IARIA

KING'S College LONDON

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

Authors:

Hayam Alamro & Costas S. Iliopoulos

Presenter: (Hayam Alamro)

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

Short resume of the presenter :

- Hayam Alamro is a Ph.D. student in Computer Science (Algorithms & Data analysis Research Group) in the Department of Informatics at King's College London.
- Hayam's research focuses on the analysis and advanced design of string algorithms, approximate pattern matching, Cybersecurity, and data privacy.
- Hayam received her M.Sc. and her B.Sc. (with second class Honour) in Computer Science and Information Systems from King Saud University, Riyadh, Kingdom of Saudi Arabia.
- Before starting her Ph.D. in the UK, Hayam was working as a Lecturer in Computer Science and Information Systems College in Princess Nora University, Riyadh, Kingdom of Saudi Arabia. Hayam also has an experience working as a Computer Assistance in the Ministry of Planning, Interest of Public Statistics, Riyadh, Kingdom of Saudi Arabia.

Outline

- Introduction
- Our contributions
- Background
- Disguised Actions Definition
- Problems Definition
- Disguised Actions with K Mismatches
- Illustration by Example
- Conclusion

Introduction

https://images.app.goo.gl/a5Yreu3X7MSCHmvU7

- A *spambot* is a computer program designed to do repetitive actions on websites, servers or social media communities.

Activities

- Carrying out certain attacks on websites/ servers.
- Involving irrelevant links to increase a website ranking in search engine results.
- Using web crawlers for planting unsolicited material.
- Collecting email addresses from different sources (phishing emails).

Introduction

https://images.app.goo.gl/a5Yreu3X7MSCHmvU7

Existing spam detection techniques

- ***Content-based*** *:* Inject repetitive keywords in meta tags to promote a website in search engines.
- ***Link-based*** *:* Add links to a web page to increase its ranking score in search engines.
- ***Supervised machine learning:*** to identify the source of spambot, rather than detecting the spambot.

Nowdays: The spammers try to manipulate spambots' actions behaviour to appear as it were coming from a legitimate user to bypass the existing spam-filter tools

Introduction


https://images.app.goo.gl/a5Yreu3X7MSCHmvU7

More relevant to our work

- ***String pattern matching-based techniques*** **(Hayati et al. and Ghanaei et al.),**

  But:

- They are inapplicable because 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).

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

Our 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.

➔ Our algorithm can compute all occurrences of indeterminate sequence $\tilde{S}$ in text T in $\boldsymbol{O(m + logn + occ)}$, where:

$\boldsymbol{m}$ is the $|\widetilde{S}|$, $\boldsymbol{n}$ is the $|T|$, and $\boldsymbol{occ}$ is the number of the occurrences of the sequence $\tilde{S}$ in T.

Our Contributions

2. We propose an efficient algorithm for solving $(\boldsymbol{f}, \boldsymbol{c}, \boldsymbol{k}, \boldsymbol{W})$***-Disguised Spambots Detection with indeterminate actions and mismatches***. It is a generalization of the previous problem (1).

➔ Our algorithm takes into account temporal information, because it considers:

- Time-annotated sequences, and
- Requires a match to occur within a time window t.

Background

- Let $T = a_0 \ldots a_{n-1}$ be a string of length $|T| = n$ over an alphabet Σ of size $|\Sigma| = \sigma$
- 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*)
- 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
- *A **degenerate or indeterminate string*** , is defined as a sequence $\tilde{X} = \tilde{x_0}\, \tilde{x_1} .. \widetilde{x_{n-1}}$, where $\tilde{x_i} \subseteq \Sigma$ for all $0 \leq i \leq n-1$
- A ***degenerate symbol*** $\tilde{x}$ over an alphabet Σ is a non-empty subset of Σ

Background

- $|\tilde{x}|$ denotes the size of $\tilde{x}$, and we have $1 \leq \tilde{x} \leq |\Sigma|$.
- If | $\tilde{x_i}$ | = 1, that is | $\tilde{x_i}$ | 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.
- A ***conservative degenerate string*** is a degenerate string where its number of non-solid symbols is upper-bounded by a fixed position constant c.

Example

$$X = ab[ac]a[bcd]bac$$

Is a degenerate string of length 8 over the alphabet Σ = {a, b, c, d}, and conservative degenerate string with c = 2.

Background

- 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...n] in which SA[i] is the ith suffix of T in ascending order.
- $\boldsymbol{LCP(T1, T2)}$ 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 $i \in [1..n]$.

Disguised (Indeterminate) Actions

- Some spambots might attempt to disguise their actions by varying certain actions.

Example

a spambot takes the actions ABCDEF, then ACCDEF, then ABDDEF

can be described as ➔ A[BC][CD]DEF

The action [BC] and [CD] are variations of the same sequence

- A, C, D, E, F ➔ **(solid symbols)**
- [BC] [CD] ➔ **(indeterminate** or **non-solid symbols)**
- A[BC][CD]DEF ➔ **(degenerate string)**

Problems Definitions

A. Given a sequence $T = a_1 \ldots a_n$, and an action sequence $\tilde{S} = s_1 s_2 \ldots s_m$, find all occurrences of $\tilde{S}$ in T where s_i might be solid or indeterminate.

B. Given a temporal annotated sequence $T = (a_{1,} t_1) \ldots (a_n, t_n)$, and an action sequence $\tilde{S} = s_1 s_2 \ldots s_m$, find all occurrences of $\tilde{S}$ in T in time window t, where s_i might be solid or indeterminate with hamming distance between $\tilde{S}$ and T is no more than k mismatches.

Our Main Problem

Since Problem B is a generalization of Problem A, we will focus on Problem B in this presentation.

(f, c, k, W)-***Disguised Spambots Detection with indeterminate actions and mismatches***:

Given a temporal annotated sequence $T = (a_1, t_1) \ldots (a_n, t_n)$, a dictionary $\bar{S}$ containing sequences $\widehat{S_i}$, 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 $\widehat{S_i} \in \bar{S}$ **in** T, such that each $\widehat{S_i}$ occurs:

I. At least f times within its associated time window W_i, and

II. With at most K mismatches according to Hamming distance.

Disguised Actions with *K* Mismatches

Preprocessing Stage

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***.

125.127.33.125 - smith [14/Oct/2019: 10:12:26 –0500] "GET /blog/page-address.htm HTTP/1.1" 200 1043 "-" "Mozilla/5.0 Chrome/80.0.3134.311"

Date – Time Stamp → **Time Point t**

Action Request → **Index Key**

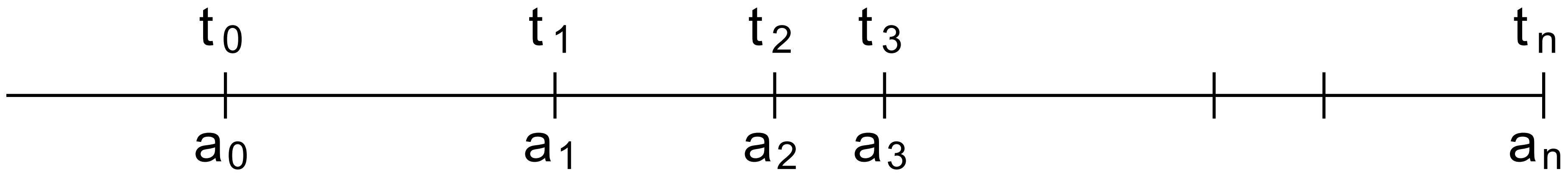
Disguised Actions with *K* Mismatches

Definition

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 $\boldsymbol{A}$ set of actions, and

$\boldsymbol{t_i}$ represents the time that action a_i took place. Note that $\boldsymbol{t_i} < \boldsymbol{t_{i+1}}$, ∀ i ∈ [0, n].



Disguised Actions with K Mismatches

Our Spambot Dictionary Representation

i	S_i	W_i (sec)
1	cbbx	20
2	byadc	25
…	…	…

Disguised Actions with K Mismatches

Algorithm Steps:

Step 1: For each ***non-solid*** $\boldsymbol{s_j}$ occurring in degenerate sequence $\tilde{S} = s_1 s_2 \ldots s_m$, we substitute each s_j with '#' symbol, where '#' is not in Σ. Let $\widehat{\boldsymbol{S}}$ be the resulting pattern after substitution.

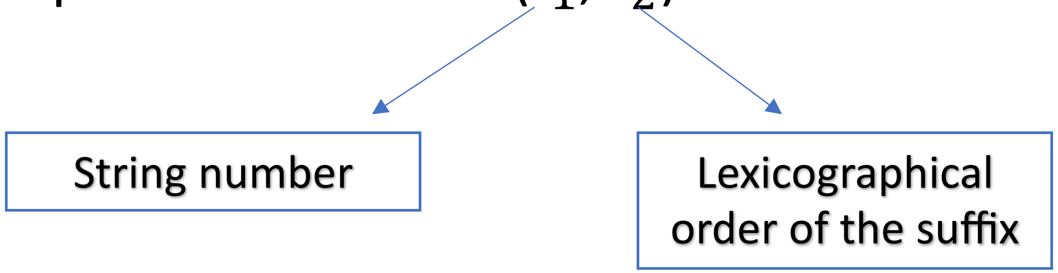
$\tilde{S}$	A	B	[GX]	C	[AD]	F
$\widehat{S}$	A	B	$\#_1$	C	$\#_2$	F

Disguised Actions with K Mismatches

Step 2: Extract the actions of the temporally annotated sequence T into T_a such that it contains only the actions $\boldsymbol{a_1} \ldots \boldsymbol{a_n}$ from T.

Step 3: Build **Generalized Enhanced Suffix Array (GESA):**

it is 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).



Disguised Actions with K Mismatches

Generalized Enhanced Suffix Array for a collection of texts ($\boldsymbol{T_a}$ and $\boldsymbol{\bar{S}_{\widehat{S_i}}}$):

$$GESA(T_a, \bar{S}_{\widehat{S_i}}) \ = \ Ta!_0\,\hat{S}_1!_1\,\hat{S}_2!_2 \ldots \hat{S}_r!_r$$

- $\hat{S}_1$, .., $\hat{S}_r$ are spambots sequences belong to dictionary $\bar{S}$
- $!_0$, .. , $!_r$ are special delimiters not in Σ, and smaller than any alphabetical letter in $\boldsymbol{T_a}$ and smaller than '#'

Disguised Actions with K Mismatches

- Our algorithm will refer to a collection of tables $(GESA, GESA^R, LCS, T, \bar{S}_{\widehat{S_i}})$ ➔ to find disguised spambots within a time window t as follows:
  - Given a temporally annotated action sequence $\boldsymbol{T} = (\boldsymbol{a_0}, \boldsymbol{t_0})(\boldsymbol{a_1}, \boldsymbol{t_1}) \ldots (\boldsymbol{a_{n-1}}, \boldsymbol{t_{n-1}})$, an action sequence $\widehat{\boldsymbol{S}} = \boldsymbol{s_1 s_2} \ldots \boldsymbol{s_m}$, and an integer t, we will compute $\boldsymbol{j_1}, \boldsymbol{j_2}, \ldots, \boldsymbol{j_m}$ such that $\boldsymbol{a_{j_i}} = \boldsymbol{s_i}, \boldsymbol{1} \leq \boldsymbol{i} \leq \boldsymbol{m}$ and $\sum_{i=1}^{m} \boldsymbol{t_{j_i}} < \boldsymbol{t}$ or $\boldsymbol{t_{j_m}} - \boldsymbol{t_{j_1}} < \boldsymbol{t}$ with **Hamming distance between $\boldsymbol{T_a}$** and $\widehat{\boldsymbol{S}}$ no more than $\boldsymbol{k}$ **mismatches.**

Disguised Actions with K Mismatches

- Our algorithm, also includes:
- Initialization for ***hashMatchTable*** to do ***bit masking*** operation
- ***Kangaroo method*** to find the ***Longest Common Sequence LCS*** between a sequence of actions in $\boldsymbol{T}$ and an action sequence $\widehat{\boldsymbol{S}}_{\boldsymbol{i}}$ with at most $\boldsymbol{K}$ mismatches in linear time.

Disguised Actions with K Mismatches

- **Step 4:** For each $\widehat{\boldsymbol{S}}_{\boldsymbol{i}}$ in the spambots dictionary $\overline{\boldsymbol{S}}_{\widehat{\boldsymbol{S}}_{\boldsymbol{i}}}$, the algorithm calculates the *Longest Common Sequence LCS* between $\widehat{\boldsymbol{S}}_{\boldsymbol{i}}$ and $\boldsymbol{T}_{\boldsymbol{a}}$ starting at position 0 in sequence $\hat{S}_i$ and at position j in sequence T_a , such that the common substring starting at these positions is maximal as follows:

➢ Find the the suffix index $\boldsymbol{i}$ of $\widehat{\boldsymbol{S}}_{\boldsymbol{i}}$ in $GESA$ using $\boldsymbol{GESA}^{\boldsymbol{R}}$ table (retains all the lexicographical ranks of the suffixes of $GESA$).

➢ Find the closest suffix $\boldsymbol{j}$ (belongs to a sequence in $\boldsymbol{T}_{\boldsymbol{a}}$) to the suffix $\boldsymbol{i}$ (of $\widehat{\boldsymbol{S}}_{\boldsymbol{i}}$) in $GESA$.

Disguised Actions with K Mismatches

- Compute the ***Longest Common Sequence LCS*** between $GESA(i)$ and $GESA(j)$ as follows:

$$\boldsymbol{LCS}\left(\hat{\boldsymbol{S}}_{\boldsymbol{i}}, \boldsymbol{T}_{\boldsymbol{a}}\right) = \boldsymbol{max}(\boldsymbol{LCP}(GESA(i_1, i_2), GESA(j_1, j_2))) = \boldsymbol{l_0}$$

Where $\boldsymbol{l_0}$ is the maximum length of the *longest common subsequence* matching characters between $GESA(i_1, i_2)$ and $GESA(j_1, j_2)$ until the first mismatch (or one of the sequences terminates).

Disguised Actions with K Mismatches

- Next, find the length of the *longest common subsequence* matching characters after previous mismatch position $\boldsymbol{l_0}$ using ***Kangaroo*** *method* until the second mismatch (or one of the sequences terminates) as follows:

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

Disguised Actions with K Mismatches

- Once our algorithm encounters '#' in the sequence $\widehat{S}_i$, it will get into the ***verification process:***
- Query whether the corresponding action a_i (in T_a) belongs to the set of actions in '#', to do that:
- The algorithm uses a ***bit masking*** operation (***And*** bit wise operation) between the two sets ('#' and a_i) such that (a_i is represented by a bit '1', and each action in '#' is represented by '1' and '0' otherwise using ***hashMatchTable***).

Disguised Actions with K Mismatches

hashMatchTable

- The columns are indexed by the (ascii code) of each character ($\boldsymbol{a_i} \in \Sigma$) (for directly access)
- The rows are indexed by the number of the spambots sequence $\widehat{\boldsymbol{S}}_{\boldsymbol{i}}$ and the number of '$\#_l$'
- The algorithm applies the following formula:

$$1 \wedge hashMatchTable[\widehat{\boldsymbol{S}}_{\boldsymbol{r}}\#_{\boldsymbol{l}}][\mathrm{ascii}[\boldsymbol{a_i}]]$$

Disguised Actions with K Mismatches

hashMatchTable

$$\widetilde{S_1} = AB[GX]C[AD]F \rightarrow \widehat{S}_1 = AB\#_1C\#_2F$$

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

Disguised Actions with K Mismatches

- **Step 5:** Finally, at each occurrence of $\widehat{\boldsymbol{S}}_{\boldsymbol{i}}$ in the sequence $\boldsymbol{T_a}$, our algorithm checks its time window W_i using the dictionary $\overline{\boldsymbol{S}}_{\widehat{\boldsymbol{S}}_i}$ in $\boldsymbol{T}$, such that it sums up each time t_i associated with its action a_i starting at the position j_2 in $GESA(j_1, j_2)$ until the length of the spambot $|\ \widehat{\boldsymbol{S}}_{\boldsymbol{i}}\ |$, and then compare it to its time window W_i. If the resultant time is less than or equal to W_i, the algorithm considers that the sequence $\widehat{\boldsymbol{S}}_{\boldsymbol{i}}$ is spambots and terminates.

➢ The algorithm will continue to find other occurrences of the spambot sequence $\widehat{\boldsymbol{S}}_{\boldsymbol{i}}$ using the adjacent suffixes to the suffix index of $\widehat{\boldsymbol{S}}_{\boldsymbol{i}}$ in GESA.

Illustration by Example

Example

- Suppose : $T_a = ABBABGCDFCBACAFAABGDFF$, $\widehat{S}_1$ = B $\#_1$ C $\#_2$ F
- $\#_1$ = [GX], $\#_2$ = [AD], $K = 2, f = 2$
- Concatenation strings of $Ta!_0\,\widehat{S}_1!_1$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	B	B	A	B	G	C	D	F	C	B	A	C	A	F	A	A	B	G	D	F	F	$!_0$	B	#1	C	#2	F	$!_1$

Illustration by Example

i	$GESA[i]$	Suffix	$GESA^R[i]$
0	(1,28)	$!_1$	5
1	(0,22)	$!_0 b\#_1 c\#_2 f!_0$	13
2	(1,24)	$\#_1 c\#_2 f!_1$	11
3	(1,26)	$\#_2 f!_1$	6
4	(0,15)	$aabgdf\, f!_0 b\#_1 c\#_2 f!_1$	14
5	(0,0)	$abbabgcdfcbacafaabgdf\, f!_0 b\#_1 c\#_2 f!_1$	27
6	(0,3)	$abgcdfcbacafaabgdf\, f!_0 b\#_1 c\#_2 f!_1$	19
7	(0,16)	$abgdf\, f!_0 b\#_1 c\#_2 f!_1$	20
8	(0,11)	$acafaabgdf\, f!_0 b\#_1 c\#_2 f!_1$	25
9	(0,13)	$afaabgdf\, f!_0 b\#_1 c\#_2 f!_1$	18
10	**(1,23)**	$b\#_1 c\#_2 f!_1$	12
11	(0,2)	$babgcdfcbacafaabgdf\, f!_0 b\#_1 c\#_2 f!_1$	8
12	**(0,10)**	$bacafaabgdf\, f!_0 b\#_1 c\#_2 f!_1$	17
13	(0,1)	$bbabgcdfcbacafaabgdf\, f!_0 b\#_1 c\#_2 f!_1$	9
14	**(0,4)**	$bgcdfcbacafaabgdf\, f!_0 b\#_1 c\#_2 f!_1$	24
15	**(0,17)**	$bgdf\, f!_0 b\#_1 c\#_2 f!_1$	4
16	(1,25)	$c\#_2 f!_1$	7
17	(0,12)	$caf\ aabgdf\, f!_0 b\#_1 c\#_2 f!_1$	15
18	(0,9)	$cbacafaabgdf\, f!_0 b\#_1 c\#_2 f!_1$	28
19	(0,6)	$cdfcbacafaabgdf\, f!_0 b\#_1 c\#_2 f!_1$	21
20	(0,7)	$dfcbacafaabgdf\, f!_0 b\#_1 c\#_2 f!_1$	26
21	(0,19)	$df\, f!_0 b\#_1 c\#_2 f!_1$	23
22	(1,27)	$f!_1$	1
23	(0,21)	$f\ !_0 b\#_1 c\#_2 f!_1$	**10**
24	(0,14)	$faabgdf\, f!_0 b\#_1 c\#_2 f!_1$	2
25	(0,8)	$fcbacafaabgdf\, f!_0 b\#_1 c\#_2 f!_1$	16
26	(0,20)	$f\ f!_0 b\#_1 c\#_2 f!_1$	3
27	(0,5)	$gcdfcbacafaabgdf\, f!_0 b\#_1 c\#_2 f!_1$	22
28	(0,18)	$gdf\, f!_0 b\#_1 c\#_2 f!_1$	0

- There are three occurrences for $\widehat{S}_1$ in $\boldsymbol{T}$ at $i = 12, 14$ *and* 15

Experimental Evaluation

- Our experiments showed that our algorithm is efficient and applicable to large action sequences.
- See our paper for details.

Conclusion

- We introduced our algorithm (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.

Conclusion

- 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.

Thank You