

A genomics–based profanity–safe Web forum

Christian Mogollón Pinzón

School of Engineering

Universidad Distrital

Bogotá, Colombia

cgmogollonp@correo.udistrital.edu.co

Sergio Rojas-Galeano

School of Engineering

Universidad Distrital

Bogotá, Colombia

srojas@udistrital.edu.co

Abstract—User-generated text is the primary source of interaction in virtual communities on Web2.0 applications such as forums, blogs or social networks. Unfortunately some users abuse this freedom of speech liberty to disseminate non-authorized profanity content (foul language, insults, advertisement, boosting or denigration of a name or a trademark). Naïve filters based on literal comparisons against black-lists of forbidden terms, fail to detect variations obtained by character transliteration or masking (e.g. writing *piss* as *P!55* or *p.i.s.s*). Recent approaches to this problem inspired in sequence alignment methods from comparative genomics in bioinformatics, have shown promise in preventing overlooking such variants. Building upon those results we have developed an experimental Web forum allowing users to generate text that is screened against transliterated profanity. In this paper we introduce the software (**ForumForte**) and describe briefly the technique and engineering behind it. We anticipate this kind of tools might prove beneficial for content moderation in mainstream applications such as newspaper forums and micro-blogging social networking sites. Our software is open-source under the New BSD License and is available at:

<http://tinyurl.com/ForumForte>



FOROS WEB SIN OBSCENIDADES MEDIANTE TÉCNICAS GENÓMICAS

Resumen— La interacción en comunidades virtuales Web2.0 como foros blogs, o redes sociales, ocurre principalmente a través de intercambio de mensajes de texto entre usuarios. Lastimosamente algunas personas aprovechan esta posibilidad para expresar comentarios con contenido obsceno o propaganda no autorizada. Para evadir los filtros típicos de censura mediante listas negras de palabras, tales usuarios recurren a transliteraciones o enmascaramientos del texto (por ejemplo cambiar *mierda* por *m1erd@* o *m.i.e.r.d.a*). Estudios recientes han propuesto detectar este tipo de variaciones mediante técnicas inspiradas en la genómica comparativa. Siguiendo esta misma línea, hemos desarrollado un foro Web experimental donde los mensajes ingresados por los usuarios son inspeccionados y depurados de obscenidades transliteradas. Este artículo presenta dicho software con una descripción breve de su diseño y su uso con datos reales provenientes de foros de noticias de periódicos y de trinos de una red social. El software se distribuye con licencia libre y también está disponible en línea en: <http://tinyurl.com/ForumForte>.

I. INTRODUCTION

One of the main aspects in Web2.0–based services is the ability of continual incorporation of user-generated content from multiple sources and formats, motivating collaboration and mutual construction of scenarios yielding richer user experiences [1]. An illustrative example are digital forums

and blogs where content is generated in the shape of written text, allowing users to express their own or comment on other’s opinions. Unfortunately this *de facto* freedom of speech is sometimes misused with inappropriate purposes such as insulting or degrading an entity, or the opposite, to boost unauthorized propaganda (*spamming*) or simply to communicate using offensive language to other participants in the community. For these reasons, usually this kind of digital services must be moderated by website administrators in order to guarantee profanity–free user–generated text content.

A naïve moderation tool consists of defining black–lists of forbidden terms that are filtered by a literal comparison against the text. However, these filters fail when facing variations of the original terms due to involuntary typos or misspellings, or more worryingly, due to deliberate attempts where symbols in the word are transliterated with the aim of trespassing the filter while visually carry on the actual meaning of the profanity. Take for example the vulgar slang term *piss* transliterated as *P!55* or *p-i-s-s* or worse still, *P-!-5-5*. Any of these attacks would easily defeat a literal comparison filter, but the message would still be clear for most readers. It is evident that the number of variants obtainable by transliteration grows combinatorial in size; thus, the black–list approach is impractical.

In a similar vein, the described phenomena has been identified in many digital media platforms [2], [3] and furthermore, has been characterised as a security threat [4]. Recent approaches to tackle this problem taking inspiration on bioinformatics techniques used to perform sequence alignment of genomes from different organisms, have shown promising results [5], [6]. Building upon those results, we have developed an experimental profanity–safe Web forum (**ForumForte**). Our software was conceived as a concrete application of our previous results on the problems of revealing masked terms in *spam* email [5], automatic evaluation of fill-in-the-blank questionnaires [7] and automatic syntax verification of short blocks of code in programming languages [8]. The technical details of the mechanism tailored to the Web forum application would be reported elsewhere.

The paper focuses on describing the software, its design, motivation and potential application for content moderation in mainstream applications such as newspaper forums and micro-blogging media platforms. **ForumForte** is available free under the New BSD License and is available online or for download at: <http://tinyurl.com/ForumForte>.

II. MOTIVATION

The transliteration scheme described above has been studied in computer science as the problem of approximate string matching [9]. These algorithms are intended to compute the *edit* character-wise distance between two texts [10], that is, the number of character corrections (substitutions, deletions or insertions) that are needed to transform one text into the other. These techniques were rapidly incorporated into word-processing applications as basic find & replace and spelling correction tools. More recently, they have also been widely used as core technology in search engines and Web crawlers.

Around roughly the same time, researchers in genomics developed algorithms for comparison and alignment of sequences of DNA and protein molecules from the genome and proteome of living organisms [11], [12]. It turned out that those sequences correspond to long strings of letters representing the initials of the molecules (in the DNA case {A, G, C, T} for (A)denine, (C)ytosine, (G)uanine and (T)hymine. Different organisms would have different genomes but when the sequences are aligned, similarities between genome sub-regions (genes) are found, except for a few places that differ. The small variations are due to mutations that insert, delete or substitute one molecule or the other. The mutation may imply a change on the function of the phenotype that the subsequence codes for. An example of the sequence alignment comparison with mutations is shown in Figure 1.

Human	ACCCGAGGT	GGTGCCAC	TACCTGGTG	GGTACGCTT	CACCTG	GAG
Chimpanzee	ACCCGAGGT	GGTGCCAC	TACCTGGTG	GGTACGCTT	CACCTG	GAG
Orangutan	ACCCGAGGT	GGTGCCAC	TACCTGGTG	GGTACGCTT	CACCTG	GAG
Cow	ACCCGAGGT	GGTGCCAC	TACCTGGTG	GGTACGCTT	CACCTG	GAG
Dog	ACCCGAGGT	GGTGCCAC	TACCTGGTG	GGTACGCTT	CACCTG	GAG
Rat	ACCCGAGGT	GGTGCCAC	TACCTGGTG	GGTACGCTT	CACCTG	GAG

Figure 1: Sequence alignment with mutations highlighted of an excerpt of the gene coding for vitamin C synthesis in several species of mammals. Cows, dogs, and rats make their own vitamin C whereas humans and great apes have to take it from diet. Figure credit: [13].

The evolutionist assumption that mutations occurred during millions of generations of descendants from a common ancient genome have given origin to the different species, can be depicted as a phylogenetic tree where branches grow every time a variation has happened. In a similar way one can think of the generation of profanity variants due to transliteration, depicted as a similarity tree where branches grow as a result of edits or corrections over the original term (see Figure 2). Instead of keeping all the possible trees which will grow combinatorially with all possible edits, the idea behind our profanity detection mechanism is to trace back the transliterated variant up to its common ancestor (that is, the canonical version of the forbidden vocable) via classical sequence alignment algorithms [11], [12] adjusted with special-purpose similarity measures designed to tackle the aforementioned transliteration scheme. We shall describe such measure in the next section.

III. MECHANISM

The essence of the filtering mechanism is the dynamic-programming approximate sequence matching algorithm (see

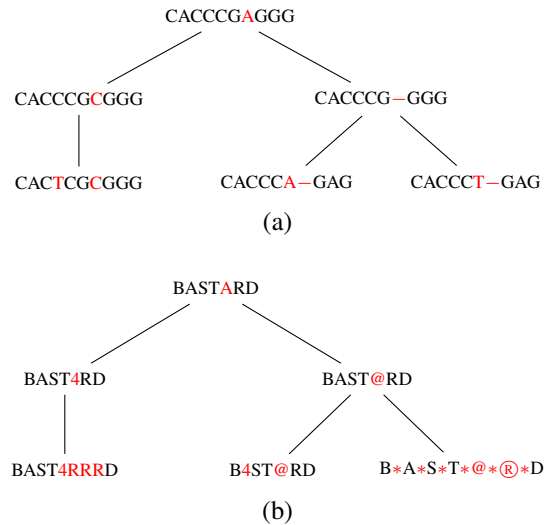


Figure 2: Sequence similarities trees. (a) Phylogenetic tree of the last exon of the gene coding for vitamin C synthesis of Figure 1. Mutations shown in red. (b) Similarity tree of transliterated variants of the offensive word *BASTARD*. Edits shown in red.

e.g. [9], [11], [12]) that performs a pairwise comparison of the symbols in the two sequences (the user-generated text and the canonical profanity vocable), while progressively computing the *edit distance* [10], or broadly speaking, the number of edits (insertions, deletions or substitutions) needed to transform one sequence into the other. The key insight in the transliteration detection problem is to carefully account for the substitution of visually “twin” symbols (e.g. substituting ‘o’ by any of {0, °, ó, ò, ö, ô, Ø, Θ, O}), see Figure 3) as well as the insertion of bogus masking characters such as {., *, ~, |, -, _, :, ;}. These couple of edits should add no value to the distance (or difference) between any two symbols, whereas edits such as deletion, insertion or any other substitution should count. Hence the definition of the edit distance between two symbols in their respective sequences (as originally introduced in [5]) is outlined in Figure 4.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	ñ	o	p	q	r	s	t	u	v	w	x	y	z	.
A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	,
4	8	(ð	é	f	6	#	!	1	c		ñ	Ñ	0	þ	9	@	5	ü	ü	ü	*	ÿ	2	-	^	
â	â	l	l	è	9	!	!	!	!	C	£	ñ	ñ	ñ	Ø	þ	\$	\$	\$	ù	ù	ù	*	ÿ	2	-	
ä	v	{	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	
Ä	V	{	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	
Ä	V	{	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	è	

Figure 3: An excerpt of the lists of twins substitutions.

The cost of filtering one instance of an user-generated text is an important question to address. The classical sequence matching algorithm has a cost in $O(nmc)$ for an input text with length n compared to a profanity vocable of length m and a cost c for the edit distance computation. The profanity vocables are usually short ($m \leq 10$), thus we will assume the cost is $O(nc)$. With respect to the cost c , there are two basic strategies to implement the membership tests needed in the `isBogusSegmentator`, `notBogusSegmentator`

```

function  $d = \text{edit-distance}(a, b)$ 
if  $\text{isNull}(b)$  then
   $d = 1;$  /* deletion */
else if  $\text{isNull}(a)$  and  $\text{notBogusSegmentator}(b)$  then
   $d = 1;$  /* insertion */
else if  $\text{isNull}(a)$  and  $\text{isBogusSegmentator}(b)$  then
   $d = 0;$  /* void insertion */
else if  $\text{isValidSubstitute}(b, a)$  then
   $d = 0;$  /* twin substitution */
else
   $d = 1;$  /* any other substitution */

```

Figure 4: Edit distance function used in the filter mechanism.

and isValidSubstitute predicates of the edit distance function. One can either use lists of variable length for each substitution set in Figure 3 and traverse the lists until the symbols is found; this operation has $O(k\ell)$ time and memory worst-case complexity for a symbol alphabet of size k and maximum substitution list size of ℓ (typically $\ell \ll k$). On the other hand one can define a binary comparison matrix $(\mathcal{M})_{ij} \in \{0, 1\}$ defining if symbol i is substitute of symbol j and vice versa; this strategy has memory $O(k^2)$ complexity and $O(1)$ access time. We chose the matrix-based option since for an ASCII-based symbol table, $k = 127$ yielding a not much expensive memory space yet favouring the constant access cost. Therefore, assuming a profanity dictionary size of t vocables, the matrix implementation yields a total cost per filtering of $O(tn)$ and linear memory $O(n + 127^2)$.

IV. IMPLEMENTATION

The development of the Web forum **ForumForte** was aimed at implementing the filter mechanism described above in an open anonymous setting where users can simply comment on particular topic; no censorship is carried on, nor personal or usage information is collected. Simply put, **ForumForte** is a forum that screens comments against profanity and posts them in a wall with fragments overwritten with an asterisks mask if any (plain or transliterated profanity) is detected. Its main purpose is to provide a test-bed for users to verify the robustness of the filter towards transliterations attacks.

ForumForte was designed as a Web application based on the software architectural MVC pattern [14] and the Java EE platform [15]. In the following, we shall describe the most representative design artefacts obtained during its development.

Figure 5 summarises the use-case scenarios for the software. There are two kind of forum users: visitors and administrator. A visitor can inspect the fora pages and comments within; he himself can also post a comment, in which case the filter mechanism is activated and detection statistics would be collected. Finally he can download files for installation.

On the other hand, there is the administrator user. This user has a password-protected account which allows him to carry out basic maintenance tasks such as forum and subject

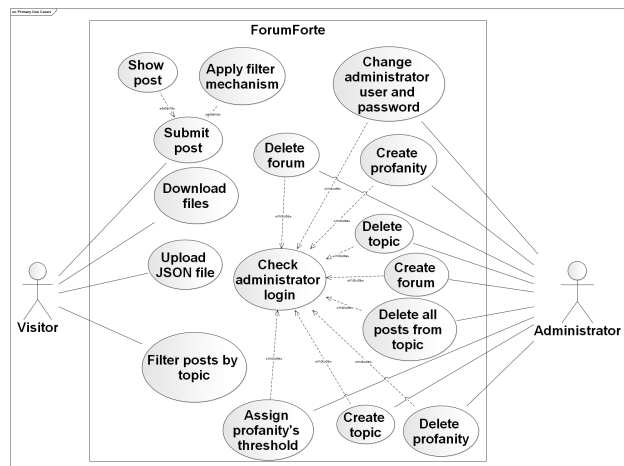


Figure 5: **ForumForte** use-case diagram.

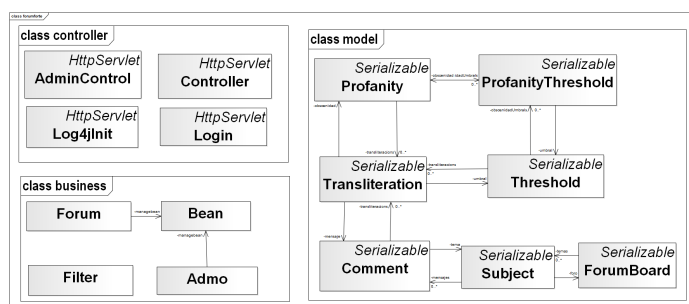


Figure 6: **ForumForte** class diagram.

creation, elimination, cleaning, password update, etc. His other usages are related to the filter mechanism: updating the profanity dictionary, assign edit-distance tolerance, and look at the performance statistics per forum or profanity vocable. The tolerance refers to the maximum edit distance allowed to consider two text sequences as equivalent (a value $\tau \in \{0, 1, 2, 3\}$). The dynamical models (activity and sequencing diagrams) for each use case are available on request.

The structural model of the software was designed as an MVC-based class diagram organised in three packages, namely business, model, and controller. (see Figure 6). The business package include classes `Forum`, `Admin`, `Filter` and `Bean`; these classes implement the logics of the functionalities previously described. On the other hand, the model package encapsulates the `ForumBoard`, `Subject`, `Threshold`, `Transliteration`, `Profanity`; these classes are responsible of managing the visualisation of forum comments and user interface. Finally, the controller package consists of classes `Controller`, `Login`, `Log4j` and `AdminControl`; these classes control interaction with both kind of users. Detailed views of this general model are also available.

Next we discuss briefly the persistence model of the software, which was implemented as a relational database using the Java Persistence API framework that carries out the mapping from the structural model. As a result, Figure 7 shows the ER diagram of the application, consisting of tables

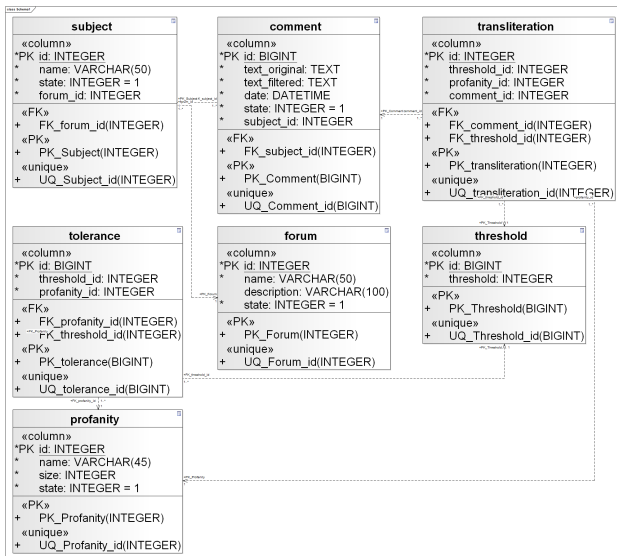


Figure 7: ForumForte persistence model.

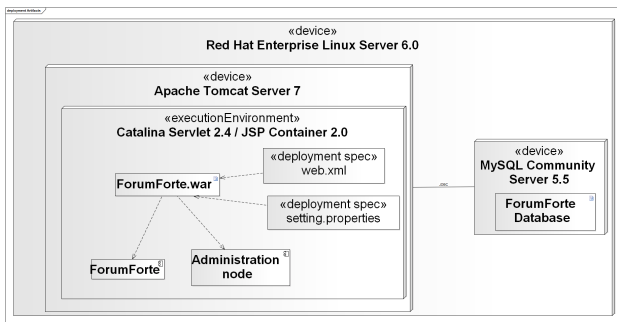


Figure 8: ForumForte deployment diagram.

forum, subject, comment, filteredComment, profanity, and tolerance. The latter were included because the administrator may fine-tune the edit-distance tolerance per profanity term, and therefore distinct detection statistics are collected.

Finally, the deployment diagram of Figure 8 shows the sub-systems used to run the software as a Website application. It can be seen there, that the Catalina Server 2.4 provides the execution environment for the components Administration and WebForum included in the ForumForte.war package. System configuration is defined in the corresponding properties and XML files. This environment runs on a Apache Tomcat 7.0.42 server within a Red Hat v6 Linux OS. Data persistence is achieved through a socket connection to a MySQL 5.5 server within a ForumForte database session.

V. DEPLOYMENT

A. Online release and installation

ForumForte is installed an available online in <http://tinyurl.com/ForumForte> (last visit: September 14th, 2015). The Web application can be utilised with any Web browser (Firefox, Chrome, Explorer and Safari). Alternatively, interested users can download and install the software in their own servers (user and installation guide also available from the same URL).

B. Forum usage

A visitor may browse the available fora by clicking the Foros option in the menu bar, which redirects to the page shown in Figure 9. Any choice here would display the list of subjects previously defined by the administrator in each forum. Then, by choosing one of the subjects, the visitor would be taken to the actual forum page, where comments made by other visitors would be shown (see Figure 10). We remark that interaction with the forum is anonymous and no private or network access information is collected by the system.

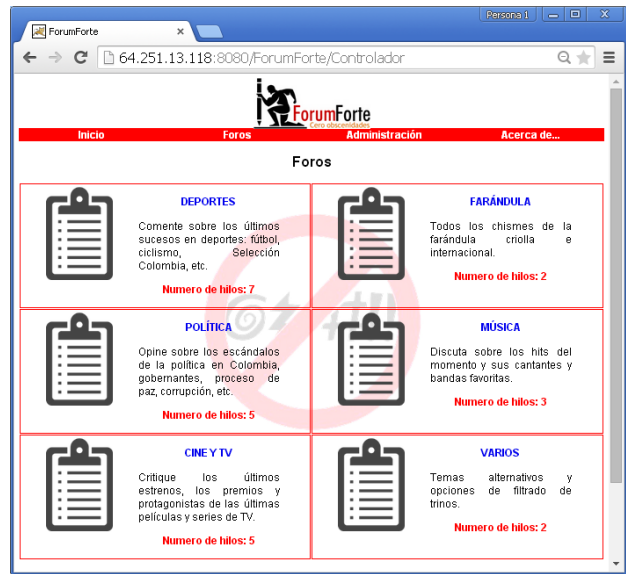


Figure 9: List of available fora in ForumForte.

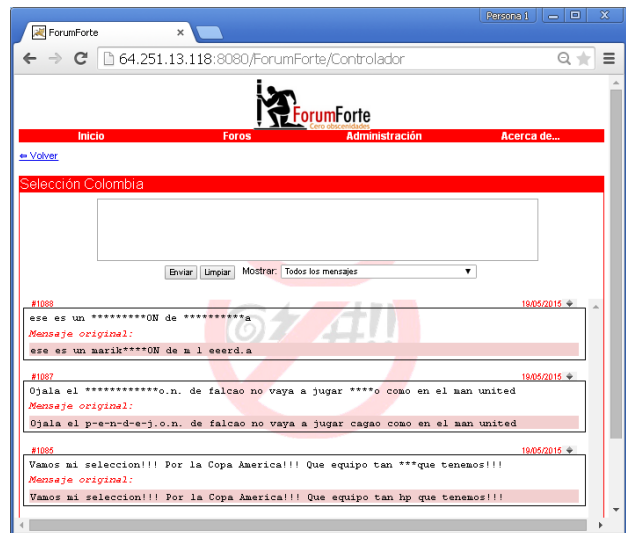


Figure 10: The appearance of an actual forum in ForumForte.

The layout of an actual forum is very intuitive and easy to use. There is basically a text box on the top of the page for the visitor to write his or her comment. The visitor can choose to either clean the current content of the text box, or to post the comment in which case, it would be submitted to the filter

engine. Once the text is processed, the original and filtered text would be posted to the forum wall as the most recent entry (the one just below the text box). Each entry consists of the filtered text sequence aligned over the original text. If one or many profanity transliteration attacks are detected they would be overwritten with a “***. . .***” mask on the top line of the entry. Comments are kept in the wall in chronological order, most recent first. Finally, the visitor may choose to display all comments in the forum, only profanity–marked comments, or only profanity–free comments (see Figure 11).

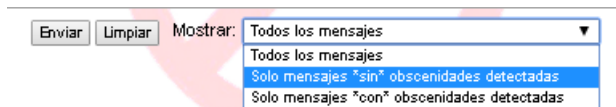




Figure 11: Display options of forum comments.

C. Forum administration

An administrator user can access the dash board of Figure 12 to carry out typical maintenance operations in forums, subjects, profanity dictionary and performance reports. For this purpose, the user should choose the *Administración* option in the menu bar, and confirm his identity with a valid password. The software supports a unique administrator whose username and password can be updated at convenience using the *Login* choice. The remainder options in the board are fora, subjects, profanity dictionary and statistics, which redirect to Web pages where the administrator can create, eliminate or clean contents of the corresponding item.

For the sake of illustration, a sample subject administration page is shown in Figure 13 where the clean up  and remove  commands are visible. The administrator can navigate to the actual forums pages and visualise the comments or participate in the discussion, by clicking over the forum name. Similarly, the profanity dictionary module allows to create, remove or tune the transliteration tolerance of non-admitted vocables in the forum pages.

In addition, the statistics module (see Figure 14) reports detection rates discriminated by forum or by profanity vocable. The latter are furthermore broken down into individual rates per tolerance parameter, providing valuable information for tuning purposes with respect to particular vocables and transliteration attack patterns. Notice that forum statistics are computed instantly with its current comment contents whereas profanity statistics are historical beginning at the moment they were created or assigned a different tolerance parameter. Statistics are lost when the associated item is removed.

VI. FILTERING EXTERNAL USER–GENERATED TEXT

ForumForte features an interesting interface to apply its filtering mechanism to external sources of user–generated text. This feature takes advantage of the JSON format for content extraction and storing provided by the Twitter® social network trough its publicly–available API. This digital media platform is essentially a world–wide community forum for free



Figure 12: The administrator dash board in ForumForte.



Figure 13: A sample subject administrator page in ForumForte.

short text messaging (comments no longer than 140 characters, known as *tweets*) with no moderation, and therefore a real–world scenario of the ideas motivating our development.

Prior to try this feature, the user should prepare an input file with a JSON format. Such file can be obtained by logging in into the Twitter API console with a valid user account¹ and then extracting *tweets* for a chosen user profile or trend topic. The file can then be processed in ForumForte entering the special–designed forum found in the path *Foros* → *Filtrado de trinos*. This page, in contrast with the typical forum template, includes additional options to load and submit the file to the filtering engine (see Figure 15), which would process each tweet in the file and post it to the forum as if it was originally typed in by a visitor.

We highlight that by adhering to the Twitter JSON format, it is possible to filter user–generated text from other sources.

¹The API Twitter is available at: <https://dev.twitter.com/rest/tools/console> (last visit: September 14, 2015)



Figure 14: The performance statistics module in ForumForte.

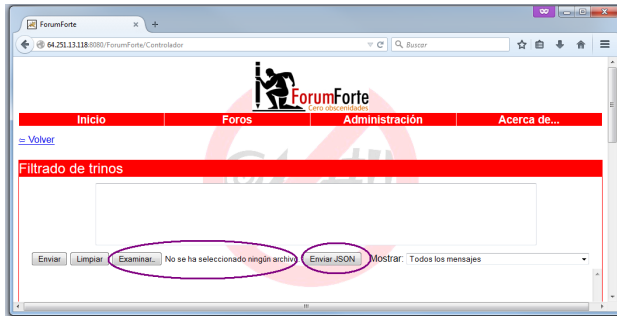


Figure 15: External JSON file processing commands in ForumForte.

To illustrate this point, we have setup a JSON file with 300 profanity messages taken from the user comments of news forums in a countrywide circulation newspaper from Colombia. We curated the messages in advanced and labeled them according to a profanity list of colombian obscenities. The detection rates of the filtering mechanism are reported in Table I, which demonstrate the feasibility and promise of the method in real-world scenarios. The JSON example files described earlier are available on request.

Obscenity length (m)	Ground truth	Detections per tolerance parameter			
		$\tau = 0$	$\tau = 1$	$\tau = 2$	$\tau = 3$
2	28	33	2652	720	720
3	33	25	2085	5774	1053
4	205	186	1920	12277	21449
5	136	115	281	2517	9502
6	33	32	60	461	4688
7	33	28	29	85	647
8	26	19	25	34	94
9	43	33	36	53	92

Table I: Detection rates in the real-life dataset. Greyed values denote detection rates closest to the ground truth. Higher values indicate occurrence of false positives. In shorter obscenities ($m \leq 6$) a tolerance $\tau = 0$ is more effective, whereas medium size or larger obscenities ($m \geq 7$), require higher tolerances $\tau = 1, 2$.

VII. CONCLUSION

The transliteration attack in user-generated text exploits the visual ability of the human mind to interpret the semantics of a message overwritten with substitutions of twins symbols or insertion of bogus segmentations. Blacklist-based computer filters, on the contrary, have limited ability to detect such variants. Taken inspiration on algorithms for sequence alignment widely-used in bioinformatics, we have built an academic software tool to counter fight such anomaly, showing promising results. We anticipate this technology may have interesting applications for content moderation in Web2.0 digital services such as forums, blogs and social networks. It is worth to note that although the case study showed in this paper was targeted at Spanish obscenities, the mechanism is language-independent and requires no training before use other than setting up a base-list of canonical forbidden terms.

A critical task during the operation of the mechanism would be related to fine-tuning the canonical versions in the base-list along with their tolerance parameters for particular purpose or language, while tracking the behaviour of malicious users in coming up with new transliteration strategies. Performing automatic tuning is an interesting topic of further research. Another potential avenue is to study speed and concurrency issues on high-volume content-generation environments.

REFERENCES

- [1] T. O'reilly, "What is Web 2.0: Design patterns and business models for the next generation of software," *Communications & strategies*, no. 1, p. 17, 2007.
- [2] S. Sood, J. Antin, and E. Churchill, "Profanity use in online communities," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2012, pp. 1481–1490.
- [3] W. Wang, L. Chen, K. Thirunarayan, and A. P. Sheth, "Cursing in English on Twitter," in *Proceedings of the 17th ACM conference on computer supported cooperative work & social computing*, 2014.
- [4] M.-E. Maurer and L. Höfer, "Sophisticated phishers make more spelling mistakes: using URL similarity against phishing," in *Cyberspace Safety and Security*. Springer, 2012, pp. 414–426.
- [5] S. A. Rojas-Galeano, "Revealing non-alphabetical guises of spam-trigger vocables," *DYNA*, vol. 80, pp. 15 – 24, 2013.
- [6] X. Zhong, "Deobfuscation based on edit distance algorithm for spam filtering," in *Machine Learning and Cybernetics (ICMLC), 2014 International Conference on*, vol. 1. IEEE, 2014, pp. 109–114.
- [7] V. P. Cardona-Zea and S. A. Rojas-Galeano, "Recognising irregular answers in automatic assessment of fill-in-the-blank tests," in *Engineering Applications (WEA), 2012 Workshop on*. IEEE, 2012, pp. 1–4.
- [8] S. A. Rojas-Galeano, "Towards automatic recognition of irregular, short-open answers in Fill-in-the-blank tests," *Tecnura*, vol. 18, 2014.
- [9] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM*, vol. 21, pp. 168–173, 1974.
- [10] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, 1966.
- [11] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443 – 453, 1970.
- [12] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, 1981.
- [13] D. Venema. (2013, June 28) Evolution basics: Genomes as ancient texts. The BioLogos Forum. Last visited: September 14th, 2015. [Online]. Available: <http://biologos.org/>
- [14] A. Leff and J. T. Rayfield, "Web-application development using the model/view/controller design pattern," in *Enterprise Distributed Object Computing Conference, Proceedings. Fifth IEEE International*, 2001.
- [15] D. Alur, D. Malks, J. Crupi, G. Booch, and M. Fowler, *Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies*. Sun Microsystems, Inc., 2003.