in Annual Symposium on Combinatorial Pattern Matching. Springer, 2001, pp. 181–192.

[31] U. Manber and G. Myers, "Suffix arrays: a new method for on-line string searches," siam Journal on Computing, vol. 22, no. 5, 1993, pp. 935–948.

[32] M. Nicolae and S. Rajasekaran, "On pattern matching with k mismatches and few don't cares," IPL, vol. 118, 2017, pp. 78–82.

[33] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch, "Replacing suffix trees with enhanced suffix arrays," J. Discrete Algorithms, vol. 2, 2004, pp. 53–86.

[34] M. Abouelhoda, S. Kurtz, and E. Ohlebusch, Enhanced Suffix Arrays and Applications, 12 2005, pp. 7–1.

[35] F. A. Louza, G. P. Telles, S. Hoffmann, and C. D. Ciferri, "Generalized enhanced suffix array construction in external memory," AMB, vol. 12, no. 1, 2017, p. 26.

[36] G. Nong, S. Zhang, and W. H. Chan, "Linear suffix array construction by almost pure induced-sorting," in DCC, 2009, pp. 193–202.

 ISBN: 978-1-61208-783-2