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.
Introduction

Our contributions

Background

Disguised Actions Definition

Problems Definition

Disguised Actions with $K$ Mismatches

Illustration by Example

Conclusion
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).
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.

**Nowadays**: The spammers try to manipulate spambots’ actions behaviour to appear as if it were coming from a legitimate user to bypass the existing spam-filter tools.
Introduction

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).

---

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.

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

- $m$ is the $|\tilde{S}|$, $n$ is the $|T|$, and $\text{occ}$ is the number of the occurrences of the sequence $\tilde{S}$ in $T$. 
2. We propose an efficient algorithm for solving \((f, c, k, W)\)-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 \(t\).
Let $T = a_0 \ldots a_{n-1}$ be a string of length $|T| = n$ over an alphabet $\Sigma$ of size $|\Sigma| = \sigma$

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)

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 $\tilde{X} = \tilde{x}_0 \tilde{x}_1 \ldots \tilde{x}_{n-1}$, where $\tilde{x}_i \subseteq \Sigma$ for all $0 \leq i \leq n - 1$

A degenerate symbol $\tilde{x}$ over an alphabet $\Sigma$ is a non-empty subset of $\Sigma$
Background

- $|\bar{x}|$ denotes the size of $\bar{x}$, and we have $1 \leq |\bar{x}| \leq |\Sigma|$.

- If $|\bar{x}_i| = 1$, that is $|\bar{x}_i|$ repeats a single symbol of $\Sigma$, we say that $\bar{x}_i$ is a solid symbol and $i$ is a solid position. Otherwise, $\bar{x}_i$ 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 $\Sigma = \{a, b, c, d\}$, and 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 $SA[1...n]$ in which $SA[i]$ is the ith suffix of $T$ in ascending order.

$LCP(T_1,T_2)$ 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 $i \in [1..n]$. 
Some spambots might attempt to disguise their actions by varying certain actions.

Example

a spambot takes the actions ABCDEF, then ACCDEF, then ABDEDF
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)
A. Given a sequence $T = a_1 \ldots a_n$, and an action sequence $\tilde{S} = s_1s_2 \ldots s_m$, 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 $\tilde{S} = s_1s_2 \ldots s_m$, 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 \(\overline{S}\) containing sequences \(\overline{S}_i\), 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 \(\overline{S}_i \in \overline{S}\) in \(T\), such that each \(\overline{S}_i\) 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*.

```
“-” “Mozilla/5.0 Chrome/80.0.3134.311”
```

- **Date – Time Stamp**
- **Action Request**
- **Time Point t**
- **Index Key**
A Temporally Annotated Action Sequence: is a sequence

\[ T = (a_0, t_0), (a_1, t_1), \ldots (a_n, t_n), \] where \( a_i \in A \), with \( A \) set of actions, and \( t_i \) represents the time that action \( a_i \) took place. Note that \( t_i < t_{i+1} \), \( \forall i \in [0, n] \).
Our Spambot Dictionary Representation

<table>
<thead>
<tr>
<th>$i$</th>
<th>$S_i$</th>
<th>$W_i$ (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cbbx</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>byadc</td>
<td>25</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Algorithm Steps:

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

<table>
<thead>
<tr>
<th>( \tilde{S} )</th>
<th>A</th>
<th>B</th>
<th>[GX]</th>
<th>C</th>
<th>[AD]</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \hat{S} )</td>
<td>A</td>
<td>B</td>
<td>#₁</td>
<td>C</td>
<td>#₂</td>
<td>F</td>
</tr>
</tbody>
</table>
Step 2: Extract the actions of the temporally annotated sequence $T$ into $T_a$ such that it contains only the actions $a_1 \ldots a_n$ 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 $(i_1, i_2)$.

- **String number**
- **Lexicographical order of the suffix**
Disguised Actions with $K$ Mismatches

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

$$GESA(T_a, \bar{S}_{S_i}) = T a!_0 \hat{S}_1!_1 \hat{S}_2!_2 ... \hat{S}_r!_r$$

- $\hat{S}_1, .., \hat{S}_r$ are spambots sequences belong to dictionary $\bar{S}$
- $!_0, .., !_r$ are special delimiters not in $\Sigma$, and smaller than any alphabetical letter in $T_a$ and smaller than ‘#’
Disguised Actions with $K$ Mismatches

- Our algorithm will refer to a collection of tables ($GESA, GESA^R, LCS, T, \tilde{S} \tilde{s}_i$) to find disguised spambots within a time window $t$ as follows:
  - Given a temporally annotated action sequence $T = (a_0, t_0)(a_1, t_1) \ldots (a_{n-1}, t_{n-1})$, an action sequence $\tilde{S} = s_1s_2 \ldots s_m$, 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 $\tilde{S}$ no more than $k$ 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 $T$ and an action sequence $\hat{S}_i$ with at most $K$ mismatches in linear time.
Step 4: For each $\hat{S}_i$ 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 at position $j$ in sequence $T_{a}$, such that the common substring starting at these positions is maximal as follows:

- Find the suffix index $i$ of $\hat{S}_i$ in $GESA$ using $GESA^R$ table (retains all the lexicographical ranks of the suffixes of $GESA$).
- Find the closest suffix $j$ (belongs to a sequence in $T_{a}$) to the suffix $i$ (of $\hat{S}_i$) in $GESA$. 
Compute the **Longest Common Sequence LCS** between \( GESA(i) \) and \( GESA(j) \) 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 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).
Next, find the length of the longest common subsequence matching characters after previous mismatch position \( l_0 \) using Kangaroo method until the second mismatch (or one of the sequences terminates) as follows:

\[
\text{max}(LCP(GESA(i_1, i_2 + l_0 + 1), GESA(j_1, j_2 + l_0 + 1))) = l_1
\]
Disguised Actions with $K$ Mismatches

- Once our algorithm encounters ‘#’ in the sequence $\hat{S}_i$, 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

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

$$1 \land hashMatchTable[\widehat{S}_r \# _l][\text{ascii}[a_i]]$$
Disguised Actions with $K$ Mismatches

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

<table>
<thead>
<tr>
<th>Ascii($a_i$)</th>
<th>65</th>
<th>66</th>
<th>67</th>
<th>68</th>
<th>...</th>
<th>71</th>
<th>...</th>
<th>88</th>
<th>89</th>
<th>90</th>
</tr>
</thead>
<tbody>
<tr>
<td>65 A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>1</td>
<td>...</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>66 B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>1</td>
<td>...</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>67 C</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>68 D</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$\widehat{S}_1#_1$</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$\widehat{S}_1#_2$</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$\widehat{S}_r#_l$</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Disguised Actions with K Mismatches

- **Step 5:** Finally, at each occurrence of $\hat{S}_i$ in the sequence $T_a$, our algorithm checks its time window $W_i$ using the dictionary $\overline{S}_{\hat{S}_i}$ in $T$, 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 $|\hat{S}_i|$, 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 $\hat{S}_i$ is spambots and terminates.

- The algorithm will continue to find other occurrences of the spambot sequence $\hat{S}_i$ using the adjacent suffixes to the suffix index of $\hat{S}_i$ in GESA.
Suppose: $T_a = A B B A B G C D F C B A C A F A A B G D F F$, $\tilde{S}_1 = B \#_1 C \#_2 F$

$\#_1 = [G X], \#_2 = [A D], K = 2, f = 2$

Concatenation strings of $T_a!_0 \tilde{S}_1!_1$
There are three occurrences for $\hat{S}_1$ in $T$ 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.
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.
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