Spambots: Creative Deception

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*—In this paper, we present our spambot overview on the creativity of the spammers, and the most important techniques that the spammer might use to deceive current spambot detection tools to bypass detection. 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 and malicious actions. In response, we define our problems to detect the aforementioned techniques in addition to disguised and "don't cares" actions. Our proposed algorithms to solve those problems are based on advanced data structures that are able to detect malicious actions efficiently in linear time, and in an innovative way.**

Keywords–Spambot; Temporally annotated sequence; Creative; Deception.

I. INTRODUCTION

A *bot* is a software application that is designed to do certain tasks. Bots usually consume network resources by accomplishing tasks that are either beneficial or harmful. According to Distil Networks' published report '2020 Bad Bot Report', prepared by data from Imperva's Threat Research Lab, the bots make up 40% of all online traffic, while the human driven traffic makes up to 60%. The report shows that the bad bots traffic has risen to 24.1% (A rise of 18.1% from 2019), while good bots traffic decreased to 13.1% (A 25.1% decrease from 2018) [1]. The good bots, for example, are essential in indexing the contents of search engines for users' search, and recently have been used by some companies and business owners to improve customer services and communications in a faster way by employing the use of Artificial Intelligence (AI). These bots are useful in large businesses or businesses with limited resources through the use of *chatbots*, which can benefit the organization in automating customer services like replying to questions and conducting short conversations just like a human. However, bots can be harmful in what is known as a *spambot*. A spambot is a computer program designed to do repetitive actions on websites, servers, or social media communities. These actions can be harmful by 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 the search engine results. Spambots can take different forms which are designed according to a spammer's desire. They can take the form of web crawlers to plant unsolicited material or to collect email addresses from different sources like websites, discussion groups, or news groups with the intent of building mailing lists to send unsolicited or phishing emails. Furthermore, spammers can create fake accounts to target specific websites or domain specific users and start sending predefined designed actions which are known as scripts. Moreover, spammers can work on spreading malwares to steal other accounts or scan the web to obtain customers contact information to carry out *credential stuffing* attacks, which is mainly used to login to another unrelated service. In addition, spambots can be designed to participate in deceiving users on online social networks through the spread of fake news, for example, to influence the poll results of a political candidate. 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 creative spambots by manipulating spambots actions' behaviour to appear as it was coming from a legitimate user to bypass the existing spam-filter tools.

This work falls under the name of digital creativity where the *http* requests at the user log can be represented as temporally annotated sequences of actions. This representation helps explore repeated patterns of malicious actions with different variations of interleaving legitimate actions and malicious actions with variable time delays that the spammer might resort to deceive and bypass the existing spam detection tools. Consequently, our proposed algorithms for tackling the creative deception techniques are based on advanced data structures and methods to keep the performance and efficiency of detecting creative deception sequences at first priority. Here, we are going to provide a summary of the creativity of the spambot programmers, and the most important creative techniques that the spammer might use to deceive current spambot detection tools. Then, we are going to present our proposed solutions at this field to tackle those problems based on employing the AI approach to monitor the behaviour of the stream of actions using advanced data structures for pattern matching and to uncover places of the spambot attacks.

In Section II, we detail the related works in the literature review. In Section III, we introduce notations, background concepts, and formally define the problems we address. In Section IV, we present our solution for detecting deception with errors. In Section V, we present our solution for detecting deception with disguised actions and errors. In Section VI, we present our solution for detecting deception with don't cares

 ISBN: 978-1-61208-783-2

actions. In Section VII, we conclude.

II. Literature Review

Spamming is the use of automated scripts to send unsolicited content to large members of recipients for commercial advertising purposes or fraudulent purposes like phishing emails. Webspam refers to a host of techniques to manipulate the ranking of web search engines and cause them to rank search results higher than the others [2]. Examples of such techniques include *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 [3] [4]. There are several works for preventing the use of content-based or link-based techniques by web spambots [5]–[10]. However, these works focus on identifying the content or links added by spambots, rather than detecting the spambot based on their actions. For example, Ghiam et al. in [4] 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 [3] proposed a method to detect web spam by using either content-based, link-based techniques or a combination of both. Gyongyi et al. in [11] 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 [12] introduced the concept of spam mass and proposed a method for identifying pages that benefit from link spamming. Egele et al. [13] 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. [14] presented a statistical approach to detect spam profiles on Online Social Networks (OSNs). The work in [14] presented a generic statistical approach to identify spam profiles on OSNs. For that, they identified 14 generic statistical features that are common to both Facebook and Twitter, then they used three classification algorithms (naive Bayes, Jrip and J48) to evaluate those features on both individual and combined data sets crawled from Facebook and Twitter networks. Prieto et al. [15] 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. They 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. There are also techniques that analyze *spambot behaviour* [9] [16]. These techniques utilize Supervised Machine Learning (SML) to identify the source of the spambot, rather than detecting the spambot. In this regard, Dai et al. [17] used SML 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. Then, 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 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 which represents a sequence of string characters corresponding to the user's online actions on Twitter. The authors in [22] adapt biological Deoxyribonucleic Acid (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. Thereafter, 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.

More relevant to our work are string pattern matching-based techniques that detect spambots based on their actions

 ISBN: 978-1-61208-783-2

(i.e., based on how they interact with the website these spambots attack) [25] [26]. These techniques model the user's 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 [25] proposes a rule-based, on-the-fly web spambot classifier technique, which identifies web spambots by performing exact string pattern matching using tries. They introduce the idea of web usage behavior to inspect spambots behavior on the web. For that, they introduce an action string concept, which is a representation of the series of web usage actions in terms of index keys to model user behavior on the web. The main assumptions of Hayati et.al proposed method are: web sites spambots mainly to spread spam content rather than consume the content, and the spambot sessions last for some seconds unlike the human sessions that last normally for a couple of minutes. The trie data structure is built based on the action strings for both human and spambots where each node contains the probability of a specific action being either human or spambot. Each incoming string through the trie is validated, and the result falls into two categories: match and not match. Hayati et.al used Matthews Correlation Coefficient (MCC) of binary classification to measure the performance of their proposed framework. The work of [26] improves upon [25] 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 Fast computation using Prefix Table *FPT* algorithm [27] under Hamming distance model to identify spambot action strings in the presence of decoy actions. However, both [25] and [26] are limited in that they consider consecutive spambot actions. This makes them inapplicable in real settings where a spambot begins to take on sophisticated forms of creative deception and 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 to deceive detection techniques. Also, both [25] and [26] do not address the issue of time (the spambot can pause and speed up) or errors (deceptive or unimportant actions). The algorithms presented here are not comparable to [25] and [26] as they address different issues.

III. Background and problems definitions

In this section, we introduce the main concepts and definitions of the key terms used throughout this paper. Then, we state the main problems that the paper will address.

A. Background

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 i^{th} 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 $T[i, i + |p|] \neq T[j, j + |p|]$ [28], [29]. For convenience, we will assume each T ends in a special character \$, where \$ does not appear in other positions and it is less than a for all $a \in \Sigma$.

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 [30] [31]. 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 the 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 [32]–[34]. The major advantage of *suffix arrays* over *suffix trees* is space. The space needed using suffix trees increases with alphabet size. Thus, suffix arrays are more useful in computing the frequency and location of a substring in a long sequence, when the alphabet is large [33]. $LCP(T_1, T_2)$ 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 $i \in [1..n]$ [29] [32].

The *suffix tree T* for a string *S* of length n over the alphabet Σ is a rooted directed compacted trie built on the set of suffixes of S. The suffix tree has n leaves and its internal nodes have at least two children, while its edges are labelled with substrings of *S*. The labels of all outgoing edges from a given node start with a different character. All leaves of the suffix tree are labelled with an integer i, where $i \in \{1 \dots n\}$ and the concatenation of the labels on the edges from the root to the leaf gives us the suffix of *S* which starts at position i. The nodes of the (non-compacted) trie, which have branching nodes and leaves of the tree are called *explicit nodes*, while the others are called *implicit nodes*. The occurrence of a substring P in S is represented on T by either an explicit node or implicit node and called the *locus* of P. The *suffix tree T* can be constructed in $O(n)$ time and space. In order to have one-to-one correspondence between the suffixes of *S* and the leaves of *T*, a character \$ $\notin \Sigma$ is added to the end of the label edge for each suffix i to ensure that no suffix is a prefix of another suffix. To each node α in T is also associated an interval of leaves $[i..j]$, where $[i..j]$ is the set of labels of the leaves that have α as an ancestor (or the interval $[i..i]$ if α is a leaf labelled by i). The intervals associated with the children of α (if α is an internal node) form a partition of the interval associated with α (the intervals are disjoints sub-intervals of $[i..j]$ and their union equals $[i..j]$). For any internal node α in the *suffix tree T*, the concatenation of all edge labels in the path from the root to the node α is denoted by $\bar{\alpha}$ and the string depth of a node α is denoted by $|\bar{\alpha}|$ [28].

Definition 1. A TEMPORALLY ANNOTATED ACTION SEQUENCE is a sequence $T = (a_0, t_0), (a_1, t_1)...(a_n, t_n)$, where $a_i \in A$, with A set of actions, and t_i represents the time that action a_i took place. Note that $t_i < t_{i+1}$, $\forall\, i \in [0, n]$ see (Figure 1).

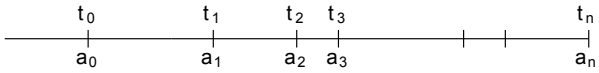

Figure 1. Temporally annotated action sequence T.

Definition 2. AN ACTION SEQUENCE is a sequence: $s_1...s_m$, where $s_i \in A$, with A is the set of all possible actions.

Definition 3. A DICTIONARY $\hat{S}$ is a collection of tuples $\langle S_i, W_i \rangle$, where S_i is a temporally annotated sequence corresponding to a spambot and W_i is a time window (total estimated time for all set of actions performed by the spambot).

Definition 4. 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 [35] [36].

Definition 5. 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 [37].

B. Problems definitions

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 actions with malicious and unnecessary actions. In response, we define our problems in the following sections and give a summary on our algorithms which we use to detect spambots utilizing the aforementioned techniques. Our algorithms take into account the temporal information, because it considers time annotated sequences and requires a match to occur within a time window.

Problem 1: DECEPTION WITH ERRORS*:* Given a temporally annotated action sequence T (a_j, t_j), a dictionary $\hat{S}$ containing sequences S_i each associated with a time window W_i, a minimum frequency threshold f, and a maximum Hamming distance threshold k, find all occurrences of each $S_i \in \hat{S}$ in T, such that each 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.

Problem 2: DECEPTION WITH DISGUISED ACTIONS AND ERRORS*:* 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.

Problem 3: DECEPTION WITH DON'T CARES ACTIONS*:* Given a temporally annotated action sequence T (a_j, t_j), a dictionary $\hat{S}$ containing sequences S_i over the alphabet $\Sigma \cup \{*\}$, each associated with a time window W_i, a minimum frequency threshold f, and a maximum Hamming distance threshold k, find all occurrences of each $S_i \in \hat{S}$ in T, such that each 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.

In the following sections, we present a summary on our algorithms which we use to tackle the aforementioned problems. It is worth noting that the algorithms require a preprocessing stage before including the main algorithm. This includes input sequences temporally annotated actions T where these temporally annotated sequences are produced from the user's 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. Then, the algorithm extracts the actions of the temporally annotated action sequence T into a sequence T_a that contains only the actions $a_0 \dots a_n$ from T.

IV. DECEPTION WITH ERRORS

Current spambot countermeasure tools are mainly based on the analysis of the spambot content features which can help in distinguishing the legitimate account from the fake account. Furthermore, there are tools that work on the network level by tracking the excessive number of different IP addresses over a network path for a period of time that might indicate the presence of a spambot network. However, these techniques do not effectively identify the behavioural change of the spambot where most of them did not come up with satisfactory results. At this regard, the spammer is constantly trying to change the spambot actions behaviour to make it appear like human actions, either by replacing specific actions by others or changing the order of actions which ultimately leads to the primary goal of creating the spambot to evade the detection. This type of manipulation falls under the umbrella of creative deception which intentionally causes errors in order to bypass the existing detection tools. For example, suppose a spambot designed to promote the selling of products to the largest number of websites as a sequence of actions: *{User Registration, View Home Page, Start New Topic, Post a Comment on Products, View Topics, Reply to Posted Topic "with Buy Link"}* can be redesigned by replacing a few actions with others such that it does not affect the goal of the spambot as following: *{User Registration, View Home Page, Update a Topic, Post a Comment on Products, Preview Topic, Reply to Posted Comment "with Buy Link"}*. To solve this type of problems, the solution should take into account the occurrence of spambot actions with mismatches. For that, our contribution to solve PROBLEM 1 supposes that the spambot performs the same sequence of malicious actions multiple times. Thus, we require a sequence to appear at least f times to attribute it as a spambot. In addition, we take into account the fact that spambots perform their actions within a time window. We consider mismatches, and we assume that the spambots

dictionary and parameters are specified on domain knowledge (e.g. from external sources or past experience).

Uncover Deception with errors

Our algorithm for solving (PROBLEM 1) needs to perform pattern matching with k errors where each sequence S_i in $\hat{S}$ should occur in T at least f times within its associated time window W_i. For that, we employ an advanced data structure *Generalized Enhanced Suffix Array* (GESA) with the help of *Kangaroo* method [38]. First, our algorithm constructs GESA for a collection of texts to compute the *Longest Common Sequence* LCS between the sequence of actions T_a and the dictionary $\hat{S}$ in linear time and space. The algorithm concatenates sequences (T_a and spambots dictionary $\hat{S}$) separated by a special delimiter at the end of each sequence to build our GESA, using *almost pure Induced-Sorting* suffix array [39] as follows:

$$GESA(T_a, \hat{S_{S_i}}) = T_a\$_0 S_1\$_1 S_2\$_2 \dots S_r\$_r$$

such that, $S_1 \dots S_r$ are sets of spambot sequences that belong to dictionary $\hat{S_{S_i}}$, and $\$_0, \dots, \$_r$ are special symbols not in Σ and smaller than any letter in T_a with respect to the alphabetical order. Then, the algorithm constructs GESA^R, a table which retains all the lexicographical ranks of the suffixes of GESA. Our algorithm uses the collection of tables (GESA, GESA^R, LCS, T, $\hat{S}$) to detect each sequence $S_i = s_1 \dots s_m$ in $T = (a_0, t_0), (a_1, t_1) \dots (a_n, t_n)$ that occurs with a maximum number of mismatches k in the spambot time window W_i. To do that, for each spambot sequence S_i in the spambot dictionary $\hat{S}$, the algorithm calculates the *longest common sequence* LCS between S_i and T_a starting at position 0 in sequence S_i and position j in sequence T_a such that the common substring starting at these positions is maximal as follows:

$$LCS(S_i, T_a) = max(LCP(GESA(i_1, i_2), GESA(j_1, j_2)) = l_0,$$

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 occurs (or one of the sequences terminates). Next, we find the length of the longest common subsequence starting at the previous mismatch position l_0 which can be achieved using the *Kangaroo* method as follows:

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

The algorithm will continue to use the *Kangaroo* method to find the other number of mismatches, until the number of errors is greater than k or one of the sequences terminates. Finally, in each occurrence of S_i in T_a, the algorithm will check its time window W_i using the dictionary $\hat{S}$ 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 is $|S_i|$, and compares it to its time window W_i. If the resultant time is less than or equal to W_i, the algorithm considers that the pattern sequence corresponds to a spambot.

V. DECEPTION WITH DISGUISED ACTIONS AND ERRORS

Spammers might attempt to deceive detection tools by creating more sophisticated sequences of actions in a creative way as their attempt to disguise their actions is by varying certain actions and making some errors. For example, a spambot takes the actions $AFCDBE$, then $AGCDBE$, then $AGDDBE$ etc. This can be described as $A[FG][CD]DBE$. They try to deceive by changing the second and third action. The action $[FG]$ and $[CD]$ are variations of the same sequence. We will call the symbols A, D, B, E solid, the symbols $[FG]$ and $[CD]$ indeterminate or non-solid and the string $A[FG][CD]DBE$ degenerate string which is denoted by $\tilde{S}$. At this case, we are not concerned which actions will be disguised, but we assume that the number of attempts to disguise is limited by a constant c and the number of errors is bounded by k.

Uncover Deception with Disguised Actions and Errors

Our algorithm for solving (PROBLEM 2) uses the following three steps which make the pattern matching with disguised actions fast and efficient:

Step 1: For each *non-solid* s_j occurring in a 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 the substitution process and will be considered as a *solid* pattern, see (Table I).

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

$\tilde{P}$	A	[FG]	[CD]	D	B	E
$\hat{P}$	A	$\#_1$	$\#_2$	D	B	E

Step 2: This step is similar to the algorithm being used in PROBLEM 1, which constructs GESA to concatenate 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 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 the alphabetical order. The algorithm works similarly to the algorithm described in the previous section with the help of the *Kangaroo* method in addition to *hashMatchTable* (Table II) to do *bit masking*. At any time, the algorithm encounters '#' at the matching pattern, it will get into the *Verification process*.

Step 3 *(Verification process):* At this step, the algorithm considers each '#' as an allowed mismatch and does *bit masking* operation using *hashMatchTable*, to find whether the current comparing action a_i in T_a has a match with one of the actions in '$\#_l$'. The columns of *hashMatchTable* are indexed by the (*ascii code*) of each alphabets in Σ by either using the capital letters or small letters to make the pattern matching testing fast and efficient.

 ISBN: 978-1-61208-783-2

TABLE II. HASHMATCHTABLE OF THE PATTERN $\tilde{P_1} = A[FG][CD]DBE$ WHERE ITS CONVERSION IS $\hat{P_1} = A\#_1\#_2DBE$

asscii(a_i)	65	66	67	68	69	70	71	…	88	89	90
a_i	A	B	C	D	E	F	G	…	X	Y	Z
$\hat{P_1}\#_1$	0	0	0	0	0	1	1	0	…	0	0
$\hat{P_1}\#_2$	0	0	1	1	0	0	0	0	…	0	0
…	…	…	…	…	…	…	…	…	…	…	…
$\hat{P_r}\#_l$	…	…	…	…	…	…	…	…	…	…	…

VI. DECEPTION WITH DON'T CARES ACTIONS

This type of creative deception can be used when we do not care about the type of some actions, which appear between important actions in a sequence of actions. This is important when we want to examine a sequence of actions where some of the actions should be included in the same order to be carried out regardless of the type of other actions in between. For example, travel booking websites which are specialized in selling products and services such as booking flights, hotels, and rental cars through the company itself or through an online travel agency are the most vulnerable businesses to spambots. More accurately, spambots are commonly used at those traveling portals to transfer customers from a specific traveling portal to a competitive one in seconds. This can be done by using an automated script called *price scraping*, which helps steal real-time pricing data automatically from a targeted traveling website to the competitor one. This can help in the decision making of the competitor's products prices, which will be adjusted to a lower price to attract more customers. The competitors also can use *web scrapping* which helps steal the content of the entire traveling website or some parts, with the intent to have the same offers with some modifications that will give them a competitive advantage. For that, we should employ the AI approach using clever algorithms, to detect the most threatened actions in a sequence of actions. For example, suppose a bot script designed to steal some parts of the targeted website (such as pages, posts, etc.), and make a click fraud on selected advertisements as follows: (*Login website:A, Web admin section:B, Select pages:C, Export:D, Save to file:E, Get element:F, Get "ad1":G, Click():H*). Note that, we give an index key for each action, for it to be easy to create a sequence of actions, like that: $ABCDEFGH$. As we see, there are some actions that can be replaced with others such as action (Select pages:C) which can be any other select (posts, images, etc.), and the same thing for the actions (EFG). Those type of actions are the ones we "don't care" about, and thus, we can formulate the bot as follows: $AB*D***H$.

Uncover Deception with Don't Cares Actions

Our algorithm for solving (PROBLEM 3) uses a fast and efficient algorithm which can locate all sequences of actions P with "don't cares" of length m in text of actions S of length n in linear time, using *suffix tree* of S and *Kangaroo* method. Suppose we have this sequence ($P = AB*D***H$), and we want to locate all occurrences of pattern P in log of actions S. Our algorithm will solve it using the following steps:

- Build the *suffix tree* of S
- Divide P into sub-patterns P_k: $P_1P_2P_3 = (AB*)(D**)(H)$
- Using the *suffix tree* of S and the *Kangaroo* method which can be applied to selected suffixes of the suffix tree of S by the use of a predefined computational method to answer subsequent queries in $O(k)$ time. This will find all occurrences of pattern P with "don't cares" in text S such that each query tests the explicit actions of each sub-pattern without caring of "don't cares" actions which are represented by '*'.

VII. CONCLUSION AND FUTURE WORK

We have presented our proposed algorithms that can detect a series of malicious actions which occur with variable time delays, repeated multiple times, and interleave legitimate and malicious actions in a creative way. Our proposed solutions tackled different types of creative deception that spammers might use to defeat existing spam detection techniques such as using errors, disguised, and "don't cares" actions. Our algorithms took into account the temporal information because they considered time annotated sequences and required a match to occur within a time window. The algorithms solved the problems exactly in linear time and space, and they employed advanced data structures to deal with problems efficiently. We are planning to extend this work by designing new methods for different variations of uncertain sequences, e.g., introducing probabilities (calculating whether it is false positive or false negative), statistics (frequency of symbols), and weights (penalty matrices for the errors).

REFERENCES

[1] E. Roberts, "Bad bot report 2020: Bad bots strike back," 2020. [Online]. Available: https://www.imperva.com/blog/bad-bot-report-2020-bad-bots-strike-back/

[2] M. Najork, "Web spam detection." Encyclopedia of Database Systems, vol. 1, 2009, pp. 3520–3523.

[3] 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.

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

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

[6] A. Zinman and J. S. Donath, "Is britney spears spam?" in CEAS 2007 - The Fourth Conference on Email and Anti-Spam, 2-3 August 2007, Mountain View, California, USA, 2007. [Online]. Available: http://www.ceas.cc/2007/accepted.html#Paper-82

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

 ISBN: 978-1-61208-783-2

[14] 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.

[15] 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.

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

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

[26] 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.

[27] C. Barton, C. S. Iliopoulos, S. P. Pissis, and W. F. Smyth, "Fast and simple computations using prefix tables under hamming and edit distance," in International Workshop on Combinatorial Algorithms. Springer, 2014, pp. 49–61.

[28] H. Alamro, G. Badkobeh, D. Belazzougui, C. S. Iliopoulos, and S. J. Puglisi, "Computing the Antiperiod(s) of a String," in CPM, pp. 32:1–32:11.

[29] T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park, "Linear-time longest-common-prefix computation in suffix arrays and its applications," in CPM, 2001, pp. 181–192.

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

[31] 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.

[32] S. J. Puglisi, W. F. Smyth, and A. H. Turpin, "A taxonomy of suffix array construction algorithms," ACM Comput. Surv., vol. 39, no. 2, Jul. 2007.

[33] 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.

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

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

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

[37] 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.

[38] N. Ziviani and R. Baeza-Yates, String Processing and Information Retrieval: 14th International Symposium, SPIRE 2007 Santiago, Chile, October 29-31, 2007 Proceedings. Springer, 2007, vol. 4726.

[39] 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