Online Social Networks and Media 9 (2019) 1–16

Contents lists available at ScienceDirect

Online Social Networks and Media

journal homepage: www.elsevier.com/locate/osnem

On the capability of evolved spambots to evade detection via genetic engineering

Stefano Cresci[a], Marinella Petrocchi[a], Angelo Spognardi[b,*], Stefano Tognazzi[c]

[a] Istitute of Informatics and Telematics (IIT-CNR), via G. Moruzzi 1, Pisa, Italy
[b] Department of Computer Science, "La Sapienza" University of Rome, Roma, Italy
[c] IMT School for Advanced Studies Lucca, Piazza San Francesco 19, Lucca, Italy

ARTICLE INFO

Article history:
Received 8 June 2018
Revised 4 October 2018
Accepted 28 October 2018
Available online 24 November 2018

Keywords:
Spambots
Social networks
Detection mechanisms
Genetic algorithms
Social fingerprinting
Online behavioral modeling
Twitter
Proactive *vs.* reactive detection -approaches

ABSTRACT

Since decades, *genetic algorithms* have been used as an effective heuristic to solve optimization problems. However, in order to be applied, genetic algorithms may require a string-based genetic encoding of information, which severely limited their applicability when dealing with online accounts. Remarkably, a behavioral modeling technique inspired by biological DNA has been recently proposed – and successfully applied – for monitoring and detecting spambots in Online Social Networks. In this so-called *digital DNA* representation, the behavioral lifetime of an account is encoded as a sequence of characters, namely a digital DNA sequence. In a previous work, the authors proposed to create synthetic digital DNA sequences that resemble the characteristics of the digital DNA sequences of real accounts. The combination of (i) the capability to model the accounts' behaviors as digital DNA sequences, (ii) the possibility to create synthetic digital DNA sequences, and (iii) the evolutionary simulations allowed by genetic algorithms, open up the unprecedented opportunity to study – and even *anticipate* – the evolutionary patterns of modern social spambots. In this paper, we experiment with a novel ad-hoc genetic algorithm that allows to obtain behaviorally evolved spambots. By varying the different parameters of the genetic algorithm, we are able to evaluate the capability of the evolved spambots to escape a state-of-art behavior-based detection technique. Notably, despite such detection technique achieved excellent performances in the recent past, a number of our spambot evolutions manage to escape detection. Our analysis, if carried out at large-scale, would allow to proactively identify possible spambot evolutions capable of evading current detection techniques.

1. Introduction

As part of today's ongoing socio-technical convergence, Online Social Networks (OSNs) have a profound impact on our everyday life. We increasingly rely on OSNs content in order to form our opinions, to plan activities, and to establish social relationships. One of the most striking examples of the influence that OSNs have on our societies could be witnessed during all the latest political elections. Indeed, during the 2014 Italian mayoral elections, the 2016 US presidential elections, the 2016 UK Brexit referendum, and the 2017 French presidential elections, social media played a dominant role in the electoral campaigns, often contributing to invert the foreseen electoral outcome[1].

It is not surprising that OSNs have also been exploited for maliciously influencing the public opinion [1,2]. One common way to achieve this goal is to employ large groups of automated (bot) accounts (henceforth *spambots*) that repeatedly spam polarized content. Worryingly, this malicious practice is pervasive: it has been witnessed in online discussions on important societal topics (e.g., politics, finance, terrorism, immigration) [3,4], as well as in debates about seemingly less relevant topics, such as products on sale on e-commerce platforms and mobile applications [5].

Among the peculiar characteristics of spambots is that they *evolve* over time, by changing their behavior – and, in general – by adopting techniques in order to evade existing detection systems [6]. As spambots became clever in escaping detection, scholars

Conflict of Interest: None.
* Corresponding author.

E-mail addresses: s.cresci@iit.cnr.it (S. Cresci), m.petrocchi@iit.cnr.it (M. Petrocchi), spognardi@di.uniroma1.it (A. Spognardi), stefano.tognazzi@imtlucca.it (S. Tognazzi).

[1] http://www.newsweek.com/full-list-russian-twitter-bots-banned-election-meddling-probe-700703

https://doi.org/10.1016/j.osnem.2018.10.005
2468-6964/© 2018 Elsevier B.V. All rights reserved.

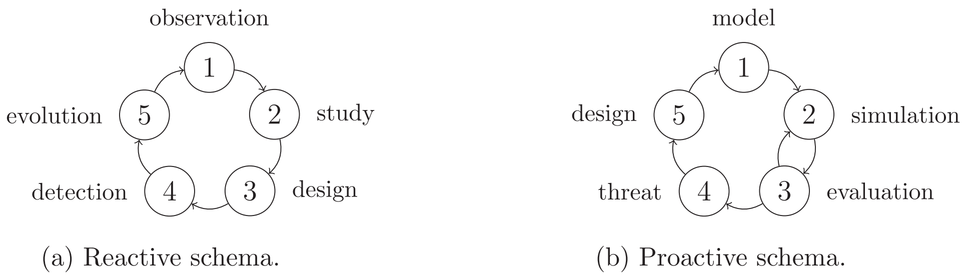


Fig. 1. The story so far and the way ahead: reactive *vs.* proactive spambot detection schemas.

and OSNs administrators tried to keep pace (i.e., *reacted*) and proposed more complex detection techniques. Most notably, spambot evolution still goes on: recent investigations highlight that new waves of *social spambots* are rising and that a paradigm-shift is taking place in spambot design [5,7]. Social spambots are now mimicking legitimate behaviors and interaction patterns in OSNs better than ever before. Being almost indistinguishable from legitimate accounts, social spambots are capable of sharing (credible) fake news, inflating the popularity of OSN users, and “reshaping political debates. They can defraud businesses and ruin reputations”[2]. We are still unable to effectively deal with these issues.

Spambot detection in OSNs is thus a never-ending clash, involving the design of techniques capable of efficiently identifying ever-evolving spammers. Until now, spambot detection has always followed a *reactive* schema [8]. As shown in Fig. 1(a), this schema starts with an observation of suspicious behaviors in OSNs, which leads to a study of malicious activities. Such study is exploited to design new detection techniques. As soon as the new detection techniques are deployed, spambot developers tweak the characteristics of their accounts, thus evading detection. This evolution therefore requires new observations to grasp the characteristics of the evolved spambots. As a consequence of this reactive schema, scholars and OSN administrators are constantly one step behind. In turn, this means that spambots are largely left free to tamper with our online environments until a new detection technique is designed and deployed.

In this work, we propose a new *proactive* approach, schematically shown in Fig. 1(b). The idea is to break the traditional reactive cycle and move towards a new paradigm. In the most general definition of this paradigm, the approach begins by defining a model for OSN accounts. The malicious accounts under investigation are thus represented/encoded via the specified model. Their representation is the input of the simulation step. The high-level goal of the simulation step is to find variants of the input representations (i.e., variants of the malicious accounts) that satisfy a given criterion. Within our context, the criterion could be that of resembling the original representations, or of resembling real legitimate accounts. In other words, the simulation could produce representations of malicious accounts that do not currently exist, but that would appear as similar to existing ones. Then, every modified representation that comes out of the simulation is evaluated in the namesake step. The evaluation aims at verifying whether the modified representations of the accounts are capable of evading detection. Those representations that evade detection are considered as possible threats, and could be taken into account during the design and development of novel detection techniques.

In the following, we instantiate the general proactive approach previously described, to the specific Twitter scenario considered in this study. Our model for OSN accounts encodes the online behaviors of a group of Twitter accounts, and is based on the technique discussed in [9]. The simulation step, implemented by an ad-hoc genetic algorithm, generates several different groups of synthetic Twitter accounts according to different sets of parameters settings. The synthesis of such groups of accounts is carried out in an iterative process where, at each stage, part of the synthetic accounts is discarded in favor of other synthetic accounts that are proven to be more similar to real legitimate accounts. With the goal of analyzing possible spambot evolutions, and thus having a mean to also anticipate their detection, the synthetic accounts generated during the simulation phase are evaluated against a state-of-the-art detection technique [10]. Those accounts that are capable of going undetected by the detection technique, represent a possible threat. Closing the loop, with the proposed approach one would ultimately be able to foresee possible future evolutions of spambots, and to *counteract* by revising and/or designing novel detection techniques, effective even before the actual evolution takes place.

1.1. Goals and implementation choices

The goal of this work is to test whether, and to what extent, it is possible to implement the proactive scheme for the Twitter scenario. In particular, the study will focus on the first four phases, *model, simulation, evaluation*, and *threat*. A behavioral modeling technique inspired by biological DNA will constitute the bases for modeling the behavior of online accounts [9]. Then, in order to study possible future evolutions of spambots, we will introduce a simulation technique based on a popular optimization meta-heuristic, namely, genetic algorithms [11]. We consider different values for the parameters of the adopted genetic algorithm: in particular, we vary (1) the behavior of the accounts given in input to the algorithm; (2) the cost function that the algorithm aims to minimize; and (3) the probability of occurrence of functions that are commonly used by genetic algorithms, namely mutations and crossover. We then evaluate the generated accounts against a state-of-the-art detection technique [10] and we identify possible future threats, i.e., spambots which are currently not detectable by that technique.

1.2. Broadening the approach

This paper tests the reactive approach by focusing on the behavior of the accounts (i.e., the sequences of actions that accounts perform). This choice opens up the possibility to leverage for our experiments the *digital DNA* behavioral modeling technique [9] as well as the *Social Fingerprinting* spambot detection technique [10]. However, despite our implementation of the proactive approach, similar analyses could be carried out by relying on different modeling and spambot detection techniques, such as those based on network/graph analysis and those based on content analysis.

[2] https://www.nytimes.com/interactive/2018/01/27/technology/social-media-bots.html

1.3. Roadmap

The remainder of this paper is structured as follows. The next section offers a background on the behavioral modeling technique, the spambot detection technique, and the genetic algorithm adopted for this study. Section 3 introduces the experiments for the synthesis of new account behaviors. In Section 4, we apply the spambot detection technique to the synthetic accounts generated in the experimental section and we evaluate the results, in terms of the capability of the new generations to escape detection. Then, Section 5 presents a critical discussion of our results, highlighting the capability of some of the evolved bots to perpetrate their malicious activities (e.g., mass retweeting, URL spamming), while remaining undetected. Section 6 briefly surveys recent related literature. Finally, Section 7 concludes the paper and suggests promising directions of future work.

2. Background

In this section, we give preparatory notions on the technique chosen to model the accounts behaviors, the connected spambot detection technique, and the genetic algorithm adopted to let accounts evolve.

2.1. Digital DNA modeling technique and social fingerprinting detection mechanism

Inspired by biological DNA, in [9] we proposed modeling online user behaviors with strings of characters representing the sequence of a user's online actions. Each action type (such as posting new content, or following, or replying to a user) can be encoded with a different character, just as in DNA sequences, where characters encode nucleotide bases. According to this paradigm, online user actions would represent the bases of their *digital DNA*. Different kinds of user behavior can be observed on the Internet [12], and digital DNA is a flexible and compact way of modeling such behaviors [13]. The flexibility lies in the possibility of choosing which actions form the sequence. For example, digital DNA sequences on Facebook could include a different base for each user-to-user interaction type: comments (C), likes (L), shares (S), and mentions (M). Then, interactions can be encoded as strings formed by such characters according to the sequence of user-performed actions. Similarly, user-to-item interactions on an e-commerce platform could be modeled by using a base for every product category. User purchasing behaviors could be encoded as a sequence of characters according to the category of products they buy. In this regard, digital DNA shows a major difference from biological DNA, where the four nucleotide bases are fixed. In digital DNA, both the number and the meaning of the bases can change according to the behavior or interaction to be modeled. Just like its biological predecessor, digital DNA is a compact representation of information – for example, a Twitter user's timeline could be encoded as a single string of 3,200 characters (one character per tweet).

A digital DNA sequence can be defined as a row-vector of characters (i.e., a string),

$$\mathbf{s} = (b_1, b_2, \dots, b_n) \quad b_i \in \mathbb{B} \;\; \forall \;\; i = 1, \dots, n$$

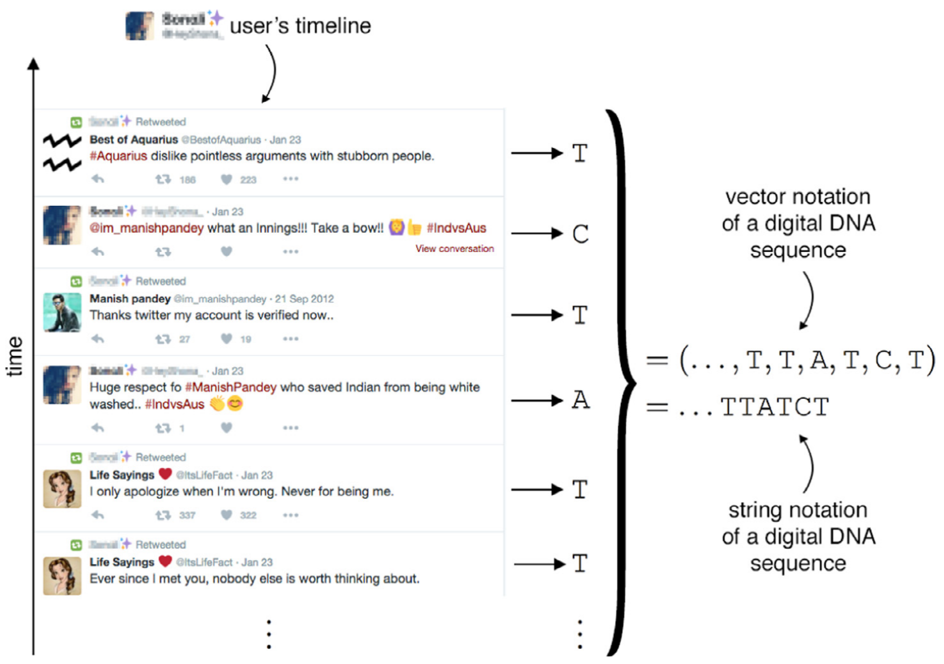
Characters b_i in $\mathbf{s}$ are drawn from a finite set, called *alphabet*,

$$\mathbb{B} = \{B_1, B_2, \dots, B_N\} \quad B_i \neq B_j \;\; \forall \;\; i, j = 1, \dots, N \quad i \neq j$$

The B_i characters are also called the (DNA) *bases* of the alphabet $\mathbb{B}$. A user's behavior can be represented with a digital DNA sequence by encoding each action of the user with an alphabet base. Then, by scanning the user's actions in chronological order and by assigning the appropriate base to each action, one can obtain the sequence of characters that makes up the digital DNA sequence of the user. For example, Fig. 2 shows the process of extracting the digital DNA sequence of a Twitter user, by scanning its timeline according to an alphabet defined as follows:

$$\mathbb{B}^3_{type} = \left\{ \begin{array}{l} \texttt{A} \Leftarrow \text{tweet}, \\ \texttt{C} \Leftarrow \text{reply}, \\ \texttt{T} \Leftarrow \text{retweet} \end{array} \right\} = \{\texttt{A}, \texttt{C}, \texttt{T}\}$$

The $\mathbb{B}^3_{type}$ alphabet encodes user behaviors according to their type of tweets produced, either *tweets*, *retweets*, or *replies*.


user's timeline
time
T
C
T
A
T
T
vector notation of a digital DNA sequence
$= (\dots, \texttt{T}, \texttt{T}, \texttt{A}, \texttt{T}, \texttt{C}, \texttt{T})$
$= \dots \texttt{TTATCT}$
string notation of a digital DNA sequence

Retweeted
Best of Aquarius @BestofAquarius · Jan 23
#Aquarius dislike pointless arguments with stubborn people.
186 223

· Jan 23
@im_manishpandey what an Innings!!! Take a bow!! #IndvsAus
View conversation

Retweeted
Manish pandey @im_manishpandey · 21 Sep 2012
Thanks twitter my account is verified now..
27 19

· Jan 23
Huge respect fo #ManishPandey who saved Indian from being white washed.. #IndvsAus
1

Retweeted
Life Sayings @ItsLifeFact · Jan 23
I only apologize when I'm wrong. Never for being me.
337 322

Retweeted
Life Sayings @ItsLifeFact · Jan 23
Ever since I met you, nobody else is worth thinking about.

Fig. 2. The digital DNA behavioral modeling technique. Excerpt of a digital DNA extraction process for a Twitter user with the $\mathbb{B}^3_{type}$ alphabet. By scanning the user's actions in chronological order and by assigning the appropriate base to each action, one can obtain the sequence of characters that makes up the digital DNA sequence of the user.

2.1.1. Similarity between digital DNA sequences

In [10], the authors proposed a spambot detection technique, grounded on the digital DNA modeling paradigm.

In order to analyze groups of users rather than single users, we studied multiple digital DNA sequences as a whole. A group $\mathbf{A}$ of $M = |\mathbf{A}|$ users can be described by the digital DNA sequences of the M users,

$$\mathbf{A} = \begin{pmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \vdots \\ \mathbf{s}_M \end{pmatrix} = \begin{pmatrix} (b_{1,1}, b_{1,2}, \ldots, b_{1,n}) \\ (b_{2,1}, b_{2,2}, \ldots, b_{2,m}) \\ \vdots \\ (b_{M,1}, b_{M,2}, \ldots, b_{M,p}) \end{pmatrix}$$

The group $\mathbf{A}$ is defined as a column-vector of M digital DNA sequences of variable length, one sequence for each user of the group.

To perform analyses on groups of digital DNA sequences, we can rely on recent advances in the fields of bio-informatics and string mining. In fact, many efficient algorithms and techniques for the analysis of biological strings have been continually proposed in such fields [14]. One of the possible means to quantify similarities between sequential data representations, such as the digital DNA sequences, is the *longest common substring* [15]. Intuitively, users that share long behavioral patterns are much more likely to be similar than those that share short behavioral patterns. Given two strings, $\mathbf{s}_i$ of length n and $\mathbf{s}_j$ of length m, their longest common substring (henceforth LCS) is the longest string that is a substring of both $\mathbf{s}_i$ and $\mathbf{s}_j$. For example, given $\mathbf{s}_i = \texttt{MASSACHUSETTS}$ and $\mathbf{s}_j = \texttt{PARACHUTE}$, their LCS is the string ACHU and the LCS length is 4. The extended version of this problem that considers an arbitrary finite number of strings, is called the *k-common substring* problem [16]. In this case, given a vector $\mathbf{A} = (\mathbf{s}_1, \ldots, \mathbf{s}_M)$ of M strings, the problem is that of finding the LCS that is common to at least k of these strings, for each $2 \leq k \leq M$. Notably, both the *longest common substring* and the *k-common substring* problems can be solved in linear time and space, by resorting to the generalized suffix tree and by implementing state-of-the-art algorithms, such as those proposed in [15]. Given that, in the *k-common substring* problem, the LCS is computed for each $2 \leq k \leq M$, it is possible to plot a *LCS curve*, showing the relationship between the length of the LCS and the number k of strings.

As a direct consequence of the definition of LCS, as the number k of accounts grows, the length of the LCS common to all of them shortens. In other words, LCS curves are *monotonic non-increasing* functions, such that,

$$\text{LCS}[k-1] \geq \text{LCS}[k] \quad \forall \;\; 3 \leq k \leq M$$

Thus, it is more likely to find a long LCS among a few accounts rather than among large groups.

2.1.2. DNA fingerprinting

In contrast with classification and supervised approaches, we devise an unsupervised way to detect spambots, by comparing their behavior with the aim of finding similarities between automated accounts. We can model the behavior of groups of spambots via their digital DNA, comparing it to that of a sample of genuine accounts. The intuition is that, because of their automated nature, spambots are likely to present higher similarities in their digital DNA with respect to the more heterogeneous genuine users.

To exploit at its best the potential of digital DNA, building on the definitions of digital DNA and LCS curves given in Section 2.1 and 2.1.1, here we study the characteristics of the LCS curves of three different datasets, see Table 1.

The `humans` dataset is a random sample of genuine (human-operated) accounts. Following a hybrid crowdsensing approach [17], we randomly contacted Twitter users by asking them a simple question in natural language. All the replies to our questions were manually verified and all the 3474 accounts that answered were certified as humans. The accounts that did not answer to our question were discarded.

The `Bot1` dataset was created after observing the activities of a group of social bots that we discovered on Twitter during the last Mayoral election in Rome, in 2014. One of the runners-up employed a social media marketing firm for his electoral campaign, which made use of almost 1000 automated accounts on Twitter to publicize his policies. Surprisingly, we found such automated accounts to be similar to genuine ones in every way. Every profile was accurately filled in with detailed – yet fake – personal information such as a (stolen) photo, (fake) short-bio, (fake) location, etc. Those accounts also represented credible sources of information since they all had thousands of followers and friends, the majority of which were genuine users[3]. Furthermore, the accounts showed a tweeting behavior that was apparently similar to those of genuine accounts, with a few tweets posted every day, mainly quotes from popular people, songs, and *YouTube* videos. However, every time the political candidate posted a new tweet from his official account, all the automated accounts retweeted it in a time span of just a few minutes. Thus, the political candidate was able to reach many more accounts in addition to his direct followers and managed to alter Twitter engagement metrics during the electoral campaign. Amazingly, we also found tens of human accounts who tried to engage in conversation with some of the spambots. The most common form of such human-to-spambot interaction was represented by a human reply to one of the spambot tweets quotes.

The `Bot2` dataset advertise products on sale on *Amazon.com*. The deceitful activity was carried out by spamming URLs pointing to the advertised products. Similarly to the retweeters of the Italian political candidate, also this family of spambots interleaved spam tweets with harmless and genuine ones.

Remarkably, the spambot datasets captured two of the different dimensions currently exploited by tamperers to perpetrate their illicit activities: retweet frauds and URL spamming. It is also very interesting that both such types of social spambots are still mostly active, since the large majority of them ($>95\%$) has not yet suspended nor deleted by Twitter [18].

We evaluate the differences and similarities among those groups, as seen through the lenses of the digital DNA sequences.

Fig. 3 shows a comparison between the LCS curves of genuine (human) accounts and those of the `Bot1` (Fig. 3(a)) and `Bot2` (Fig. 3(b)) groups. As shown, the LCS of both groups of spambots are rather long even when the number of accounts grows. This is strikingly evident in Fig. 3(b) (`Bot2` – spammers of *Amazon.com* products). For both the spambot groups, we observe a sudden drop in LCS length when the number of accounts gets close to the group size, namely at the end of the x axis. In contrast to the remarkably high LCS curves of spambots, genuine accounts show little to no similarity – as represented by LCS curves that exponentially decay, rapidly reaching the smallest values of LCS length.

This preliminary yet considerable differences between the LCS curves of genuine accounts and spambots suggest that, despite the advanced characteristics of these novel spambots, the $\mathbb{B}^3_{type}$ digital DNA is able to uncover traces of their automated and synchronized activity. In turn, the automated behaviors of a large group of accounts results in exceptionally high LCS curves for such accounts. Indeed, we consider high behavioral similarity as a proxy for automation and, thus, an exceptionally high level of similarity among a large group of accounts might serve as a red flag for anomalous behaviors.

[3] This was made possible also by the adoption of social engineering techniques, such as the photo of a young attractive woman as the profile picture and the occasional posting of provocative tweets.

Table 1
Statistics about three Twitter datasets.

			Relationships		
Dataset	**Accounts**	**Tweets**	Followers	Friends	Total
Bot1 (retweeters of political candidates)	991	1,610,176	4,031,897	4,022,884	8,054,781
Bot2 (*Amazon* spammers)	464	1,418,626	1,352,370	818,913	2,171,283
Humans	3474	8,377,522	4,740,286	2,153,107	6,893,393

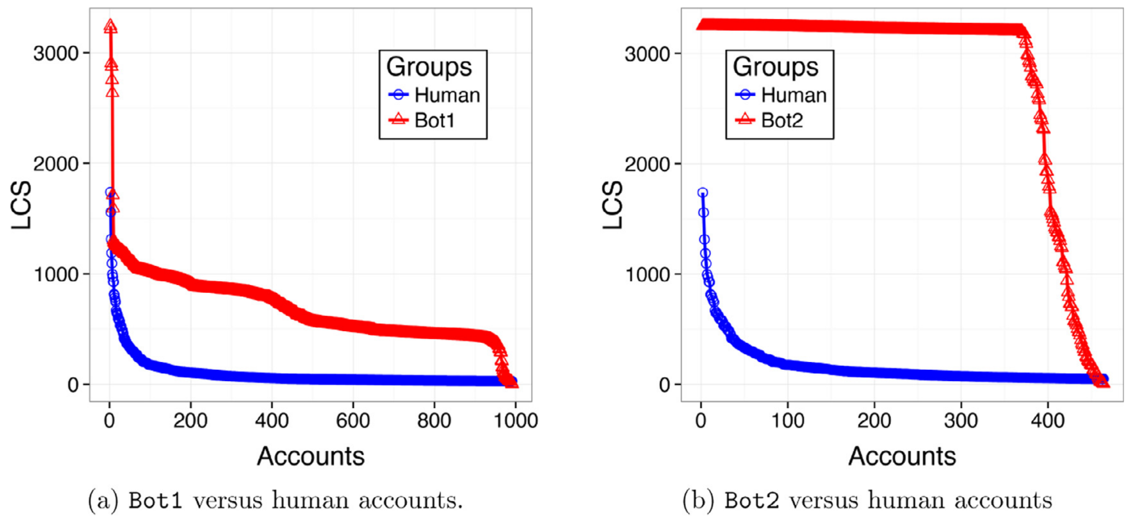


Fig. 3. Comparison between LCS curves of spambots and genuine accounts for the $\mathbb{B}^3_{type}$ alphabet.

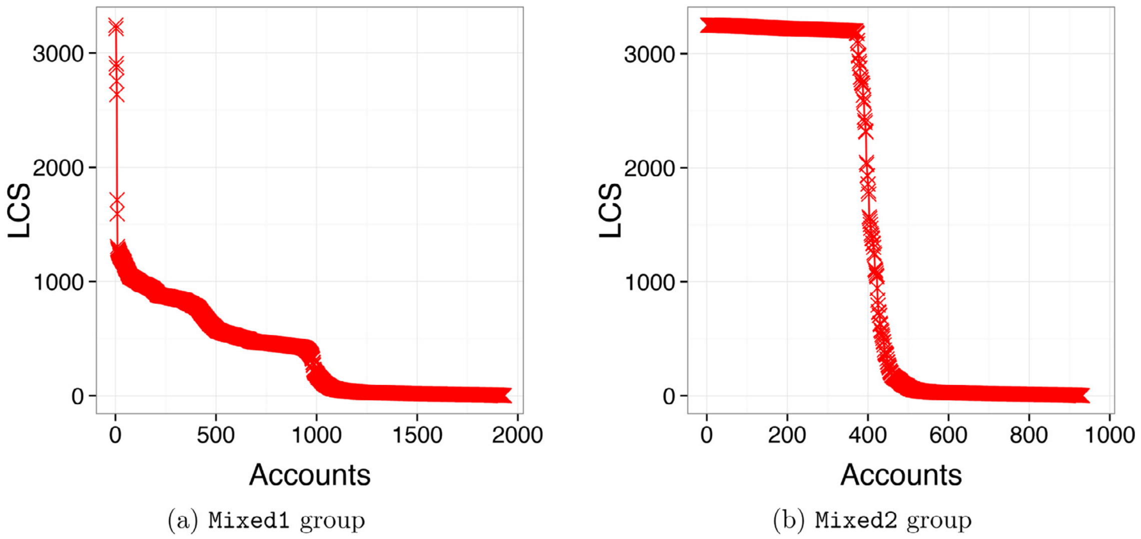


Fig. 4. LCS curves for two groups of heterogeneous accounts, modeled via the $\mathbb{B}^3_{type}$ alphabet.

2.1.3. Unsupervised detection of subgroups of similar users

In previous sections, we saw that groups with different characteristics lead to qualitatively different LCS curves. Here, we consider LCS curves obtained from sequences of an unknown and heterogeneous group of users. We mix together all the spambots of the Bot1 and Bot2 groups, with an equal number of genuine accounts. Henceforth, such heterogeneous groups of accounts are referred to as Mixed1 and Mixed2, respectively. Fig. 4 shows the LCS curves obtained via the $\mathbb{B}^3_{type}$ alphabet.

Such a slow decrease is sometimes interleaved by steeper drops, such as those occurring in the region of 500 and 1000 accounts. Another – and even more evident – steep drop is shown in the right hand plot of Fig. 4, in the region of 400 accounts. LCS curves in both plots asymptotically reach their minimum value as the number of accounts grows. Overall, such LCS curves show a different behavior than those related to a single group of similar accounts, such as the ones shown in Fig. 3(a) and (b). Indeed, the plots of Fig. 4 lack a single trend that spans for the whole domain of the LCS curves. Instead, they depict a situation where a trend seems to be dominant only until reaching a certain threshold. Then, a steep fall occurs and another – possibly different – trend kicks in. Notably, such portions of the LCS curves separated by the steep drops resemble LCS curves of the single groups of similar users (i.e., Bot1, Bot2, human) used to obtain the sets of heterogeneous users (i.e., Mixed1, Mixed2). The steep drops of LCS curves separate areas where the length of the LCS remains practically unchanged, even for significantly different numbers of considered accounts. In the left hand plot of Fig. 4, for instance, the LCS remains almost unchanged when considering a number of accounts between 500 and 1000. The same also applies to the right hand plot of Fig. 4, for a number of accounts lower than 400. Such *plateaux* in LCS curves are strictly related to homogeneous groups of highly similar accounts. Note that it is possible to observe multiple plateaux in a single LCS curve, as in the case of Fig. 4(a). This represents a situation where multiple (sub-)groups exist among the whole set of considered accounts. Furthermore, the steeper and the more pronounced is a drop in a LCS curve, the more different are the two subgroups of accounts split by that drop.

To summarize, LCS curves of an unknown and heterogeneous group of users can present one or more *plateaux*, which are related to subgroups of homogeneous (i.e., with highly similar behaviors) users. Conversely, *steep drops* represent points marking big differences between distinct subgroups. Finally, *slow and gradual decreases* in LCS curves represent areas of uncertainty, where it might be difficult to make strong hypotheses about the characteristics of the underlying accounts. We can than conclude that LCS curves of an unknown and heterogeneous group of users are capable of conveying information about relevant and homogeneous subgroups of highly similar users.

The previous findings can be leveraged to find subgroups of users with similar behaviors [10]. We exploit the discrete derivative of a LCS curve to recognize the points corresponding to the steep drops. The steep drops of LCS curves appear as sharp peaks in the derivative plot and represent suitable splitting points to isolate different subgroups among the whole set of users. All the suitable splitting points might be ranked according to their corresponding derivative value (i.e., how steep is the corresponding drop) and then, a hierarchical top-down (i.e., divisive) clustering approach may be applied, by repeatedly dividing the whole set of users based on the ranked points, leading to a convenient dendrogram structure. For instance, this approach can be exploited in situations where the LCS curve exhibits multiple plateaux and steep drops, in order to find the best possible clusters that can be used to divide the original set of heterogeneous users.

The discrete derivative of the LCS curve of a set of M users can be easily computed as

$$\mathrm{LCS}'[k] = \frac{\Delta_k\,\mathrm{LCS}}{\Delta_k\,\mathrm{accounts}} = \frac{\mathrm{LCS}[k] - \mathrm{LCS}[k-1]}{1}$$

for $k = 3, \ldots, M$. Given that LCS curves are monotonic non-increasing functions defined over the $[2, M]$ range, their derivatives LCS′ will assume only zero or negative values, with steep drops in the LCS corresponding to sharp negative peaks in LCS′. Simple peak-detection algorithms can be employed in order to automatically detect the relevant peaks in LCS′ [19]. Notably, this approach does not require a training phase and can be employed pretty much like a clustering algorithm, in an unsupervised fashion.

To prove the effectiveness of this unsupervised approach, we applied it to the LCS obtained from the unlabeled `Mixed1` and `Mixed2` groups, with the goal of separating spambots from genuine users. Fig. 5(a) and (b) show stacked plots of the LCS of the `Mixed1` and `Mixed2` groups respectively, together with their discrete derivative LCS′, both in linear and logarithmic scale. The logarithmic scale plots of the derivatives have been computed as $\log_{10}|\mathrm{LCS}'|$ and they have been added for the sake of clarity, since they highlight the less visible peaks of the linear scale plots. In order to facilitate the detection of peaks in LCS′, we smoothed the original LCS curves before computing their derivatives [20]. This preprocessing step acts pretty much like a low-pass filter, allowing to flatten the majority of noisy fluctuations.

In Fig. 5, the solid vertical blue lines correspond to the most pronounced peaks in the LCS′ of `Mixed1` and `Mixed2`. As shown, the proposed methodology accurately identified reasonable splitting points in order to find two clusters among the whole sets of unlabeled users. In detail, those users laying on the left of the vertical splitting line – that is, users sharing long behavioral patterns (i.e., long LCS) – are labeled as spambots. Conversely, the users to the right of the vertical splitting line – i.e., users sharing little similarities – are labeled as genuine ones.

2.2. Genetic algorithms

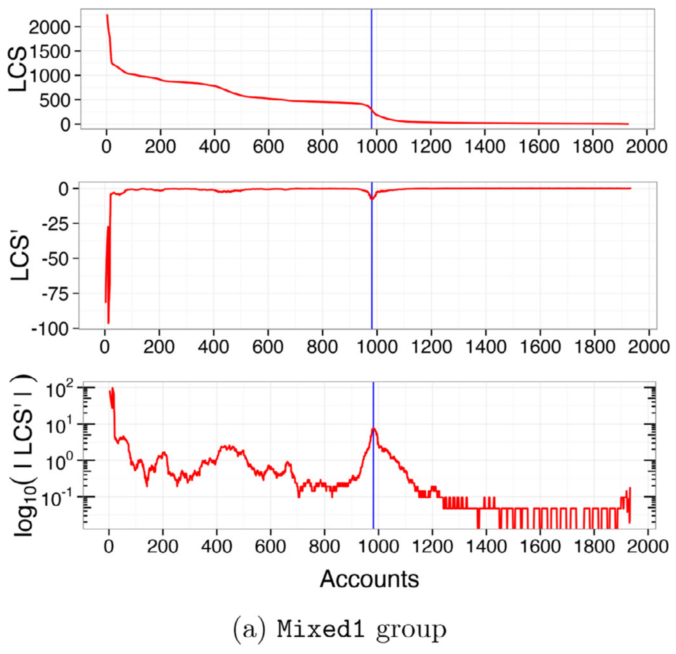
Genetic Algorithms [11] are a popular meta-heuristic technique, used to solve optimization problems inspired by the natural evolution, where only the best candidates survive along several generations. In this scenario, a possible candidate solution is called *individual* and a sample of candidate solutions is called *population* of individuals. The idea is to start from an arbitrary, or random, initial population and, then, similarly to what happens in nature, through a mechanism of recombination and mutation of those individuals a new *generation* of candidates is obtained, which hopefully improves the quality of the previous solutions. To establish the quality of a candidate solution, each individual is associated with a measure called *fitness score*. The cornerstone of most genetic algorithms is the *fitness function*, that outputs the fitness score of an individual given as the input. During each generation, each individual is evaluated using the fitness function and then the current population is modified via either a single input function called *mutation* or a two input function called *crossover*. The outputs of these functions are called *offsprings* and these are the new candidate solutions that will form the new generation along with some of the individuals of the previous generation. In this work, when we refer to an individual, we refer to a group of users and our population is composed by a set of groups of users that evolve from a generation to the next one. We will present a variety of options that can be used in order to provide a solid simulation technique that can be usable in this concrete case study.

2.2.1. Encoding

The first ingredient of a Genetic Algorithm is the *encoding*. Each individual is encoded into a *genome* which is commonly either a binary vector or a string of characters. For the scope of this study, we rely on the DNA-based behavioral modeling presented in Section 2.1 to encode candidate solutions directly into the genomes that will be used during the evolution of the genetic algorithm. As we mentioned earlier, in this case study a candidate solution is a group of users, each user is encoded with a string of length n, where n is the length of the activities that we consider (i.e., the number of the user actions on Twitter). Each group of users will be formed by m users, therefore, the genome used in the genetic algorithm will be a matrix $m \times n$ of characters (as each activity will be labeled with a character). This allows us to reason within a two-level environment: the user level and the group level. As we will detail in the following sections, the mutations and crossovers can happen at both levels. For the entirety of this work we use exclusively this encoding.

2.2.2. Mutation

The *Mutation Function* is a unary operator that takes in input an individual and returns an offspring. In this work, we develop a mutation function that acts at user level. Each activity of each user can be subject to mutation, more specifically, the operator switches the selected activity with another activity. For instance, a tweet can be switched to a retweet or a mention, a mention can be switched to a retweet or a tweet and a retweet can be switched to a tweet or a mention. Not all activities will be influenced by mutations, as there is a tunable parameter p_m that helps us to control the rate



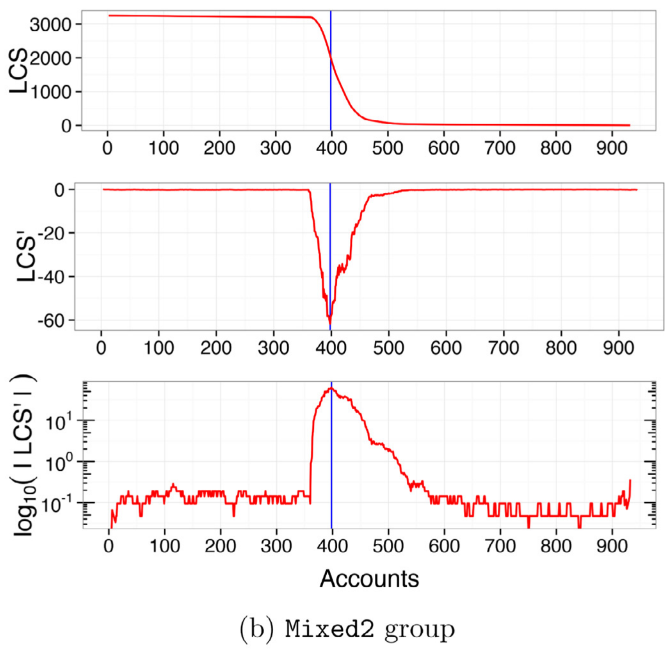


Fig. 5. Discriminating between spambots and genuine users among an unknown set of accounts using the LCS derivative. The curve peaks that represent the best candidate points for the split are marked in all graphs with a vertical blue line. Accounts to the left of the splitting points are identified as spambots, while those to the right are identified as genuine users. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

of mutations throughout the run of the genetic algorithm. Parameter $0 \leq p_m \leq 1$ denotes the *probability* of each activity to be subject to a mutation. Different choices of this parameter lead to different results: by setting $p_m = 0$, the mutation operator is completely turned off, while $p_m = 1$ has the effect of switching all the activities of all the users. If p_m is close to zero, then the offspring will be very similar to the individual in input; if p_m is close to one, then the returned offspring will be very different from the individual in input. This parameter offers a lot of flexibility on how to explore the solution space. We will show in Section 4 how the choice of this parameter impacts the overall output of the algorithm.

2.2.3. Crossover

In a traditional genetic algorithm, the crossover function is a binary operator that takes in input two individuals and usually returns two offsprings. The choice of the crossover function is up to the developer of the algorithm and, as for the mutations, there are a list of flexible options. There are some popular crossover functions, such as *one-point crossover* and *two-points crossover*, that can be efficiently applied to many problems, but, in general, developers can design and implement their own crossover function, based on the demands of the problem they are tackling. For the case study under investigation, we exploit the two-level encoding that we have chosen and we design two crossover functions that act at user level and one crossover function that acts at group level. The main crossover technique we use throughout this work is derived from the standard one-point crossover strategy. This type of crossover has two parent genomes as the input and two offsprings as the output. Let the reader assume that the genomes have length n, one random *crossover point* r is chosen between 1 and n. The first offspring genome generated is the copy of the first parent from 1 to the crossover point r and the remaining part from $r+1$ to n is the copy of the remaining genome of the second parent. The second offspring is generated in a similar manner, as shown in Fig. 6. In the most common scenarios where genetic algorithms are used, the genome is one-dimensional. In this work, instead, each individual is bound to a two-dimensional matrix and, therefore, we can exploit this encoding to develop an efficient two-level

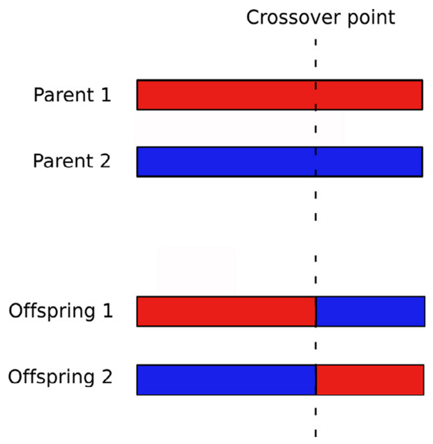


Fig. 6. Standard one-point crossover.

non-standard crossover strategy. Let us recall that each individual represents a group of users and it is encoded in a matrix of $m \times n$ characters where the number of users in each group is m and the number of activities per users is n. Because of the fact that each individual is a group of users, if we take a traditional path, the elementary unit of the genome would be an individual. The *Group Level Crossover* is designed via a standard one-point crossover scheme in which the parents are individuals and therefore, in this case, the parents and offspring will be groups of users. Given two groups of m users, a random crossover point r is chosen and the two groups are split with respect to point r and, then, mixed as shown in Fig. 7. Acting this way, the goal is to provide ever changing groups of users that evolve by trading subgroups of users from one group to the other. This technique allows us to obtain groups that are more diversified and more heterogeneous and, therefore, intuitively, more human-like. However, by applying this single crossover function, we would operate by trading users only. As of now, in order to diversify the users' activities, we can only rely on the mutation function. Let us dive one level deeper:

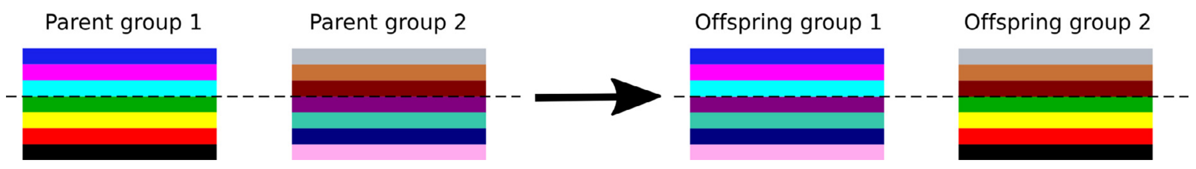


Fig. 7. Grouplevel crossover.

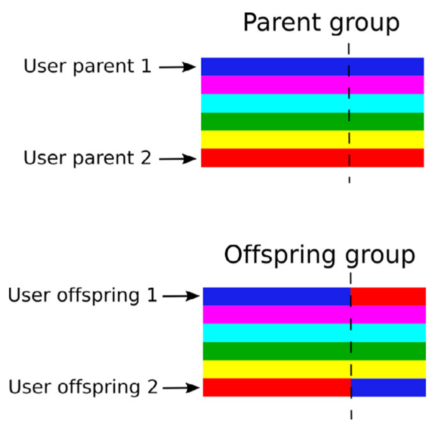


Fig. 8. User level crossover.

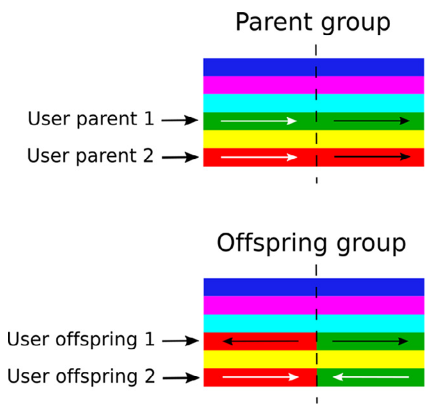


Fig. 9. User level crossover with reverse string.

ideally, we would like a crossover function that acts at the level of the activities of the users rather than of the groups. For this reason, we opt to design two more crossover functions that act at the level of each user's activity. Within each group, each user is represented with a one dimensional vector of size n, the idea is to develop a crossover function that can take two users as the input and generate two offsprings that combine the activities of the parents, thus providing a more diversified set of activities within each group. We present two one-point crossover functions that act at user level. The first, namely *User Level Crossover* is similar to the standard one-point crossover and it is presented in Fig. 8. This is a pretty straightforward implementation of a standard one point crossover: two users are chosen as parents and the offsprings are the recombination of the two, in a traditional fashion. For instance, let the reader assume to have a user $A = a_1a_2a_3a_4$ and a user $B = b_1b_2b_3b_4$; setting 2 as the crossover point, the two offsprings are the following users $O_1 = a_1a_2b_3b_4$ and $O_2 = b_1b_2a_3a_4$. The second, namely *User Reverse Level Crossover*, it is slightly more convoluted. The idea is that, in order obtain a more diversified behavior of the users, we choose a crossover point and we do the crossover operation in a similar manner as we did previously, with the difference that, before attaching the strings of activities together, we reverse one of the strings. With two users, $A = a_1a_2a_3a_4$ and $B = b_1b_2b_3b_4$, with 2 as the crossover point, the two offsprings are the following users $O_1 = b_4b_3a_3a_4$ and $O_2 = b_1b_2a_2a_1$. Proved by the experimentation described in the next sections, adding the user reverse level crossover lead to individuals of higher quality. A visual summary of how this crossover function works is provided in Fig. 9. Similarly to what happens with the mutation operator, we can tune some parameters. Only a subset of individuals and users is subject to the crossover operations and, therefore, we can define some parameters to control these subsets. For the group level crossover, we define N_{GCO} as the number of offsprings to generate. Choosing $N_{GCO} = 0$, we avoid this operator, while, with $N_{GCO} = n$, every time that the group level crossover operates, a new population of n offsprings is generated. Similarly, we define M_{UCO} as the number of offsprings we want to generate with the user level crossover operation. Once again, if we set $M_{UCO} = 0$, this means that we avoid this operator, while, with $M_{UCO} = m$, each time we apply this operator we obtain a set of user offsprings that is as big as the original group of users. We define the parameter that controls the user reverse level crossover, namely M_{URCO}, in the same manner of M_{UCO}. Armed with these parameters, we can tune how the search space is explored using the crossover operators. Different choices of these parameters lead to different results, as we will show in Section 3.

2.2.4. Fitness function

As mentioned before, the fitness function serves as a tool to provide the quality score of each individual. Thus, it comes at no surprise that it plays an important role in every genetic algorithm, for two main reasons. First of all, the fitness score is calculated for *every* individual of *every* generation and, therefore, the overall running time of a genetic algorithm strongly depends on how fast the fitness functions can be computed. Secondly, the goal of every optimization algorithm is either to minimize or maximize an objective function. The quality of a candidate has to be measured with respect to the objective function. In some cases, the objective function and the fitness function can coincide and therefore an individual with a good fitness score is a good solution for the original problem. This is not always the case, as sometimes computing the objective function might be too computationally expensive. In general, there is a trade-off between the time spent to compute the fitness scores and their accuracy. In some cases, such as the task at hand in this paper, even defining the goal or what the objective function should be is not straightforward. Here, the goal is to generate a group of sequences that resemble the behavior of a group of genuine accounts. Regarding how to measure such resemblance, we present a couple of possible fitness functions that can be used in this setting and we will show that a different choice of the fitness function provides substantially different outcomes. In our scenario, we have a target discrete LCS curve X and we want to generate a new discrete LCS curve Y that resembles X as much as possible. To evaluate the distance between the two curves, a first candidate can be the maximum distance between X and Y.

$$max(X, Y) = max\{|X(i) - Y(i)| \; : \; i \in \{1, \ldots, |X|\}\} \tag{1}$$

By minimizing (1), the two curves would be very similar to each other. Note that, in the extreme case, in which $max(X, Y) = 0$, the two curves would coincide. This fitness function seems to be promising, as it is computable in a very fast manner and it is, in some sense, a measure of similarity. We will show in Section 3 that this is not the case. In order to provide a better solution, a more sophisticated fitness function is necessary: ideally, a fitness function that accounts for the entirety of the curve rather than just the point of maximum distance is a better and suitable candidate. The distance between two LCS curves can be computed with any well-known metric of distance between functions or statistical distributions. We rely on the Kullback-Liebler distance (D_{KL}), since it has already been fruitfully employed in recent similar work. D_{KL} is an information theoretic metric that measures how much information is lost when a target probability distribution $P_X(x)$ is approximated by $\hat{P}_X(x)$. In detail, D_{KL} is the symmetric version of the Kullback-Liebler divergence d_{KL} defined as,

$$d_{KL}(\hat{P}_X, P_X) = \sum_x \ln\left(\frac{\hat{P}_X(x)}{P_X(x)}\right)\hat{P}_X(x)$$

thus,

$$D_{KL}(\hat{P}_X, P_X) = \frac{d_{KL}(P_X, \hat{P}_X) + d_{KL}(\hat{P}_X, P_X)}{2} \tag{2}$$

In this scenario, the target distribution $P_X(x)$ is obtained from the LCS curve of legitimate accounts, while the approximating distribution $\hat{P}_X(x)$ is obtained from the LCS curve of a group of bots. The D_{KL} distance is computed by the fitness function that accepts as input an individual G_j and the target LCS curve b of legitimate accounts. The output is a scalar that represents the fitness score of the individual. With a fitness function like D_{KL} we account for the distance between each point of the curves and, therefore, we can have a more accurate measure of similarity, if compared with the maximum distance. As discussed earlier, we will obtain a more accurate measure, while however spending more time to compute the function.

3. Experiments

In this section, we set up the second phase of the proposed proactive schema, namely the *simulation*. In the simulation phase, we will rely on different parameters, to be able to evaluate as many evolutions as possible.

3.1. The algorithm

The genetic algorithm that we use in order to generate new sequences of digital DNA combines the features described in Sections 2.2.1, 2.2.2, 2.2.3 and 2.2.4. The input of the genetic algorithm is a group of genuine sequences that we want to emulate: we will refer to this group as the *target group*. The objective of the genetic algorithm is to minimize the distance between the LCS curve of the generated sequences and the LCS curve of the target group. The parameters that control the mutation and crossover probabilities will be discussed in Section 3.2. During each iteration of the algorithm, we apply the mutation and crossover operators in order to obtain the next generation of individuals, a graphical overview of the work involved in one iteration is presented in Fig. 10. The output of the algorithm is the population after the last generation, i.e., a set of groups of synthesized user sequences.

3.2. Experimental setup

All the experiments run on a machine with an Intel Xeon E7-4830v4, with a 64-bits architecture at 2 GHz, 112 cores and 500 GB of RAM. For an efficient, linear-time computation of the LCS curves, we rely on an adapted version of the GLCR toolkit[4] implementing the state-of-the-art algorithms described in [15]. The following parameters are the same for all the experiments: each group consists of 100 users, each user features 2000 activities and each run features 30 individuals. Each run stops after 1000 generations.

3.3. The parameter space

Here we give an overview of the possible parameters that can be chosen to tune the algorithm. Each run of the algorithm starts with an initial population which can be arbitrary or randomly chosen. As a baseline, after a preliminary experimental evaluation, we use a starting population in which each user has 1000 *A*, 500 *C* and 500 *T*. We can explore the effects of starting from a different point of the search space. One idea is to start with plain activities for all users of all groups. For instance, we can start with a group of users in which all the activities of all the users are *A*. Respectively, we analyze what is the result after initializing all the users' activities with *C* and *T*. Table 2 presents an overview of the parameters and their ranges.

3.4. The experiments

Here, we present different strategies to implement possible evolutionary branches. To evaluate possible evolutions of spambots starting from scratch, we design and launch many experiments. With different evolutions at disposition, it will be possible to identify which ones can be a threat and which ones can still be detected by state-of-the-art detection techniques. In the proactive schema presented in Fig. 1(b), step 2 (simulation) and step 3 (evaluation) are strongly connected, as the evaluation step provides the feedback on the proposed evolutions generated via the simulation. In order to illustrate how these two steps interact between each other, we show a variety of possible evolutions. These candidate evolutions are generated via the genetic algorithm with different choices of the parameters in Table 2. Each run of the genetic algorithm generates 30 different groups of new evolved spambots. The detection algorithm will be then applied in the next section, to evaluate whether the evolved spambots can still be detected or not. In Table 3 we present a summary of the evolutions that we will evaluate. In order to give an experimental validation of the choice of D_{KL} as a fitness function, we show a comparison between a run of the genetic algorithm with the best known choice of parameters using D_{KL} as a fitness function against a run of the genetic algorithm with the same parameters using *MAX* as the fitness function. We present an extensive evaluation of the effect of the mutation function: we show the results of 8 different choices of mutation probability. We start with the mutation disabled and we finish with mutation probability 1 (i.e., all the activities off all the users are mutated). We present an evaluation of the extreme values of the crossover operators: we show what happens when we disable the user crossover operator and when we apply the crossover operator to all the possible users and groups.

4. Evaluation

In this section, we discuss the important role covered by the evaluation phase. As mentioned in the previous section, the simulation and evaluation phases constantly interact between each other: once we obtain some evolved candidates from the simulation phase, we need to check if they can be a threat for the particular model we are studying. Thus, we evaluate the candidate evolutions against the unsupervised detection technique [10] that has been presented in Section 2.1.3.

[4] https://www.uni-ulm.de/in/theo/research/seqana.html

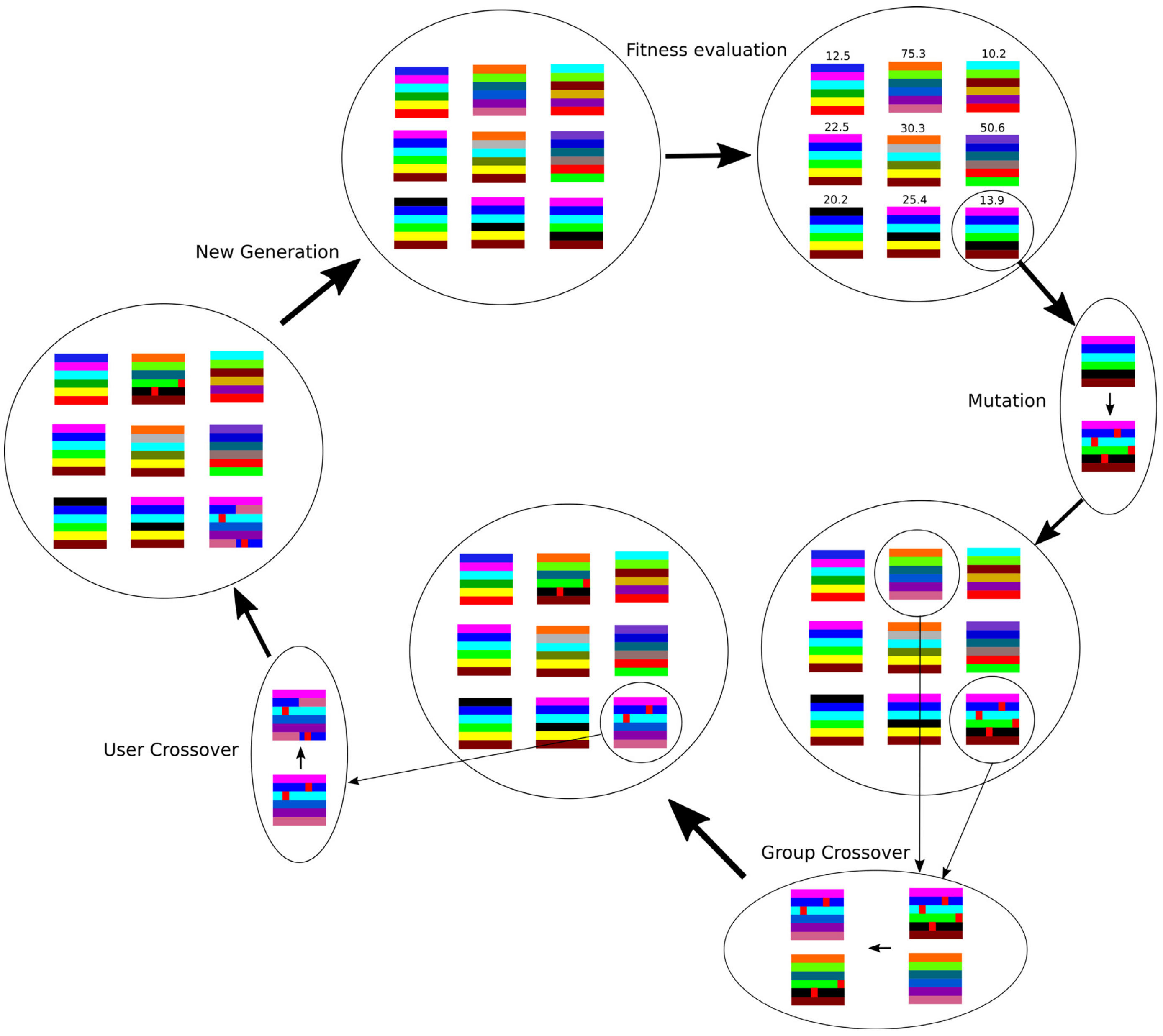


Fig. 10. An overview of all the actions taken by the algorithm during a generation.

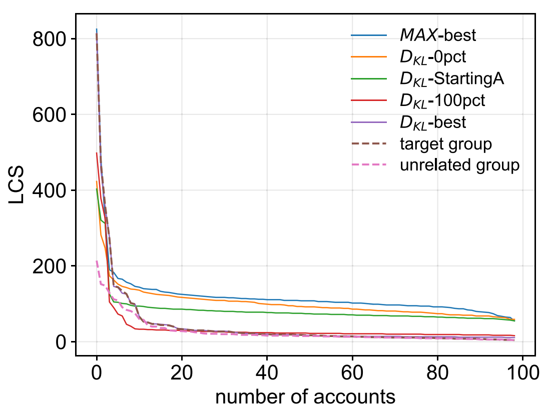
Table 2
Parameters overview.

Parameter	Range	Description
p_m	[0, 1]	Mutation probability
N_{GCO}	$\{0, \ldots, n\}$	Offsprings generated by the Group Crossover
M_{UCO}	$\{0, \ldots, m\}$	Offsprings generated by the User Crossover
M_{URCO}	$\{0, \ldots, m\}$	Offsprings generated by the User Reverse Crossover
Fitness function	$\{max, D_{KL}\}$	The choice of fitness function presented in Section 2

In order to evaluate the evasion ability of the newly generated candidates, we test them against the detection algorithm. In particular, for each best individual of a run of the genetic algorithm, we create two mixed groups of 200 users each. The first group (the *synthes + target group*) is obtained combining the best individual (composed by 100 synthesized users) with the 100 users that were used as the target group by the genetic algorithm (3.1). The second group (the *synthes + unrelated group*) is composed by the best individual and a group of 100 real Twitter users, unrelated to the experiments. The rationale is to verify that the synthesized sequences can evade the unsupervised detection technique even if they are injected in a group of humans that was never considered by the genetic algorithm.

For each of the candidate evolutions proposed in Section 3.4, we perform an evaluation using the following schema:

1. We fix the parameters of the genetic algorithm;
2. We run the genetic algorithm to generate 30 new individuals (i.e., 30 groups of 100 users);
3. We take the best individual, according to the fitness score;



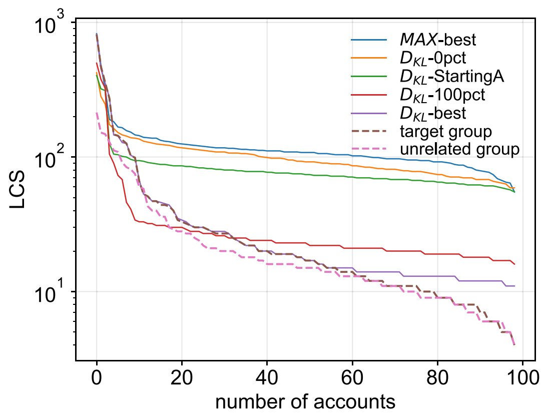


Fig. 11. Comparison of LCS curves.

Table 3
Summary of the candidate evolutions (when not specified, the initial population is composed by 1000 *A*, 500 *C*, 500 *T*).

Instance code	Description
D_{KL}-StartingA	D_{KL} fitness, all the activities in the initial population are *A*
D_{KL}-StartingC	D_{KL} fitness, all the activities in the initial population are *C*
D_{KL}-StartingT	D_{KL} fitness, all the activities in the initial population are *T*
D_{KL}-noOnePt	D_{KL} fitness, the User Crossover is disabled
D_{KL}-allCx	D_{KL} fitness, all the crossovers are set to their *max*
D_{KL}-0pct	D_{KL} fitness, the mutation is disabled
D_{KL}-5pct	D_{KL} fitness, 5% mutation probability
D_{KL}-10pct	D_{KL} fitness, 10% mutation probability
D_{KL}-25pct	D_{KL} fitness, 25% mutation probability
D_{KL}-50pct	D_{KL} fitness, 50% mutation probability
D_{KL}-75pct	D_{KL} fitness, 75% mutation probability
D_{KL}-90pct	D_{KL} fitness, 90% mutation probability
D_{KL}-100pct	D_{KL} fitness, 100% mutation probability
D_{KL}-best	D_{KL} fitness, best known choice of parameters
MAX-best	*MAX* fitness, best known choice of parameters

4. We create two mixed groups of 200 users each:
   - the synthes + target group, composed by the best individual mixed with 100 users from the target group of genuine users;
   - the synthes + unrelated group, composed by the best individual mixed with 100 genuine users unrelated to the run of the genetic algorithm;
5. We run the unsupervised detection technique on the mixed groups.

4.1. *Detection results of the synthes + target group with the derivative heuristic*

The results of the detection against the synthes + target groups are in Table 4. For each choice of parameters, we also report the fitness score of the best individual, obtained by the last generation of the genetic algorithm. Each row is organized in the following manner: in the first column, there is the instance code based on the ones presented in Table 3, for example, the code D_{KL}-StartingA represents the best individual generated using all *A* as a starting point and D_{KL} as the fitness function. The second column (*Cut*) is the cutting point identified by the unsupervised detection technique for the related instance. This determines the performance of the detection results, presented in the subsequent columns, that report *Precision, Recall, Specificity* and *Accuracy*. The second to last column (*F1*) reports the value of *F-measure*, a metric that summarizes in one single value the other results. Finally, the last column presents the fitness score based on the D_{KL} fitness function of the individual. As a comparison, we also include in the table the best individual that we ever generated using the genetic algorithm, represented by the row D_{KL}-best. Moreover, Fig. 11 shows the LCS curves of some instances, to have an overview of the synthesized users. We report the LCS in both the linear y-axis and in the log y-axis. In the figures, we see that the instance with the best fitness score (D_{KL}-best) overlaps the LCS curve of the target group, while the instances that have a higher (i.e., worse) fitness score do not resemble the LCS of the target group. Moreover, the best instance starts to significantly differ from the target group only when the number of accounts is higher than 60: the delta between the two LCS is always lower than 10. Similarly, we observe that the LCS of the unrelated group and that of the target group are very similar, with significant differences only in the very first points and negligible differences when the number of accounts is higher than 50.

In the following we give some insights into the results of the experiment, looking at Table 4.

Looking at the last column of the *MAX* instance, if we consider the fitness score using the D_{KL} on the LCS of that instance, we notice that it has a very high fitness value and, in facts, it performs much worse when compared to the instances of the other runs. The poor quality of this individual resides in the fact that most of the synthesized users do not elude the unsupervised detection technique: it is worth reminding that, if the detection is effective, our synthesized users are not good bot candidates.

This first observation suggests that, to have better candidates, we have to use a better fitness function, leading us to adopt the D_{KL}. However, a better fitness function is still not enough to have good candidates: if we consider the results of the instances obtained starting from naive and simple starting populations (such as the ones presented in the instances D_{KL}-StartingA, D_{KL}-StartingC and D_{KL}-StartingT) we can see that, despite having a better fitness score, these candidates can not still evade the detection technique efficiently.

These results confirm that designing experiments that can elude and highlight the weaknesses of a given detection technique is not a simple task. They also suggest that all the ingredients that we presented in Section 3.1 are necessary in order to generate effective candidates. From the table, we can see that the results tend to get better once we start tuning the crossover and mutation operators, starting from the baseline starting population described in Section 3.3. In fact, we can observe that the candidates D_{KL}-allCx and D_{KL}-noOnePt, that only perform the crossover, improve their evasion against the detection technique and, similarly, increasing the mutation probability (the D_{KL}-XXpct instances) also effectively affects the evasion ability of the synthesized accounts.

At a first glance, we can say that all those instances can be a potential threat because from the accuracy results and F-measure of the detection algorithm, most of these generated bots go unde-

Table 4
Summary of the detection results vs. *synthes + target* group.

Instance code	Cut	Precision	Recall	Specificity	Accuracy	F1	D_{KL}
MAX-best	66	0.848	0.56	0.90	0.730	0.675	6887
D_{KL}-StartingA	40	0.900	0.36	0.96	0.660	0.514	3866
D_{KL}-StartingC	23	0.739	0.17	0.94	0.550	0.276	975
D_{KL}-StartingT	21	0.667	0.14	0.93	0.535	0.231	709
D_{KL}-allCx	27	0.296	0.08	0.81	0.445	0.126	661
D_{KL}-noOnePt	40	0.475	0.19	0.79	0.490	0.271	455
D_{KL}-0pct	56	0.804	0.45	0.89	0.670	0.577	546
D_{KL}-5pct	30	0.400	0.12	0.82	0.470	0.185	336
D_{KL}-10pct	30	0.333	0.10	0.80	0.450	0.154	420
D_{KL}-25pct	33	0.394	0.13	0.80	0.465	0.195	418
D_{KL}-50pct	34	0.382	0.13	0.79	0.460	0.194	329
D_{KL}-75pct	25	0.200	0.05	0.80	0.425	0.080	409
D_{KL}-90pct	26	0.154	0.04	0.78	0.410	0.063	517
D_{KL}-100pct	24	0.042	0.01	0.77	0.390	0.016	501
D_{KL}-best	**41**	**0.512**	**0.21**	**0.80**	**0.505**	**0.298**	**33**

Table 5
Summary of the detection results vs *synthes + unrelated* group.

Instance code	Cut	Precision	Recall	Specificity	Accuracy	F1
MAX-best	43	0.814	0.35	0.92	0.635	0.490
D_{KL}-StartingA	42	0.881	0.37	0.95	0.660	0.521
D_{KL}-StartingC	23	0.739	0.17	0.94	0.550	0.276
D_{KL}-StartingT	21	0.667	0.14	0.93	0.535	0.231
D_{KL}-allCx	27	0.296	0.08	0.81	0.445	0.126
D_{KL}-noOnePt	40	0.475	0.19	0.79	0.490	0.271
D_{KL}-0pct	56	0.804	0.45	0.89	0.670	0.577
D_{KL}-5pct	30	0.400	0.12	0.82	0.470	0.185
D_{KL}-10pct	30	0.333	0.10	0.80	0.450	0.154
D_{KL}-25pct	33	0.394	0.13	0.80	0.465	0.195
D_{KL}-50pct	34	0.382	0.13	0.79	0.460	0.194
D_{KL}-75pct	25	0.200	0.05	0.80	0.425	0.080
D_{KL}-90pct	26	0.154	0.04	0.78	0.410	0.063
D_{KL}-100pct	24	0.042	0.01	0.77	0.390	0.016
D_{KL}-best	**37**	**0.514**	**0.19**	**0.82**	**0.505**	**0.277**

tected. Now, it is important to stress the fact that the results of the detection are presented with respect to the choice of a cutting point which is calculated by a heuristic. Therefore, the cutting point might not be the optimal one.

This observation lead us to better explore the evasion ability of the obtained instances, considering all the possible cutting points, since, looking at the results of Table 4, it appears that groups that have a higher fitness score than D_{KL}-best seem to elude the detection technique better. For instance, if we consider the accuracy of the algorithm against the best individual ever generated, we can see that it is 0.505 while, on the other hand, the accuracy of the detection for the instance D_{KL}-100pct is 0.390. This would mean that the latter instance can elude the detection algorithm better than the best solution, although, if we consider the fitness score, we can see that it is much higher and, therefore, the quality of that solution is worse. In any case, this implies that we have spotted one potential weakness in the detection algorithm, proving another strength of the proposed proactive schema: the heuristic that is used to decide in which point to cut is inefficient against groups of bots synthesized via the genetic algorithms.

4.2. Detection results of the synthes + unrelated group with the derivative heuristic

The detection results against the synthes + unrelated group are summarized in Table 5. This experiment aims at verifying if the genetic algorithms proposed by the proactive scheme depend on the choice of the target group. This would mean that we can generate groups of users that emulate particularly well only the target group, but might be unable to evade the detection when mixed with other unrelated users. However, this is not the case, as the results presented in Table 5 show. First of all, we can notice that the detection results in Table 5 are very similar to (and sometimes they are exactly the same of) the results of Table 4. As in the previous section, we can observe that *MAX* as fitness function produces the worst candidates. Then, we notice that crossover operations do not suffice to evade the detection and that permutations and crossovers are the right combination to have effective bots. The similarity of results makes us to conclude that the proactive scheme via genetic algorithms is not bound to the users taken as target group for the fitness function, since the detection technique fails to spot some of the evolved bots when they are mixed to a group of genuine accounts that was never observed by the algorithm.

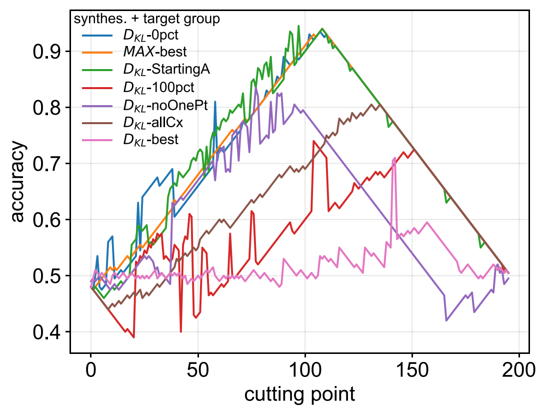
4.3. Detection results for any cutting point

In this section, we want to dive deeper into the analysis of the generated users, trying to address the following questions: if we imagine to have the best detection heuristic possible, how will our bots perform? Said in another way, what does it happen if we try all the possible cutting points? Can we still spot the evolved bots, if we choose the optimal cutting point?

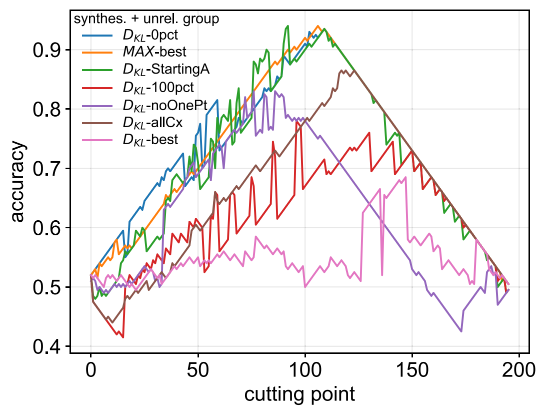
To provide an answer to the above questions, we evaluate the detection results for all the possible cutting points and we report the results in Fig. 12–14.

In Fig. 12, we compare the accuracy of different cutting points on the mixed groups of both synthesized and target accounts and the synthesized and unrelated accounts. For instance, if we use 110 as the cutting point, we can see from Fig. 12(a) that the detection algorithm has a 0.5 accuracy against the best generated group. At that cutting point, the accuracy is around 0.7 against the group generated with 100% of mutation, while the other instances perform worse than 0.9. If we do the same analysis using the unrelated group, in Fig. 12(b) we can see once again that the best instance (D_{KL}-best) goes undetected, independently from the cutting point, while the other instances with worse fitness are not as robust with respect to different choices of cutting points.

The same holds for all the other metrics: precision and recall in Fig. 13 and F-measure and specificity of Fig. 14. For example, if we observe the F-measure, we can see that the individual with the best fitness score outperforms the others, independently of the

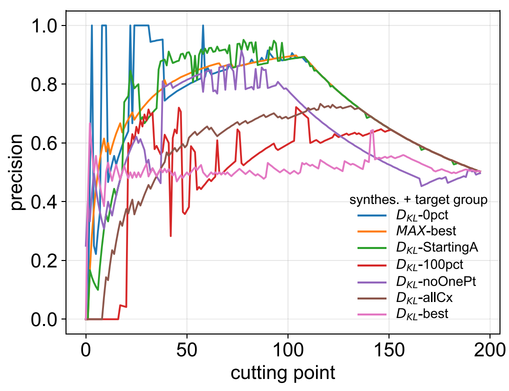


(a) Accuracy, target group

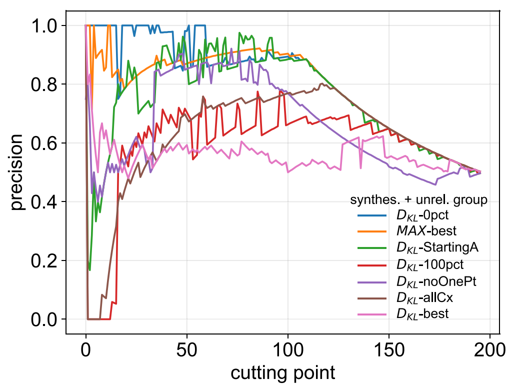


(b) Accuracy, unrelated group

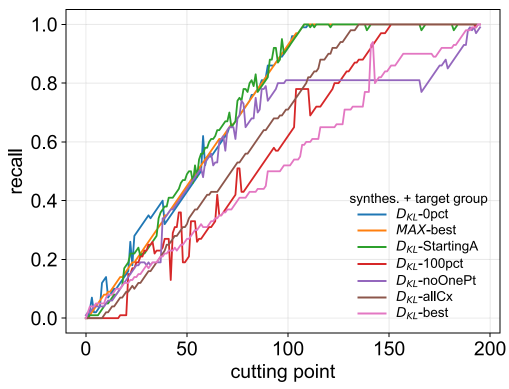
Fig. 12. Detection results for all the possible cutting points. Accuracy of *synthes + target* group (on the left) and *synthes + unrelated* group (on the right).



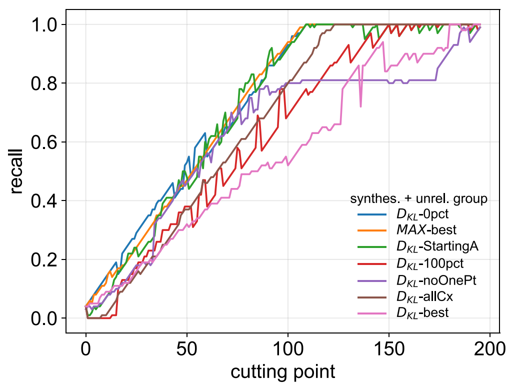
(a) Precision, synthes + target



(b) Precision, synthes + unrelated



(c) Recall, synthes + target



(d) Recall, synthes + unrelated

Fig. 13. Detection results for all the possible cutting points. Precision and recall of *synthes + target* group (on the left) and *synthes + unrelated* group (on the right).

choice of the cutting point. The instances that perform well against the heuristic cut based on the derivative perform poorly against other cutting points.

Again, if we compare the synthes + target group results (Fig. 14(c)) with those of the synthes + unrelated group (Fig. 14(d)), we see that the choice of different cutting points can expose some of the evolved bots generated by the weaker instances.

In conclusion, we empirically proved that, in order to generate some possible candidate evolutions, many strategies have to be evaluated and many parameters have to be tested, before highlighting possible weaknesses of the detection algorithm.

5. Discussion

In this section, we clarify why the adopted genetic algorithm is capable of generating evolved spambots whose behaviors are both malicious and undetectable, thus attractive for spambots developers.

In the following, we compare some behavioral characteristics of the spambots in our datasets with those of the evolved bots. We remind the reader that accounts in `Bot1` dataset are retweeters of a political candidate, while the accounts in `Bot2` dataset are URL spammers.

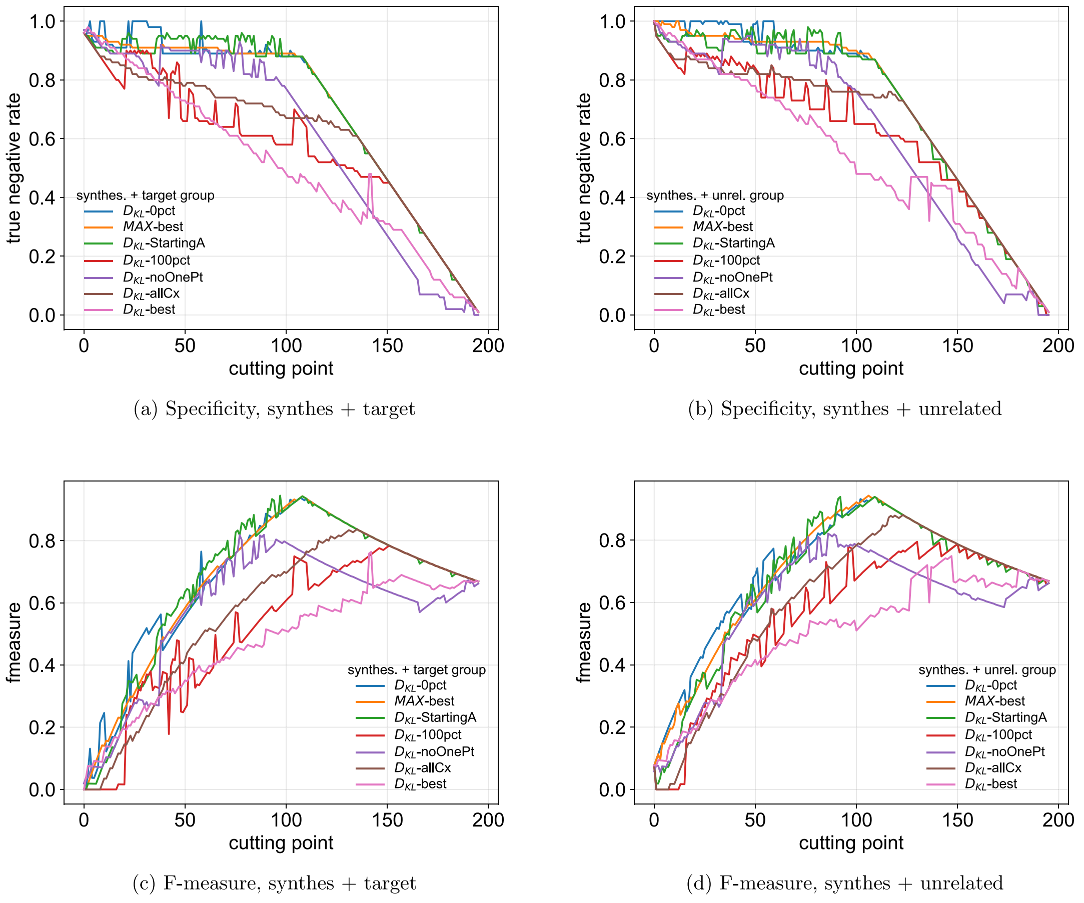


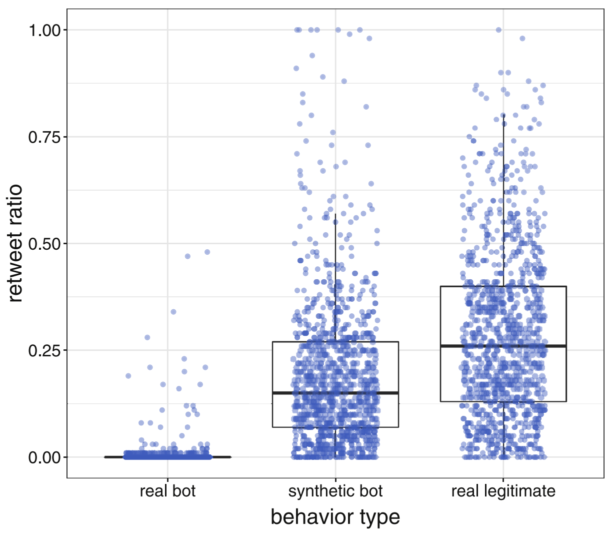
Fig. 14. Detection results for all the possible cutting points. Specificity and F-measure of *synthes + target* group (on the left) and *synthes + unrelated* group (on the right).

Fig. 15(a) shows the statistical distribution of the *retweet ratio* for the accounts of the Bot1 dataset, before (labeled *real bot* in figure) and after (labeled *synthetic bot*) the application of the genetic algorithm. For the sake of completeness, we report the same distribution also for the humans dataset (labeled *real legitimate*). The retweet ratio is computed as the fraction of retweets over all tweets produced by an account. Each point in the figure represents a distinct account. We can easily see that despite being retweeters of a political candidate, the accounts in Bot1 overall retweet only a minority of the time and thus have a rather low retweet ratio. The consequence of the application of the genetic algorithm to the accounts in Bot1 is that the evolved spambots retweet more often than the original ones. This is because the algorithm aimed to make the new behavior of the bots more similar to that of the humans, which indeed feature a median $\frac{\#retweets}{\#tweets}$ ratio in the region of 0.25. Thus, the number of retweets done by the evolved bots is greater than that done by the original bots. Moreover, whereas the original bots are detected by the Social Fingerprinting technique (see Fig. 5(a), Section 2.1.3), their evolution is such that the performances of the technique is very poor (Tables 4 and 5). Indeed, the evolved retweeters are not detectable anymore, despite seeing their chances of retweeting increase. To this regard, the evolved behavior obtained with the application of our genetic algorithm, made the bots both more effective in their retweeting strategy and less easily detectable.

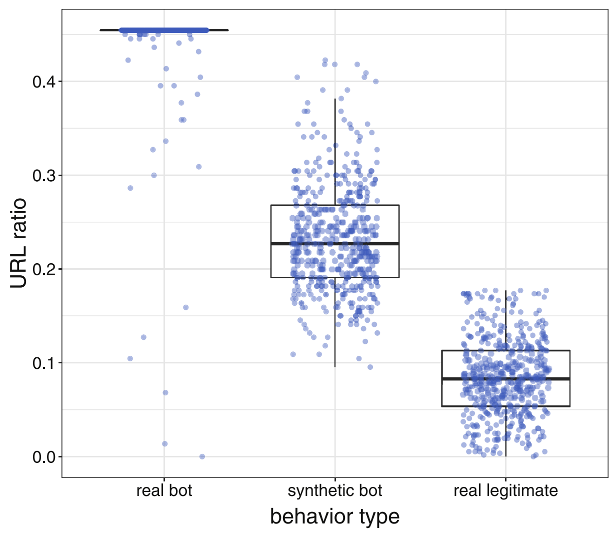
Fig. 15(b) considers the spambots in Bot2 and measures the statistical distribution of the *URL ratio* – that is, the ratio between the number of tweets containing a URL and the total number of tweets (i.e., $\frac{\#URL}{\#tweets}$). As shown by the plots, it is quite clear that the real spambots are URLs spammers. Indeed, their URL ratio is in the region of 0.45. Such spambots are detected by the Social Fingerprinting technique (see Fig. 5(b), Section 2.1.3). After the evolution, their URL ratio decreases to about 0.23. This is because the algorithm let the original Bot2 evolve to make their behavior more similar to that of the legitimate ones. Even if the evolved bots decrease their "spammy" behavior, such characteristic is necessary to evade detection (Tables 4 and 5, Section 4). However it is worth noting that, on average, the evolved bots are still capable of tweeting 1 tweets with URLs every 4 tweets, all of this while still remaining undetectable. Importantly, this URL ratio is much greater than the one featured by the legitimate accounts in the humans dataset and thus, the evolved bot are still capable of performing their URL spam activity, although at a reduced rate.

6. Related work

With the widespread adoption of OSNs, search engines, and e-commerce platforms, we assisted to the rapid proliferation of many studies on modeling and analysis of online behaviors [21]. Modeling and analyzing online user behaviors deserves attention for a variety of reasons. One is to mine substantial information regarding events of public interest [22]. In addition, linking behaviors to a ground truth in the past leads to predict what will likely happen in the future when similar behaviors take place [23]. Furthermore, online behavioral analysis helps in detecting fictitious and deceitful accounts that may distribute spam or lead to a bias in the public opinion [3,4,6,24,25]. Past research on online behaviors exploits different techniques, relying, e.g., on social and interaction graphs [26,27], textual content [22,28], and other complex data representations [23].



(a) Retweets/tweets ratio: comparison among accounts in Bot1, the evolved generation, and the legitimate accounts



(b) URLs/tweets ratio: comparison among accounts in Bot2, the evolved generation, and the legitimate accounts

Fig. 15. Malicious behavior featured before and after GA application.

To this regard, the definition of a unifying approach is an open challenge. The introduction of the novel notion of *digital DNA* sequences to characterize online user behaviors on social media brings a high flexibility and makes this original modeling technique well suited for different scenarios, with the potential to open up new directions for research. As an example, in [13], the authors study online human behaviors from a different perspective, with the aim at verifying if a general model exists for approximating them, regardless of the precise contextual activities the users are carrying out. The promising results in [13], also supported by other recent literature [29–31], let us continue to investigate this line of research, shifting however the research challenge towards the possibility of synthesizing spambots that are able to resemble legitimate behaviors and, thus, that are able to remain undetected from state-of-art detection techniques.

The threatening outcome of this study has however also another – positive – effect: the possibility to predict future evolutions of spambots and thus to re-design more efficient detection techniques, following the proactive spambot detection approach [8].

7. Conclusions

In this paper, we presented a novel approach to the task of bot detection. We first discussed a new proactive detection paradigm, that allows a quantitative study on future evolutions of social bot behaviors. Then, we provided a concrete and extensive experimental campaign on the applicability of this proactive scheme, analyzing groups of evolved social spambots that are, to date, still mostly active (not suspended nor deleted by Twitter). In theory, the proposed proactive framework is general and applicable to a broad set of accounts and detection techniques. In practice, we presented the effectiveness of the proactive scheme on a specific behavior-based state-of the-art detection system, that proved very effective in recent evaluation campaigns. Our results are promising, since we were able to synthesize spambot behaviors that remain undetectable to the detection technique.

Although unlikely to completely defeat malicious behaviors, the application of the proposed proactive schema would nonetheless bring groundbreaking benefits. Indeed, the possibility to foresee possible evolutions of spambots, and to *a priori* design and test detection techniques, would substantially raise the bar for spambot developers. As a final remark, the novel proactive schema has been here grounded on genetic algorithms and on the recent advances in digital DNA behavioral modeling, since they currently represent its key enabling factors. However, it is likely that in the near future the same proactive approach to spambot detection could leverage different techniques and methodologies, thus widening the applicability of different proactive solutions. In fact, the implementation of the proposed proactive approach with different models of the accounts and different detection techniques (e.g., those based on network characteristics or on the content of posted messages) currently represents a promising research direction.

Acknowledgments

This work was supported in part by the EU H2020 Program under the schemes INFRAIA-1-2014-2015: Research Infrastructures grant agreement #654024 SoBigData: Social Mining & Big Data Ecosystem and MSCA-ITN-2015-ETN grant agreement #675320 European Network of Excellence in Cybersecurity (NECS) and in part by the MIUR under grant “Dipartimenti di eccellenza 2018–2022” of the Department of Computer Science of Sapienza University.

References

[1] E. Ferrara, O. Varol, F. Menczer, A. Flammini, Detection of promoted social media campaigns, in: Proceedings of the 10th International Conference on Web and Social Media (ICWSM’16), AAAI, 2016, pp. 563–566.
[2] S. Cresci, Harnessing the Social Sensing revolution: Challenges and Opportunities, University of Pisa, 2018 PhD dissertation.
[3] J. Ratkiewicz, M. Conover, M.R. Meiss, B. Gonçalves, A. Flammini, F. Menczer, Detecting and tracking political abuse in social media, in: Proceedings of the 5th International Conference on Web and Social Media (ICWSM’11), AAAI, 2011, pp. 297–304.
[4] S. Cresci, F. Lillo, D. Regoli, S. Tardelli, M. Tesconi, $FAKE: Evidence of spam and bot activity in stock microblogs on Twitter, in: Proceedings of the 12th International Conference on Web and Social Media (ICWSM’18), AAAI, 2018, pp. 580–583.
[5] S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, M. Tesconi, The paradigm-shift of social spambots: Evidence, theories, and tools for the arms race, in: Proceedings of the WWW’17 Companion, ACM, 2017, pp. 963–972.

[6] C. Yang, R. Harkreader, G. Gu, Empirical evaluation and new design for fighting evolving Twitter spammers, IEEE Trans. Inf. Forensics Secur. 8 (8) (2013) 1280–1293.
[7] E. Ferrara, O. Varol, C. Davis, F. Menczer, A. Flammini, The rise of social bots, Commun. ACM 59 (7) (2016) 96–104.
[8] S. Tognazzi, S. Cresci, M. Petrocchi, A. Spognardi, From reaction to proaction: unexplored ways to the detection of evolving spambots, in: Proceedings of the 27th Web Conference Companion (WWW'18 Companion), ACM, 2018, pp. 1469–1470.
[9] S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, M. Tesconi, DNA-inspired online behavioral modeling and its application to spambot detection, IEEE Intell. Syst. 5 (31) (2016) 58–64.
[10] S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, M. Tesconi, Social fingerprinting: detection of spambot groups through DNA-inspired behavioral modeling, IEEE Trans. Dep. Secur. Comput. 4 (15) (2018) 561–576.
[11] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998.
[12] X. Hu, J. Tang, H. Liu, Online social spammer detection, in: Proceedings of the 28th International Conference on Artificial Intelligence (AAAI'14), AAAI, 2014, pp. 59–65.
[13] S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, M. Tesconi, Exploiting digital DNA for the analysis of similarities in Twitter behaviours, in: Proceedings of the 4th International Conference on Data Science and Advanced Analytics (DSAA'17), IEEE, 2017, pp. 686–695.
[14] D. Gusfield, Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology, Cambridge University Press, 1997.
[15] M. Arnold, E. Ohlebusch, Linear time algorithms for generalizations of the longest common substring problem, Algorithmica 60 (4) (2011) 806–818.
[16] L. Chi, K. Hui, Color set size problem with applications to string matching, in: Combinatorial Pattern Matching, Springer, 1992, pp. 230–243.
[17] M. Avvenuti, S. Bellomo, S. Cresci, M.N.L. Polla, M. Tesconi, Hybrid crowdsensing: a novel paradigm to combine the strengths of opportunistic and participatory crowdsensing, in: Proceedings of the 26th International Conference on World Wide Web Companion, 2017, pp. 1413–1421.
[18] S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, M. Tesconi, The paradigm-shift of social spambots: evidence, theories, and tools for the arms race, in: Proceedings of the 26th International Conference on World Wide Web (WWW'17 Companion), ACM, 2017, pp. 963–972.
[19] G. Palshikar, Simple algorithms for peak detection in time-series, in: Proceedings of the 1st International Conference on Advanced Data Analysis, Business Analytics and Intelligence, IIMA, 2009.
[20] V. Lampos, N. Cristianini, Nowcasting events from the social web with statistical learning, ACM Trans. Intell. Syst. Technol. (TIST) 3 (4) (2012) 72.
[21] L. Cao, In-depth behavior understanding and use: the behavior informatics approach, Inf. Sci. 180 (17) (2010) 3067–3085.
[22] A. Tsakalidis, S. Papadopoulos, A. Cristea, Y. Kompatsiaris, Predicting elections for multiple countries using twitter and polls, IEEE Intell. Syst. 30 (2) (2015) 10–17.
[23] K. Li, Y. Fu, Prediction of human activity by discovering temporal sequence patterns, Pattern Anal. Mach. Intell. IEEE Trans. 36 (8) (2014) 1644–1657.
[24] H. Liu, J. Han, H. Motoda, Uncovering deception in social media, Social Netw. Anal. Min. 4 (1) (2014) 162.
[25] S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, M. Tesconi, Fame for sale: efficient detection of fake Twitter followers, Decis. Support Syst. 80 (2015) 56–71.
[26] C. Yang, R. Harkreader, G. Gu, Empirical evaluation and new design for fighting evolving twitter spammers, IEEE Trans. Inf. Forensics Secur. 8 (8) (2013) 1280–1293.
[27] M. Jiang, P. Cui, A. Beutel, C. Faloutsos, S. Yang, Catching synchronized behaviors in large networks: a graph mining approach, ACM Trans. Knowl. Discov. Data (TKDD) 10 (4) (2016) 35.
[28] X. Hu, J. Tang, H. Liu, Online social spammer detection, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2014, pp. 59–65.
[29] P. Van Mieghem, N. Blenn, C. Doerr, Lognormal distribution in the digg online social network, Eur. Phys. J. B 83 (2) (2011) 251–261.
[30] C. Wang, B.A. Huberman, Long trend dynamics in social media, EPJ Data Sci. 1 (1) (2012) 1–8.
[31] C. Doerr, N. Blenn, P. Van Mieghem, Lognormal infection times of online information spread, PLoS One 8 (5) (2013) e64349.

Stefano Cresci received his PhD in Information Engineering from the University of Pisa in 2018. He is a Researcher in the Institute of Informatics and Telematics of the Italian National Research Council (IIT-CNR). His research interests broadly fall at the intersection of Web Science and Data Science and include topics such as social media analysis, online social networks security, and crisis informatics. He is member of the IEEE and of the IEEE Computer Society.

Dr. Marinella Petrocchi (MSc 1999, PhD 2005) is a computer science researcher at IIT-CNR, Italy. Her research interests include online behavioral models and analyses, aimed at investigating the fundamental laws to discriminate genuine vs fake accounts behaviours.

Prof. Angelo Spognardi is assistant professor at Computer Science department, Sapienza University of Rome. He has previously been affiliated to Systems Engineering (ESE) Compute DTU, Technical University of Denmark, the Institute of Informatics and Telematics in Pisa, Italy, and at INRIA Rhone-Alpes in Grenoble, France. His main research activity is focused on information security. In particular: fake phenomenon in Social Media, fake content analysis within review based systems and social networks; analysis of the effects of fake information on evaluation metrics and resilient metrics; fake follower detection on Twitter; system and network security, with a particular emphasis on mobile and resource constrained devices, like Wireless Sensor Networks, mobile devices and RFID systems.

Dr Stefano Tognazzi (MSc 2015) is a PhD Candidate at IMT School For Advanced Studies Lucca, Italy. His research interests include system modeling and analysis, aimed at discovering model reduction techniques that preserve the structure and the behavior of the original model.