REVEALING NON-ALPHABETICAL GUISES OF SPAM-TRIGGER VOCABLES

RECONOCIMIENTO DE VARIANTES ENMASCARADAS DE VOCABLOS DESENCADENADORES DE CORREO INDESEADO

SERGIO A. ROJAS-GALEANO

PhD., Assistant Professor, Universidad Distrital, Colombia. srojas@udistrital.edu.co

Received for review August 23 th, 2012, accepted June 14th, 2013, final version July, 2 th, 2013

ABSTRACT: Unsolicited bulk email (spam) nowadays accounts for nearly 75% of daily email traffic, a figure that speaks strongly for the need of finding better protection mechanisms against its dissemination. A clever trick recently exploited by email spammers in order to circumvent textual-based filters, involves obfuscation of black-listed words with visually equivalent text substitutions from non-alphabetic symbols, in such a way it still conveys the semantics of the original word to the human eye (e.g. masking viagra as v1@gr@ or as v-i-a-g-r-a). In this paper we discuss how a simple-yet-effective adaptation of a classical algorithm for string matching may meet this stylish challenge to effectively reveal the similarity between genuine spam-trigger terms with their disguised alpha-numeric variants.

KEYWORDS: Uncovering of spam vocables, approximate string matching algorithm.

RESUMEN: El 75% del correo electrónico que se transmite hoy en día, corresponde a mensajes masivos no solicitados (comúnmente denominados spam), lo que evidencia la necesidad de continuar fortaleciendo los mecanismos de protección contra su propagación. Uno de los tretas más ingeniosas utilizadas últimamente por los spammers para sobrepasar los filtros basados en comparación de texto, es el enmascaramiento de las palabras vedadas mediante substituciones con símbolos no alfabéticos, de manera que aún visualmente logren transmitir la semántica del término original (por ejemplo, enmascarando viagra como v1@gr@ o como v-i-a-g-r-a). En este artículo se discute una técnica simple pero efectiva para contrarrestar esta sutil trampa, que consiste en la adaptación de un algoritmo reconocido de apareamiento de textos para revelar efectivamente la similitud existente entre vocablos desencadenadores de filtros spam y variantes alfanuméricas enmascaradas.

PALABRAS CLAVE: Desenmascaramiento de vocablos spam, algoritmo de apareamiento aproximado de texto.

1. INTRODUCTION

Electronic mail (email) is nowadays one of the most popular communication mediums utilized to exchange information at a corporate or personal level. Indiscriminate abuse of the system has led to what is commonly known as spam: unsolicited bulk email intended to broadcast commercial advertisement free of charge or to mislead people to visit illegal or suspicious Web pages. This improper practice is a real burden to the operating cost of the system; according to Barracuda Central (www.barracudacentral.org, last visited: 21-June-2013), a privately held company specialised in email traffic monitoring, almost 75% of an average 600M emails sent daily is spam. The topics of spam messages range from pharmaceutics to lotteries to replicas to illegal advertisement to pornography, just to name a few. These numbers reveal the magnitude of the problem, and although filters are becoming stronger in stopping spam [1-5], spammer's efforts have likewise grown in sophistication.

One of the latest tactics involves masking filter-triggering words by substituting letters with visually equivalent non-alphabetic symbols that are still easily recognized by the naked eye. Some examples of this type of masking are shown in Table 1. Such type of encoding is reminiscent of first-order (i.e. one symbol) internet slang ciphering largely popular in forums, instant and text messaging, and hacker's newsboards (e.g. leet speak [6], chatspeak [7, 8], text-talk [9]). Table 2 shows a non-exhaustive list of substitution ASCII-code symbols for the English alphabet.

Several approaches have been proposed to tackle this masking manoeuvre; we discuss some of them below. Most of the many (combinatorial) non-alphabetical guises for a given spam term would be so infrequent that they would hardly appear in any filter's black-list. Furthermore, maintaining a large black-list covering all the variants would be unfeasible. An obvious choice to account for the transliterations induced by Table 2 would be to perform an inverse mapping of each symbol in the

disguised word. Unfortunately, this is not a one-to-one mapping, making it difficult to find the set of rules that define the exact reversing. A more sophisticated option is to train a probabilistic generative model (for example, a Hidden Markov Model) of the genuine spam term and use it for recognising any of its variants (see [10, 11]). This technique chooses the hidden sequence of states -in this case insertion, substitution, deletion- that most probably explains the observed sequence of symbols of the spam term variation. The main disadvantage of these methods is the computational time required to train and use the models, despite recent optimisations alleviating this issue [12].

Table 1. Examples of first-order substitution masking of the spam term "viagra". Substitution site is underlined in the header row. (Viagra® is a trademark of Pfizer).

viagra	viagra	viagra	viagra	viagra	viagra
uiagra	v1agra	vi@gra	via9ra	viag®a	viagrá
úiagra	vlagra	viägra	viaqra	viagΩa	viagræ

Table 2. A non-exhaustive substitution lists of extended-ASCII symbols for the 26 letters of the English alphabet.

a	b	c	d	e	f	g	h	i	j	k	l	m
@ á à ä â å æ α ª A	6 β ϕ B	© ç < Ç C	δ ϕ D	é è ë ê æ ϵ E	£ ‡ Γ F	9 ç q G	λ H	í ì ï î : ! 1 ; I	¡ J	x K	1 ! / £ L	μ M

n	o	p	q	r	s	t	u	v	w	x	y	z
ñ ∩ N	0 º ó ò ö ô Ø Θ O	? P	g Q	γ π ® 7 R	$ § 5 S	† + ⌈ τ T	v μ ú ù ü û U	u √ V	ω u v W	× % X	ψ μ Y	≥ 2 Z

In this paper we take a different viewpoint and regard the masking trick as a word-to-word matching application, that is, the aim is to determine if the disguised and the genuine spam-trigger vocables coincide. Popular algorithms to compute string-to-string similarity are well-known [13-15]. These algorithms were the basis of subsequent applications in the field of molecular biology and bioinformatics, where they were adapted to align sequences of genomic or proteomic molecules from different organisms in order to compare their similarity [16, 17]. Motivated by the sequence alignment method, in the following we describe an application of the string-to-string matching algorithm for the purposes of detecting cases of non-alphabetically disguised spam terms.

Notation. Lower-case boldface is used to denote character strings (e.g. **a**). Plain font symbols designate characters or scalar values (e.g. $i, n \in \Re$). Capital letters denote matrices or sets (e.g. *D, A*). Single subscripts indicate loci in a string and double subscripts indicate entries of a matrix. The null character is marked as '−'.

2. METHOD

To begin with we recall the classical string-to-string matching algorithm proposed by Wagner and Fischer [15] (see Algorithm 1), which is a refined version of the Levenshtein [14] algorithm to compute the *edit* distance between two strings **a** and **b** with lengths *n* and *m*, respectively. The edit distance basically scores the edits (corrections) needed to transform one string into another, based on an edit cost function $\delta(\cdot,\cdot)$. Such a function defines the cost of insertion, deletion or substitution of two given symbols from the strings.

Algorithm 1. String-to-string edit distance

Input: Strings $\mathbf{a} = (a_1,\ldots, a_n)$, $\mathbf{b} = (b_1,\ldots, b_m)$,
Similarity function $\delta(\cdot,\cdot)$
Output: $D_{(i+1,j+1)}$,the edit distance between **a** and **b**
1: $D_{(1,1)} = 0$
2: **for** $(i = 1,\ldots,n)$ **do**
$D_{(i+1,1)} = D_{(i,1)} + \delta(a_i, -)$
3: **for** $(j = 1,\ldots,m)$ **do**
$D_{(1,j+1)} = D_{(1,j)} + \delta(-, b_j)$
4: **for** $(i = 1,\ldots,n)$ **do**
for $(j = 1,\ldots,m)$ **do**
$D_{(i+1,j+1)} = \min (D_{(i,j+1)} + \delta(a_i, -),$
$D_{(i+1,j)} + \delta(-, b_j),$
$D_{(i,j)} + \delta(a_i, b_j))$

The algorithm maintains a dissimilarity (distance) matrix $D \in \Re^{(n+1)\times(m+1)}$ where any entry $D_{(i+1,j+1)}$ holds the edit distance between the prefix substring $\mathbf{a}_{[1:i]}$ and the prefix substring $\mathbf{b}_{[1:j]}$; the matrix is progressively filled with a dynamic programming procedure that reuses previously calculated distances between shorter substrings, until it gets to the full extent of the input strings.

The computation of the matrix entries works as follows. To start with, observe that distances to, and from, the

empty string would be stored in the first column and first row of D respectively. Hence the first loop of the algorithm fills up the first column of D with the number of deletions needed to transform the successive prefix substrings of **a** into an empty string. Similarly, the second loop fills up the first row with the number of insertions needed to build up the successive prefix substrings of **b** out of an empty string. The heart of the algorithm is the final double-nested loop that computes intermediate distances $D_{(i+1,j+1)}$ or edits needed to transform the substrings with prefix $\mathbf{a}_{[1:i]}$ into that with prefix $\mathbf{b}_{[1:j]}$. This distance is found as the minimum of three quantities that reuse distances between shorter prefix substrings previously examined, namely the distance between $\mathbf{a}_{[1:i-1]}$ and $\mathbf{b}_{[1:j]}$. plus the cost of deletion of symbol $\mathbf{a}_i$ from **a**, the distance between $\mathbf{a}_{[1:i]}$ and $\mathbf{b}_{[1:j-1]}$ plus the cost of insertion of symbol $\mathbf{b}_j$ into **a**, and lastly, the distance between $\mathbf{a}_{[1:i-1]}$ and $\mathbf{b}_{[1:j-1]}$ plus the cost of substitution of $\mathbf{a}_i$ by $\mathbf{b}_j$ in **a**. The cost of insertion, deletion and substitution is determined by the edit cost function $\delta : A \times A \rightarrow \Re$ mentioned above, A being the set of symbols from an admissible alphabet.

Let us come back now to our problem of interest. In order to reveal the obfuscated spam terms, we need to consider allowing substitutions of letters from the English alphabet with symbols from Table 2, plus the occasional deletion of one symbol or insertion of bogus ornament symbols disrupting the original term. Our idea is therefore to use Algorithm 1 with a carefully designed edit cost (or similarity) function capable of detecting the masking operations. For this purpose, let us first express the set of common bogus segmentation characters as:

$$S_\beta = \{ \cdot, *, \sim, |, -, \_, :, ; \}.$$

Secondly, let S_Δ denote the set of admissible substitution symbols for a given character Δ (including itself); this set is determined by the column of Table 2 indexed by Δ in the head row (e.g. for the letter $\Delta :=$ '*l*', $S_l = \{ 1, l, !, /, £, L \}$). Now, let us define the edit cost function $\delta(\cdot,\cdot)$ as:

$$\delta(a_i, b_j) = \begin{cases} 1 & b_j \text{ is null} & (\textit{deletion}) \\ 1 & a_i \text{ is null and } b_j \notin S_\beta & (\textit{insertion}) \\ 0 & a_i \text{ is null and } b_j \in S_\beta & (\textit{bogus segmentation}) \\ 0 & b_j \in S_{a_i} & (\textit{admissible substitution}) \\ 1 & \text{otherwise} & (\textit{non-admissible substitution}) \end{cases} \quad (1)$$

The rationale of Equation (1) is two-fold. On the one hand, replacement of a normal character with an admissible non-alphabetical symbol, or the insertion of a bogus character separator, should be *ignored*, in other words treated as no-cost edits. On the other hand, insertions as well as deletions or else substitutions with non-admissible characters should be treated as actual corrections that incur a cost of one edit.

We observe that the combination of Algorithm 1 with Equation (1) leads to a plausible method to match a disguised vocable and its original term. As an example of the method's operation, Table 3 shows how a perfect match is obtained for the similarity between the vocables "**viagra**" and "**v.1.@.g.r.@**".

Table 3. Resulting distance matrix D for the input strings **a**:= "**viagra**" and **b**:= "**v.1.@.g.r.@**" (left column and top row respectively) computed with Algorithm 1 coupled with Equation (1). The final dissimilarity score $D_{(7,12)} = 0$ indicates a perfect match.

		v	·	1	·	@	·	g	·	r	·	@
	0	1	1	2	2	3	3	4	4	5	5	6
v	1	0	0	1	1	2	2	3	3	4	4	5
i	2	1	1	0	0	1	1	2	2	3	3	4
a	3	2	2	1	1	0	0	1	1	2	2	3
g	4	3	3	2	2	1	1	0	0	1	1	2
r	5	4	4	3	3	2	2	1	1	0	0	1
a	6	5	5	4	4	3	3	2	2	1	1	0

The method would be capable of uncovering spam-filter triggering terms by scanning the email subject or contents against a list of genuine previously-known black-words, i.e. a vocabulary of typical spam terms written in plain lowercase English (notice that capital letters are included in the admissible substitution sets, so messages in uppercase will be matched as well). This can be done as a pre-processing step during email filtering: instead of searching for vocables with any exact-match in the black-list, the filter may search for matches with zero or small edit distance to any of the entries in the black-list, using the method just described, thus uncovering their guises.

3. EXPERIMENTS

We experimented with two different datasets as described below. The focus was in detecting variants of spam triggering vocables, since these are the main target intended by spammers to fool filters; thus we

did not design experiments regarding detection of non-spamming terms. For all of our experiments we developed scripts in Octave version 3.2.4 (code and datasets are available upon request).

3.1. Sanity test

In order to illustrate the potential feasibility of the method, we conducted a preliminary trial on a list of 186 masked versions of the vocable "**viagra**", that we extracted from the Cockeyed Web bulletin (see "There are 600,426,974,379,824,381,952 ways to spell Viagra" by R. Cockerham in: www.cockeyed.com/lessons/viagra/viagra.html, last visited: 21-June-2013). As can be seen in the excerpt shown in Table 4, this test list includes instances of obfuscation caused by substitution, insertion, deletion and segmentation. In this experiment we scanned each entry in the list and compute its distance to the genuine spam term; then we use the resulting distance matrix to perform sequence alignment [13, 15]. The findings are described next.

Table 4. An excerpt of the **viagra** test set.

via-gra	V-I-A-G-R-A	Vi/agra	Viag&ra
ViaJ1gra	vi**agra	vigra	V|i|a|g|r|a
'V 1 @ G' Ra	ViagrYa	ViaaPrga	viagr*a
v*ia*gra	via*gra*	v*i*ag*ra	*via*g*ra
v*iag*ra*	v*iagr*a*	*v*iag*ra*	vi*a*gr*a*
via*g*r*a*	*v*i*a*g*ra	*v*iag*r*a*	*vi*a*g*r*a

Table 5. Histogram of edit distances of the masked spam term variants contained in the **viagra** dataset.

Distance	0	1	2	3	4	Total
Prevalence	150 (80%)	26 (14%)	9 (5%)	1 (1%)	0 (0%)	186 (100%)

Table 6. The ten detected variants with distance equal to or greater than 2 in the in the **viagra** dataset (obfuscated on top of alignment to the genuine term).

Viagorea	Viargvra	ViaTagra	ViagDrHa	ViaVErga
viag-r-a	via-g-ra	via--gra	viag-r-a	viag-r-a

ViaJ1gra	Viarg-a	ViaZUgra	ViaaPrga	Viaoygra
via--gra	via-gra	via--gra	via-gr-a	via--gra

The distribution of edit distances to the genuine word "**viagra**" on this test set is shown in Table 5. This results show that the algorithm matched perfectly 80% of the examined instances (150 variants).

A closer inspection of the 10 cases with a distance equal to or greater than 2 is given in Table 6. It is worth noting that most of these cases are nevertheless correctly aligned to the genuine spam word (8 out of 10). The variant found with a highest distance of 3 (and wrong alignment as well) was "**ViaVErga**", which in our opinion, would be hardly recognizable by the human eye as the intended spam word. In fact, this variant would convey a different semantic in a language like Spanish. From a practical point of view, a spam filter would define a low threshold in the similarity score computed with our method, for example a distance less than or equal to 2, to activate the blocking mechanism. Such a threshold would have obtained 99% sensitivity in this **viagra** test list.

3.2. Extended test

After validation of the sanity test, a broader experiment was conducted to further study the empirical behaviour of the described method.

For this purpose we visited a number of spam-related web forums and gathered a list of 100 common spam-filter triggering sentences. Then we built a dataset comprising 100 automatically generated obfuscated variants for each spam-trigger sentence with a corruption rate $0 \leq p \leq 1$. Thus, a character Δ in the sentence was kept unaltered with probability $1-p$; otherwise with probability $p/4$ it was exposed to one of the following edits: substitution with one of its admissible characters of Table 2, random substitution with any other character, replication (insertion of up to 10 occurrences of the same character), or insertion of up to 5 bogus repeated characters from the list S_β.

We remark that the random substitution edit operation introduces noise that is meant to have an adverse effect on our matching method. An excerpt of this spam-obfuscated list is shown in Table 7.

The performance of the method in this setting was estimated as follows. We generated datasets as explained above, for a range of values for $p \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. One experiment was run for each dataset. Each of the hundred canonical spam sentences was compared against its hundred variants and the average edit distance $\{\hat{d}_i\}_{i=1}^{100}$ was recorded, where *i* denotes the index of the canonical sentence in the dataset. Then we summarized these results in the average distance histogram shown in Figure 1.

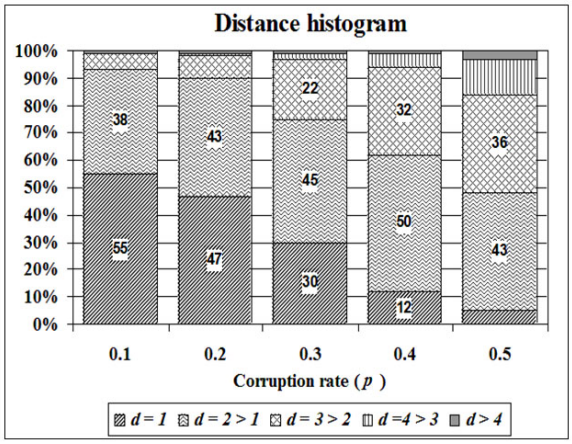

Figure 1. Average distance histogram in the obfuscated datasets for different values of p.

We found that for small corruption rates ($p \in \{0.1, 0.2\}$) the method recognised more than 90% of the obfuscated variants, obtaining an edit distance $\hat{d}_i \leq 2$ (see the darker bottom regions of the two leftmost bars in Figure 1).

On the other hand, higher corruption rates required higher distance thresholds for a 90+% recognition success: $\hat{d}_i \leq 3$ for $p \in \{0.3, 0.4\}$, and $\hat{d}_i \leq 4$ for $p = 0.5$. The latter is nonetheless noteworthy considering that the noise induced with such rate can achieve up to 12% of random substitutions, an obfuscation level that mostly loses its deception factor to the human eye, as it is exemplified in the rightmost column of Table 7.

In this sense one would expect that in order to be effective the obfuscation tactic would be carried out by spammers using the lowest corruption rates. Nonetheless, it is interesting that if a threshold on the edit distance of $\hat{d}_i \leq 4$ were defined to block a suspicious (candidate spam) sentence, the method would have obtained $\approx 97\%$ accuracy in this test, regardless of corruption rate or sentence length.

Further evidence of the latter claim was given by the following analysis. We first measured the average length $\{\bar{l}_i\}_{i=1}^{100}$ of the corrupted sentences, that is, the average number of characters of the one hundred variants generated per each sentence i.

Next we computed the set of ratios $\frac{\hat{d}_i}{\bar{l}_i}$ for each test list generated with different values of p. This ratio gives us an indication of the proportion of average edits needed to transform the variants to its canonical form, with respect to their average lengths.

A surface plot of this measure with respect to the different values of p is displayed in Figure 2. This plot reveals that the five surfaces maintain a constant trend, which in turn indicates that the average number of corrections needed to recognise the spam variant stays at a constant proportion to the average length yielded by their respective corruption rates.

Table 7. An excerpt of the spam-obfuscated test lists. Sentence index (identifier) within the list is indicated in brackets.

Original sentence	Corrupted sentence at p rate	
	$p = 0.1$	$p = 0.5$
amateur teens (2)	amateur teeeeeeeeens amateuêêê teeeens amâteur teemmmms	ammmmaTeur.. ******t:::::eeñs 44mmmmmtÄuuuuuuur____teen::$$ aMXtte/ur||LLLLLLeeeeeeeennnnnnnnnns
best rates (7)	besT rätès best rattteeeeeeeees 8est râtes	be5555t r::::::aaaaaaaaaateeeeeeeees bes******T ®ates bbbbbbbbbe5t r+++++atzzzs;
call free (12)	call free~~~~ calÌ fRee call----- free	c;;;;;;a*****ll frrrrrrrrree calllllllll Fre---e {aLl===== fRe
diet pill (19)	dietttttttt pill ddddiet pill diEEt Pill	ddddddiet p*****ll ddddddiett MMiiill===== RRRiiiiiiie---t=== pill
exclusive pics (24)	exclusive p||||ics exclusive pic5 exclu$$$ ive pi©$$	e******xxcLusi====vè ...pi|||cs èx-----c;;|usIve ppppppppiWssssss ëxcluuuuuuuuusiiive p:c*sssssss
free adult videos (30)	free ad,,,ult videos £ree aduuuuuuuuult videos free dult videosssssss	fre;;;;e adult ..ËËYYeos fffre adu|t ||deos Frebbb ad-uuuuuuuultGvvi|||Dos***

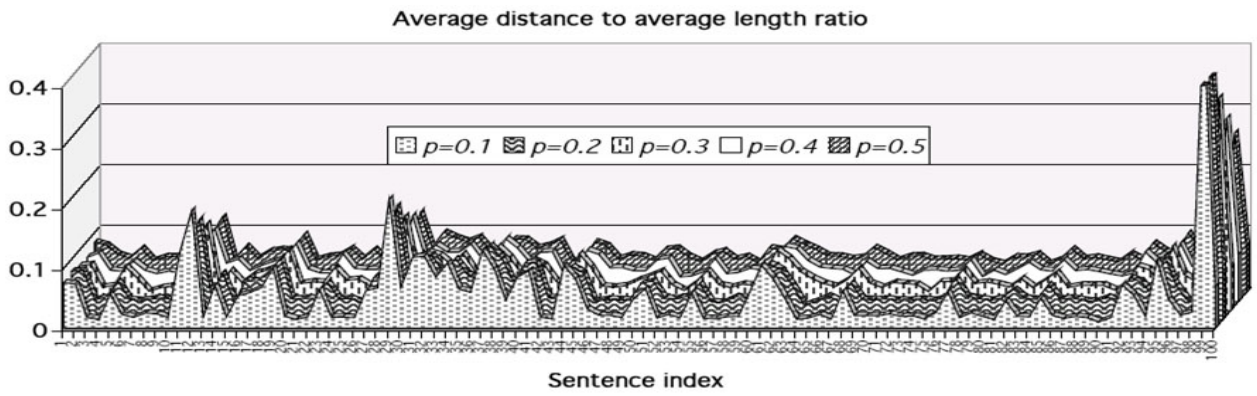

Figure 2. Trend chart of the ratio of average distance to average length in the obfuscated datasets for different values of p. Indices (identifiers) of the collected one hundred spam-triggering sentences are indicated on the horizontal axis.

It is interesting to note that for most sentences the proportion $\hat{d}_i / \hat{l}_i$ not only remains constant irrespective of p, but also is typically smaller than 0.1 (that is, smaller than 10%). There are only four peaks showing a greater proportion, those corresponding to the 11th, 12th, 29th and 99th sentences: $\mathbf{s}_{11}$:="**bukkake**", $\mathbf{s}_{12}$:="**call free**", $\mathbf{s}_{29}$:="**free access pass**" and $\mathbf{s}_{99}$:="**xxx**". This fact is explained by the current difficulty posed to the method when distinguishing between bogus and legitimate character repetitions, a common feature in these four instances. Right now legitimate character repetitions are accounted as one edit; to see why, notice that the first occurrence of the repeated letter in the original term is matched by the algorithm against the repeated occurrences of the same character in the variant; the second (and subsequent) repetitions in the original term are thus accounted as one insertion yielding a cost of 1 according to Equation (1). As a consequence, for example in $\mathbf{s}_{29}$, $\hat{d}_{29} \geq 4$, that is, $1/4$ of the original length l_{29}. The above rationale can be extended to the other cases, $\mathbf{s}_{99}$ being particularly prominent due to its short length.

4. CONCLUSION AND FUTURE WORK

The method we have just described is intended to counter-attack the clever trick of ciphering spam-trigger words with spurious character obfuscation aimed at fooling spam filters. The core of this method is the edit cost function −Equation (1)− which is plugged into a classical string matching algorithm. Such function in fact establishes an equivalence class for the admissible substitutions and segmentators in the masking character universe defined in Table 2 augmented with the segmentation characters of the set S_β.

It is worth noting that the method requires no training before use; it only needs as input a cleaned (canonical) vocabulary list of spam trigger terms. We believe it could be valuable if used as a pre-processing step to uncover obfuscated text inside the contents of an email, in order to subsequently verify the rectified content using the current available and emerging spam-filter technology (as those described in [18] and [19]). It would be also interesting to refine the algorithm with sophisticated string-matching kernel machines so as to incorporate it into state-of-the-art spam filters (some examples in this direction are proposed in [20]).

We would like to conclude emphasizing on a few challenges concerning special cases of spam text obfuscation, currently not solved in the proposed method. The first one is the discrimination between obfuscation and genuine letter replication as discussed before (for example when masking **call free**►**calllll freeeeee**, only the first two "**ll**" and "**ee**" should be accounted as legitimate in the obfuscated sentence). In this regard, at present the algorithm is unable to match spurious repetitions with legitimate doubled-letter words. The second one, a challenge that may pose difficulties on body text parsing, is the discrimination of the blank space, which is not only the most common interword separator but also can be used for visually-undetected letter segmentation (e.g. **sex**►**s e x**). Yet one more compelling challenge is the transposition of consecutive letters (e.g. **password** ►**passwrod**), even though the visual deception factor of this tactic

is moderate and thus its prevalence is uncommon. In a different direction, worthy of attention, is the study of the method when allowing real-valued costs for the edit operations, to see how they effectively contribute to solve the obfuscation tactic.

As a final note, equally appealing are practical issues regarding hardware implementation of the described techniques, such as the study conducted in [21] and potential applications in other obfuscation-prone user-generated content such as text messaging and social networks posts. We remark that any application comprising processing of natural written text such as dialog human-machine-interaction [22], text morphological analysis [23], or speech synthesis from textual input [24], could benefit from the method described here, as it may increase their robustness rates to spelling mistakes.

REFERENCES

[1] Blanzieri, E. and Bryl, A., A survey of learning-based techniques of email spam filtering. Artificial Intelligence Review, 29, pp.63-92, 2008.

[2] Carpinter, J. and Hunt, R., Tightening the net: A review of current and next generation spam filtering tools. Computers and Security, 25(8), pp.566-578, 2006.

[3] Cormack, G., Email spam filtering: A systematic review. Found. Trends Inf. Retr., 1, pp.335-455, April 2008.

[4] Meyer, T.A. and Whateley, B., SpamBayes: Effective open-source, Bayesian-based, email classification system. In: Proceedings of the First Conference on Email and Anti-Spam, 2004.

[5] Zhang, L., Zhu, J. and Yao, T., An evaluation of statistical spam filtering techniques, ACM Transactions on Asian Language Information Processing (TALIP), 3, pp.243-269, 2004.

[6] Ross, N., Writing in the information age. English Today, 22(3), pp.39-45, 2006.

[7] Crystal, D., E-Mail Essentials: How to Make the Most of E-Communications, Kogan Page, 2001.

[8] Yunker, F. and Barry, S., Threaded podcasting: The evolution of on-line learning. In: Proceedings of the International Conference on e-Learning (Ed. Remenyi, D.), Academic Conferences Limited, 2006.

[9] Crystal, D., Txtng: The gr8 db8, Oxford University Press, 2008.

[10] Gordillo, J. and Conde, E., An HMM for detecting spam mail. Expert Systems with Applications, 33, pp. 667- 682, 2007.

[11] Rigoutsos, I. and Huynh, T., Chung-Kwei: a Pattern-discovery-based System for the Automatic Identification of Unsolicited E-mail Messages (SPAM). In: Proceedings of the First Conference on Email and Anti-Spam, 2004.

[12] Lee, S., Jeong, I. and Choi, S., Dynamically weighted Hidden Markov Model for spam deobfuscation. In: IJCAI-07, pp. 2523-2529, 2007.

[13] Kondrak, G., Algorithms for language reconstruction [PhD thesis]. University of Toronto, 2002.

[14] Levenshtein, V., Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady, 10(8), pp.707-710, 1966.

[15] Wagner, R. and Fischer, M., The string-to-string correction problem. Journal of the ACM, 21, pp.168-173, 1974.

[16] Needleman, S. and Wunsch, C., A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, 48(3), pp.443-453, 1970.

[17] Smith, T. and Waterman, M., Identification of common molecular subsequences. Journal of Molecular Biology, 147(1), pp.195-197, 1981.

[18] Almeida, T. and Yamakami, A., Advances in spam filtering techniques. In: Computational Intelligence for Privacy and Security (Eds. Elizondo .et al.), volume 394 of Studies in Computational Intelligence, pp. 199-214, Springer Berlin - Heidelberg, 2012.

[19] Caruana, G. and Li, M., A Survey of Emerging Approaches to Spam Filtering. ACM Computing Surveys, Vol. 44(2), 2012.

[20] Sculley, D., Wachman, G. and Brodley, C., Spam filtering using inexact string matching in explicit feature space with on-line linear classifiers. In: TREC, Volume Special Publication 500-272 (Eds. Voorhees, E. and Buckland, L.), National Institute of Standards and Technology (NIST), 2006.

[21] Tee, H., FPGA unsolicited commercial email inline filter design using Levenshtein distance algorithm and longest common subsequence algorithm [MSc. Thesis]. Faculty of Computer Science and Information Technology, University of Malaya, 2010.

[22] Zapata, C. and Mesa, J., Los modelos de diálogo y sus aplicaciones en sistemas de diálogo hombre-máquina: revisión de la literatura, DYNA, Vol 76 (160), 2009.

[23] Zapata, C. and Mesa, J., Una propuesta para el análisis morfológico de verbos del español, DYNA, Vol 76(157), 2009.

[24] Rueda, H., Correa, C. and Arguello, H., Design and development of speech synthesis software for Colombian spanish applied to communication through mobile devices, DYNA, Vol 79(173), 2012.