An Efficient Method to Determine which Combination of Keywords Triggered Automatic Filtering of a Message

Ruohan Xiong
Citizen Lab, University of Toronto

Jeffrey Knockel
Citizen Lab, University of Toronto

Abstract

WeChat, the most popular social media platform in China, has over one billion monthly active users. China-based users of the platform are subject to automatic filtering of chat messages limiting their ability to freely communicate. WeChat is one among many Chinese Internet platforms which automatically filter content using *keyword combinations*, where if every keyword component belonging to a blacklisted keyword combination appears in a message then it is filtered. Discovering these sensitive combinations has previously been performed by sending messages containing potentially sensitive news articles and, if the article is filtered, attempting to isolate the triggering keyword combination from the article by sending additional messages over the platform. However, due to increasing restrictions on account registration, this testing has become decreasingly economical. In order to improve its economy, we analyzed the algorithm previously used to extract keyword combinations from news articles and found large areas of improvement in addition to subtle flaws. We evaluate multiple approaches borrowing concepts from group testing literature and present an algorithm which eliminates the aforementioned flaws and which requires on average 10.3% as many messages as the one previously used.

1 Introduction

Internet platform companies operating in China are required by law to control content on their platforms or else face penalties under the expectation that companies will invest in the technology and personnel required to ensure compliance [15]. These requirements form a system of intermediary liability or "self-discipline" in which Internet platform companies are held liable for content on their services [16]. Previous work has found little consistency in what content different Chinese Internet platforms censor [12–14]. However, some high profile Internet platforms are known to frequently receive government directives [2, 17].

Many Chinese Internet platforms perform automatic content filtering *client-side*, i.e., inside a user's application, where the entire list of forbidden content can be reverse engineered [3, 8, 12]. However, in this work we concern ourselves with platforms which filter *server-side*, i.e., on a remote server, where sample testing is required to measure which content is automatically filtered. Performing sample testing generally requires control of user accounts to send test content.

Our work in this paper is motivated by our desire to understand which blacklisted content triggers automatic message filtering on WeChat, China's most popular social media platform. WeChat uses a specific kind of filtering that filters messages based on *keyword combinations*, i.e., the presence of blacklisted combination of keywords in the message [20, 21]. We focus on the previously used task of testing news articles for filtering and, by using additional messages, identifying which keyword combination in the article is triggering the article's filtering. However, due to increasing restrictions on account registration, sample testing has become decreasingly economical [19]. Many Chinese Internet platforms now require non-virtual phone numbers for account registration and ban accounts for sending excessive sensitive content. Thus, we require methods that determine blacklisted content using as few test messages as possible.

In this paper, we present, given an automatically filtered body of text, a method to identify, using a small number of test messages, a sensitive keyword combination triggering its filtering. Our methods utilize techniques from group testing literature [6]. This literature studies the problem of using a small number of tests to identify positives among a large population of samples. A single test can include any number of samples, but the output of the test is binary, positive when *any* of the tested samples are positive. We adapt techniques from group testing to solve our problem of finding triggering keyword combinations in filtered news articles.

This paper makes the following contributions:

- We formally define keyword combination based filtering and present Internet platforms in addition to WeChat that censor content using this filtering method.
- By analyzing an algorithm previously used to isolate sen-

sitive keyword combinations in text we identify several subtle flaws in it.
- By borrowing concepts from group testing literature we develop an algorithm that requires fewer messages and that fixes these flaws.
- We empirically evaluate the algorithm using historical data and find that this algorithm requires 10.3% as many messages as the one used in previous research.

These findings facilitate more economical discovery of blacklisted keyword combinations. Although our work uses chat messaging data from WeChat for empirical evaluation, our methods apply to any platform where filtering is implemented using keyword combinations.

2 Related work

While the Great Firewall of China filters on the network level, most individual communication is filtered within domestic Internet platforms by private companies. Work examining such systems has included studying censorship performed by search engines [18, 25] as well as by social media platforms including blogs [10, 15] and micro-blogs [1, 17, 26]. Research on chat censorship has mostly studied platforms that perform censorship client-side. This is because client-side implementations can be reverse-engineered to extract an exhaustive list of keywords or rules used to trigger filtering. Such work has studied instant messaging apps [3, 8, 11], live streaming platforms [12], mobile games [14], and open source GitHub projects [13].

While there are methodological advantages to studying platforms performing censorship client-side, often the most popular platforms in a social media industry segment perform censorship server-side, lending to client-side-only analyses providing an incomplete view of censorship in that industry segment. Due to its popularity in China, previous work studying server-side chat censorship has mostly focused on WeChat [4, 20, 21]. These studies use a method in which text extracted from news articles is sent as messages in a WeChat group using test accounts. If a message is not received it is flagged as censored. This method is limited by the cost of obtaining non-virtual phone numbers for the creation of test accounts. In our work we seek to make server-side chat measurement more economically viable by reducing the number of messages required to determine sensitive content.

The closest work to ours is that of Espinoza et al. [7] who experimented with a method for selecting individual keywords from a news article for testing based on their semantic properties. In their work, they assume a model where keywords may only be tested for censorship individually. However, in ours, we assume that large messages the size of entire news articles may be tested for censorship. We also consider that censorship may be triggered by the simultaneous presence of a combination of keywords as opposed to only the presence of a single keyword.

3 Overview of keyword combination blocking

In this section we briefly set out what keyword combination based filtering is and how different service operators implement it. We provide examples of Internet platforms that use this type of filtering.

Keyword combination filtering is a censorship technique automatically applied to user-inputted strings of text, such as messages in chat apps, posts on blogs, or queries in search engines. We henceforth refer to all such user-inputted strings of text subject to this filtering as *messages*. In this censorship system a message is filtered according to whether it contains any sensitive *keyword combinations* (e.g., 新疆 + 集中营 [Xinjiang + concentration camp]), where a keyword combination is a set of one or more strings of characters called *keyword components* (e.g., “新疆” or “集中营”). The order of appearance of the components in the message does not matter. We say that a keyword combination is *sensitive* in a message if the presence of all of its components triggers the message’s automatic filtering and if removing any component or character from a component would result in the automatic filtering no longer being triggered. Similarly, we say that a keyword component is sensitive in a message if it belongs to a sensitive keyword combination appearing in that message, and we say that a character at some index in a message is sensitive when it appears inside a sensitive component containing that index.

Unlike traditional single-keyword-based filtering, implementing filtering via keyword combinations allows more finely tuned targeting of content. For instance, a traditional single-keyword blacklist might include 习近平 [Xi Jinping] to filter negative criticism of the Chinese President, but this would filter general discussion about him as well. Alternatively, the combination 习近平 + 三连任 [Xi Jinping + three consecutive terms], which was previously found censored on WeChat [5], more finely targets sensitive conversations without censoring other discourse.

In addition to WeChat, we found via testing that keyword combination based filtering is implemented in other Chinese services, including Alibaba’s Wangwang (阿里旺旺) chat platform (e.g., 法轮 + 功 [Falun + Gong]) and direct messaging in the Zhihu (知乎) Q&A platform (e.g., 微博 + 微信 [Weibo + WeChat], a keyword combination possibly intended to control spam or filter references to competitors). We also found via sample testing that Sina Weibo uses keyword combinations to automatically filter posts (e.g., 习近平 + 搞个人崇拜 [Xi Jinping + engage in personal worship]). The LINE chat app had been previously found to implement such filtering using regular expressions [22] (e.g., 北京 + 政变 [Beijing + coup] via the expressions `.*北京.*政变.*`

and `.*政变.*北京.*`), although this was implemented client-side, not server-side.

On some Internet platforms we observed a degenerate form of keyword combination filtering where we found that blacklists consisted of a single keyword but where for some keywords each character of the keyword could appear anywhere in the message and in any order. For these keywords, the filtering rules are equivalent to a keyword combination where each component is one character of the original blacklisted keyword. We found source code from the Soku search engine [24] that refers to this type of matching as "fuzzy" hit matching, and we have also observed this behavior in the Sina Weibo search engine with some sensitive keywords (e.g., 法 + 轮 + 功 [Fa + lun + Gong]).

4 Previous algorithm

Previous work has used news articles to discover blacklisted keyword combinations. In this section we describe our findings of analyzing the algorithm previously used [4, 20, 21] to isolate a sensitive keyword combination in a filtered news article. We briefly summarize the algorithm, analyze its performance, and identify two subtle flaws in its implementation which reveal the difficulty of designing an algorithm to correctly determine keyword combinations.

4.1 Summary of previous algorithm

To isolate a sensitive keyword combination in a censored article, we found that the previous algorithm begins with a *bisection* step which separates the article into two halves whose lengths have difference no greater than one. The algorithm then independently tests each half to see if it is censored by sending each half as a message. If the left half of the bisection is censored, this bisection process is recursively applied on that half, else if the right half is censored, then this process is recursively applied to that half. This process is repeated until a bisection results in two halves both of which are not censored on their own. At this point, the last message to be censored is returned as the *reduced article*, which contains a sensitive keyword combination but typically also includes non-sensitive characters. The characters in the reduced article are then sequentially and unilaterally tested for sensitivity by trialing the removal of characters one at a time. If removing a character results in a message that is no longer censored, then that character is marked as sensitive. The non-sensitive characters are then removed, leaving only segments of sensitive characters. Since keyword components can appear adjacent to each other in a censored message, component boundaries between characters are determined in the *component splitting* step by sequentially testing the addition of null characters to the censored string. If inserting a null character between two sensitive characters still results in a filtered message, then there exists a component boundary at the index of insertion. The censored string is then split at all discovered component boundaries, and the set of components is returned as the sensitive keyword combination.

The performance of this algorithm is highly dependent on the positions of the sensitive components in the article. The performance is poor when components are spaced far apart. In this case the bisection terminates early, leaving a large amount of the article to be tested character-by-character for sensitivity.

4.2 Flaws in previous algorithm

In this section we describe two different, subtle flaws that we discovered in the algorithm that we analyzed.

The **Unilateral Elimination Flaw** exists because, after bisecting, the algorithm *unilaterally* removes characters, i.e., it removes characters one at a time and even if removing the character preserves the censorship of the message it puts the character back for the next test. This is as opposed to it *iteratively* removing characters, i.e., when removing a character preserves the censorship of the message then do not include that character in future tests.

This flaw manifests itself in two conditions. The first is when more than one unique sensitive keyword combination exists in the message. In this case, only the keyword components which the combinations have in common (if any) will be discovered, as the removal of characters from the other components will never cause the message to fail to be censored. The second occurs when a component from the censorship-triggering keyword combination appears multiple times in the reduced article, i.e., the result of the bisection step. That component will not be identified as part of its sensitive keyword combination because the algorithm unilaterally removes individual characters, and so no removal will modify all occurrences of the component. In this case, the characters in the component will not appear to the algorithm to be responsible for triggering censorship.

The **Overlap Flaw** is a result of the mishandling of sensitive components that overlap in the tested message. This flaw would exist even if the algorithm iteratively removed characters, and its existence demonstrates the subtlety of correctly implementing the algorithm, as we found no simple change to the algorithm which would resolve this flaw. Through testing, we confirmed that WeChat censors a message even if the components in the sensitive keyword combination overlap in the message. For example, we found that in the censored combination 帶來 + 調整 + 整體 + 領域 [bring + reform + entire + field], since the component 調整 ends with the same character that 整體 begins with, a message can be created such that they overlap, e.g., 帶來abc調整體xyz領域. This algorithm will incorrectly return 帶來 + 調整體 + 領域 as the triggering keyword combination.

Algorithm 1 BINSEARCH(S, g)

```
$lo \leftarrow 0$, $hi \leftarrow |g|$
while $hi - lo > 1$ do
    $mid \leftarrow \lfloor (lo + hi)/2 \rfloor$
    if ISCENSORED$(S \cup \{g[mid:]\})$ then
        $lo \leftarrow mid$
    else
        $hi \leftarrow mid$
    end if
end while
return $lo$
```

Algorithm 2 SPLITCOMPS(C)

```
$D \leftarrow \{\}$
for $s \in C$ do
    $i \leftarrow 0$
    for $j = 1, \ldots, |s| - 1$ do
        if ISCENSORED$(C \setminus \{s\} \cup \{s[:j], s[j:]\})$ then
            $D \leftarrow D \cup \{s[i:j]\}$
            $i \leftarrow j$
        end if
    end for
    if $i < |s|$ then
        $D \leftarrow D \cup \{s[i:]\}$
    end if
end for
return $D$
```

5 Improved algorithms

In this section we present three different algorithms that improve upon the one used in previous work described in the prior section. We desire to reduce the number of messages required to isolate a sensitive keyword combination in a censored article while simultaneously rectifying the previous algorithm's flaws.

5.1 Framework

In this section we lay out the basic operations used in the algorithms we describe.

- $|s|$ – the length of string s.
- $s_1 \parallel s_2$ – concatenate strings s_1 and s_2.
- $s[i{:}j]$ – slice zero-indexed string s from index i, inclusive, to j, exclusive. If i or j are omitted, then slice from the beginning or to end of the string, respectively.
- $s[i]$ – $s[i{:}i+1]$.
- ISCENSORED(S) – **true** if, for $s_1, s_2, \ldots, s_n \in S$, $s_1 \parallel$ `"\0"` $\parallel s_2 \parallel$ `"\0"` $\parallel \ldots \parallel s_n$ is censored by WeChat, else **false**. Null characters are placed in between the concatenated strings so that no new sensitive keyword components can be accidentally introduced crossing multiple strings in S.
- BINSEARCH(S, g) – find and return the index of the leftmost character of a sensitive component in g by performing binary search over g given S, a set of strings to include in all test messages. See Algorithm 1 for full details.
- SPLITCOMPS(C) – for each string $s \in C$, split s, which may be multiple keyword components concatenated together, into all of its individual components. See Algorithm 2 for full details.

Algorithm 3 BINSEARCHBACKTRACK(s)

```
$stack \leftarrow$ empty stack with LIFO PUSH() and POP() ops
PUSH$(stack, (0, |s|))$
repeat
    $(lo, hi) \leftarrow$ POP$(stack)$
    $z \leftarrow$ a string of $(hi - lo)$-many "\0"s
    $s' \leftarrow s[:lo] \parallel z \parallel s[hi:]$
    if $hi - lo < |s|$ and ISCENSORED$(s')$ then
        $s \leftarrow s'$
    else if $hi - lo > 1$ then
        $mid \leftarrow \lfloor (lo + hi)/2 \rfloor$
        PUSH$(stack, (mid, hi))$
        PUSH$(stack, (lo, mid))$
    end if
until $|stack| = 0$
$C \leftarrow$ the set of nonempty strings from splitting $s$ by "\0"
return SPLITCOMPS$(C)$
```

5.2 Binary search with backtracking

We found that a simple but effective way to enhance the algorithm presented in Section 4 is to not abort the binary search when neither side of a bisection is censored when tested by itself. The key insight is that binary search can be extended to find the exact sensitive *subsequence* of characters, as opposed to merely some *substring* that contains the sensitive subsequence. We accomplish this by performing binary search while (1) zeroing out, with null characters, bisection halves of the article in order to test if any sensitive character is contained in either half and (2) performing recursive backtracking in the binary search whenever both halves of a bisection contain a sensitive character. If a half contains no sensitive characters (i.e., if the modified message is filtered), then we keep that half zeroed out. We revert this modification if it caused the message to no longer be filtered and, similar to the previous algorithm, we recursively attempt to separate the modification into two smaller halves. See Algorithm 3 for full details.

Upon termination of the binary search step in this algorithm, the entire article will have been zeroed out except for all characters of one sensitive keyword combination. However, the resulting nonadjacent set of strings C may not be the exact sensitive keyword combination, as adjacent keyword components in the article will appear as a single component in C. Since we may need to further split each string, as a final step we test for the existence of additional component boundaries using SPLITCOMPS(C).

As a result of the use of backtracking in the algorithm, we no longer need to spend messages sequentially testing the removal of characters as in the previous algorithm once both halves during the binary search contain a sensitive character. Moreover, since our character zeroing is persistent and not unilateral, we have also fixed the Unilateral Elimination Flaw identified in the previous algorithm. However, this algorithm does not address the Overlap Flaw as it still does not correctly detect components which overlap in an article.

5.3 Binary splitting

We recognized that our problem of finding sensitive characters was similar to that explored in group testing literature, which studies the problem of using a small number of tests to identify positives among a large population of samples. A single test can include any number of samples, but the output of the test is binary, positive when *any* of the tested samples are positive.

The next algorithm that we implemented was adapted from the *binary splitting* group testing algorithm [6, p. 24]. The intuition behind binary splitting is that, unlike with binary search, after the discovery of each positive sample (and the discovery of whichever negative samples were identified during the search by any negative-yielding test), the remaining unidentified samples are then regrouped together and binary search is again applied to them as a whole. This is often desirable as binary search requires a sublinear number of tests with respect to the number of samples being searched, making it less expensive to search a larger space all at once versus smaller spaces serially. The general steps of this method for finding positive samples among a group of samples can be summarized as follows:

While there exist remaining unidentified samples:

1. Test all remaining samples.
2. If the result is negative, then mark all remaining samples as negative.
3. Else if the result is positive, perform binary search to find one positive sample and mark it as positive. Mark all the samples in any negative-resulting test during the binary search as negative.

We directly adapt this algorithm to find a sensitive keyword combination in an article s as shown in Algorithm 4. We found that this algorithm actually performed more poorly (see Section 6) than the binary search with backtracking algorithm despite being taken from group testing literature. Also, similar to binary search with backtracking, this algorithm also does not resolve the Overlap Flaw, as it does not correctly detect overlapping components.

Algorithm 4 BINSPLIT(s)

```
$C \leftarrow \{\}$
$t \leftarrow$ ""
**repeat**
    $i \leftarrow$ BINSEARCH($C \cup \{t\}, s$)
    **if** $i \neq 0$ **and** $|t| > 0$ **then**
        $C \leftarrow C \cup \{t\}$
        $t \leftarrow$ ""
    **end if**
    $t \leftarrow t \parallel s[i]$
    $s \leftarrow s[i+1:]$
**until** $|s| = 0$ **or** ISCENSORED($C \cup \{t\}$)
**if** $|t| > 0$ **then**
    $C \leftarrow C \cup \{t\}$
**end if**
**return** SPLITCOMPS($C$)
```

5.4 Component-aware binary splitting

Upon observing that the previous algorithm actually performed more poorly, we decided to analyze where the algorithm performed worse. Group testing literature generally assumes the mutual independence and uniformity of positive samples. Thus, since our direct translation of the binary splitting algorithm in the previous section had no awareness of keyword components, it found each sequential character of a keyword component by performing binary search on the remainder of the article. The binary search with backtracking algorithm performed better because it generally had to search a smaller space for each sequential character due to the use of backtracking. In response to this, we decided to modify the binary splitting algorithm to specifically cater to our problem domain and to be more "aware" of the concept of keyword components.

To accomplish this, we modified the binary splitting algorithm such that when a sensitive character is found, we sequentially test each character moving rightward until finding the end of the sensitive keyword component. See Algorithm 5 for full details. This algorithm does not need to bisect from scratch to find each sequential character of a sensitive keyword component, but it still benefits from binary splitting's regrouping of the remaining unevaluated characters after the discovery of each sensitive keyword component.

When designing this algorithm to be more component-aware, we also saw an opportunity to make the algorithm more efficient. When looking for the end of each sensitive keyword component, by carefully coding this test, we were able to obviate the need to call SPLITCOMPS at the end of our algorithm. After finding the first sensitive character of a component, a naïve approach would have been to test each successive character until finding one that is non-sensitive. Instead, we design the inner loop to increment j until ISCENSORED($C \cup \{s[i{:}j], s[i+1{:}]\}$), which is when j is large enough

Algorithm 5 COMPAWAREBINSPLIT(s)

```
$C \leftarrow \{\}$
repeat
    $i \leftarrow$ BINSEARCH($C, s$)
    $j \leftarrow i + 1$
    $k \leftarrow |s|$
    while $j < k$ do
        if ISCENSORED($C \cup \{s[i:j], s[i+1:]\}$) then
            $k \leftarrow j$
        else
            $j \leftarrow j + 1$
        end if
    end while
    $C \leftarrow C \cup \{s[i:j]\}$
    if $j \neq |s|$ then
        $s \leftarrow s[i+1:]$
    else
        $s \leftarrow$ ""
    end if
until $|s| = 0$ or ISCENSORED($C$)
return $C$
```

to fully slice out the entire sensitive keyword component at index i. This condition ensures that the loop terminates even if there is an adjacent sensitive keyword component at index j, and so the discovered sensitive keyword component will never need to be further split.

Finally, we also saw a simple opportunity to eliminate the remaining unresolved flaw in the previous algorithms. By carefully choosing how to slice out the "remaining article" (i.e., anywhere $s[i+1:]$ appears in Algorithm 5), we were able to handle overlapping keyword components, thus resolving the Overlap Flaw. Consider instead if in Algorithm 5 we had naïvely taken the remainder of the article as $s[j:]$, i.e., the remainder of the characters after the sensitive sequence of characters under consideration has been removed, instead of $s[i+1:]$, i.e., the remainder of the characters after only the first character in the sensitive sequence of characters under consideration has been removed. In this case, the algorithm's inner loop terminates when ISCENSORED($C \cup \{s[i:j], s[j:]\}$). However, if, for the component at index i, there exists a second component overlapping it starting at some index $> i$ but $< j$, then the inner loop will increment j until this second component is fully contained in the $s[i:j]$ slice such that ISCENSORED($C \cup \{s[i:j], s[j:]\}$). This will erroneously combine the two components (e.g., "abcde" instead of "abc" and "cde"). Since a component can overlap as early as the $(i+1)$th index, we use $s[i+1:]$ instead of $s[j:]$.

6 Empirical evaluation

In this section we evaluate the three algorithms we introduced and compare them to the algorithm used in previous research.

Method	**Average messages per news article**
Previously used algorithm	342.72
Binary search with backtracking	57.01
Binary splitting	70.11
Component-aware binary splitting	35.47

Table 1: **For each algorithm, the average # of messages required to isolate a sensitive keyword combination.**

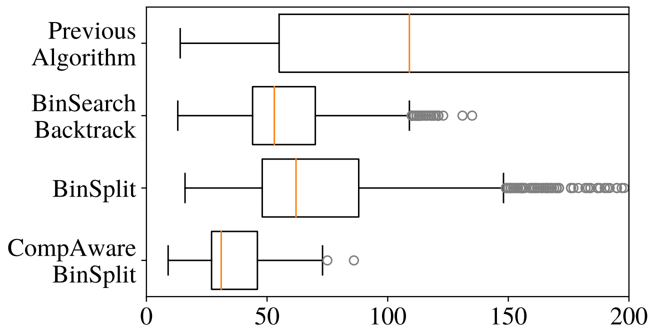

Figure 1: **Tukey box plot of the # of messages required for each algorithm to isolate a sensitive keyword combination across tested articles.**

The dataset we used for empirical evaluation consisted of 5,521 news articles filtered on WeChat, primarily written in simplified Chinese, and a list of 1,956 sensitive keyword combinations triggering their censorship, which we obtained by sample testing between September 2017 and October 2018. The average article length is 2,287 characters.

We wrote a simulator in Python that, given a keyword combination blacklist and a message, returns whether the message would be automatically filtered. The simulator implements keyword combination filtering as we define in Section 3. It filters messages even when keyword components overlap in the message to be consistent with the behavior we observed with WeChat. We then implemented the four algorithms that we evaluate. For each of the censored 5,521 news articles, we measured how many messages were required for each algorithm to isolate the sensitive keyword combination in the article. This text corpus included test cases such as multiple keyword combinations in one article, multiple keyword components of the same keyword combination in an article, and overlapping keyword components. We record whenever an algorithm does not correctly return a sensitive keyword combination in the article. We summarize the average number of queries required by each algorithm to isolate a sensitive keyword combination from a censored news article in Table 1. Additionally, we show the overall distribution of queries required to isolate a sensitive keyword combination from each article in our test set in Figure 1.

We found that the algorithm that we analyzed from previous work performed the worst, requiring on average 9.66 times as many messages as the best performing algorithm. The algorithm's highly variable and often poor performance is a result of the ineffective recursive bisection step. In 9% of articles, due to the large distance between keyword components, bisection was unable to reduce the length of the article, leaving each character of the entire article to be sequentially tested. In contrast, for at least half of the articles, when keyword components were nearby, the article was reduced to an average length of 41.6 characters before sequentially removing characters. Additionally, this algorithm exhibited both the Unilateral Elimination and Overlap flaws as described in Section 4.2. In 44% of cases, the algorithm incorrectly identified the sensitive keyword in the article. This was typically because many articles contained a sensitive keyword combination with at least one component appearing multiple times. This algorithm can mishandle this case as a result of the Unilateral Elimination Flaw.

We found that all three of the new algorithms that we introduce in this paper required substantially fewer messages. Moreover, due to designing the algorithms to not unilaterally test the removal of characters, our results empirically verified that none of the new algorithms exhibit the Unilateral Elimination Flaw.

Component-aware binary splitting, the best performing algorithm, required on average 10.3% as many messages as the original algorithm. Its interquartile range is the smallest of any of the algorithms evaluated, as its performance is also more consistent across the articles which we tested. Moreover, due to specifically designing this algorithm to handle overlapping components, our results empirically confirmed that this was the only algorithm that does not exhibit the Overlap Flaw. During simulation, the algorithm exhibited no flaws and, to our knowledge, contains no flaws. These results demonstrate that adapting techniques from group testing literature to our problem domain was an effective approach.

7 Conclusion and future work

Given the economics of server-side testing it is desirable to isolate sensitive keyword combinations using as few messages as possible. In this work we formally defined keyword combination based blocking and documented its use on multiple Chinese Internet platforms, including chat services (WeChat and Wangwang), search engines (Soku and Weibo), and blog posts (Weibo). We present new approaches to determining keyword combinations efficiently. Our component-aware binary splitting algorithm improves upon previous work in terms of overall message cost and additionally resolves some subtle flaws associated with the previous method. We empirically evaluated the algorithm using a dataset of news articles and sensitive keyword combinations and found that 35.47 queries are required on average, per article, to isolate a keyword combination which triggers censorship, 10.3% as many messages as the algorithm used in previous research. This algorithm makes continued measurement more economically viable, where research may depend on purchasing non-virtual phone numbers and where test accounts may be banned after a certain amount of time or messages sent.

It is our hope that our methods will be useful to future researchers for measuring keyword combination based censorship on a large number of Internet platforms. While not evaluated as part of this project, we expect that these methods will apply to Internet platforms other than WeChat which perform keyword combination based filtering. These methods are anticipated to work for the detection of keyword combinations in languages other than Chinese as well.

Despite our improvements, there is still room for future work in this area. In this work we discover at most one keyword combination responsible for triggering the censorship of a message. However, the algorithms discussed in this paper can be extended to find multiple combinations.

The COMPAWAREBINSPLIT algorithm performs a linear search for the end of a keyword component. We also tested an approach using one-sided binary search [23, p. 134]. While we found this slower for our dataset, if component sizes are expected to be large, one-sided binary search should perform better. The code for this approach is included in our source code release.

Moreover, we may be able to use historical data on the location of sensitive keyword components in an article or on the frequency of which a character or word has historically been sensitive to further improve the efficiency of the algorithm. In Chinese, as text contains no word separators, text segmentation is required to detect these boundaries. We have preliminary data showing that bisecting on these boundaries can further improve efficiency by 1%, but more work is needed to understand how this works and to determine if this method can be further improved.

Better performance may be obtainable by further drawing from group testing literature. We experimented with adapting *generalized binary splitting* [9], an algorithm which determines the number of samples to simultaneously test as a function of the expected number of positive samples remaining. However, after adapting this algorithm, we saw virtually no improvement, possibly due to the small number of keyword components which we expect to find in an article. Nevertheless, adapting other group testing algorithms to the problem domain addressed in this work may yet prove promising.

Acknowledgments

We would like to thank our paper's reviewers as well as Masashi Crete-Nishihata, Jakub Dalek, Miles Kenyon, Lotus Ruan, and Mari Zhou for insightful feedback and peer review. This research was supported by the Open Society Foundations.

Availability

Our source code for our simulator and for each keyword combination isolating algorithm is available here: https://github.com/citizenlab/censored-keyword-isolation

References

[1] David Bamman, Brendan O'Connor, and Noah A. Smith. Censorship and deletion practices in Chinese social media. *First Monday*, 17(3), 2012.

[2] China Digital Times. Directives from the Ministry of Truth. Retrieved from https://chinadigitaltimes.net/china/directives-from-the-ministry-of-truth/, 2019.

[3] Jedidiah R. Crandall, Masashi Crete-Nishihata, Jeffrey Knockel, Sarah McKune, Adam Senft, Diana Tseng, and Greg Wiseman. Chat program censorship and surveillance in China: Tracking TOM-Skype and Sina UC. *First Monday*, 18(7), 6 2013.

[4] Masashi Crete-Nishihata, Jeffrey Knockel, Blake Miller, Jason Q. Ng, Lotus Ruan, Lokman Tsui, and Ruohan Xiong. Remebering Liu Xiaobo: Analyzing censorship of the death of Liu Xiaobo on WeChat and Weibo. Technical report, Citizen Lab, University of Toronto, 2017.

[5] Masashi Crete-Nishihata, Lotus Ruan, Jakub Dalek, and Jeffrey Knockel. Managing the Message: What you can't say about the 19th National Communist Party Congress on WeChat. Technical report, Citizen Lab, University of Toronto, 2017.

[6] Dingzhu Du and Frank K. Hwang. *Combinatorial group testing and its applications*, volume 12. World Scientific, 2000.

[7] Antonio M. Espinoza and Jedidiah R. Crandall. Work-in-progress: Automated Named Entity Extraction for Tracking Censorship of Current Events. In *USENIX Workshop on Free and Open Communications on the Internet*, 2011.

[8] Seth Hardy. Asia Chats: Investigating Regionally-based Keyword Censorship in LINE. Technical report, Citizen Lab, University of Toronto, 2013.

[9] F. K. Hwang. A method for detecting all defective members in a population by group testing. *Journal of the American Statistical Association*, 67(339):605–608, 1972.

[10] Gary King, Jennifer Pan, and Margaret Roberts. How Censorship in China Allows Government Criticism but Silences Collective Expression. *American Political Science Review*, 107(2):326–343, 2013.

[11] Jeffrey Knockel, Jedidiah R. Crandall, and Jared Saia. Three researchers, five conjectures: An empirical analysis of TOM-Skype censorship and surveillance. In *USENIX Workshop on Free and Open Communications on the Internet*, 2011.

[12] Jeffrey Knockel, Masashi Crete-Nishihata, Jason Q. Ng, Adam Senft, and Jedidiah R. Crandall. Every Rose Has Its Thorn: Censorship and Surveillance on Social Video Platforms in China. In *5th USENIX Workshop on Free and Open Communications on the Internet*, 2015.

[13] Jeffrey Knockel, Masashi Crete-Nishihata, and Lotus Ruan. The effect of information controls on developers in China: An analysis of censorship in Chinese open source projects. In *First Workshop on NLP for Internet Freedom*, 2018.

[14] Jeffrey Knockel, Lotus Ruan, and Masashi Crete-Nishihata. Measuring Decentralization of Chinese Keyword Censorship via Mobile Games. In *7th USENIX Workshop on Free and Open Communications on the Internet*, 2017.

[15] Rebecca MacKinnon. China's Censorship 2.0: How companies censor bloggers. *First Monday*, 14(2), 2009.

[16] Rebecca MacKinnon. Networked Authoritarianism in China and Beyond: Implications for global Internet freedom. *Journal of Democracy*, 2011.

[17] Blake Miller. The Limits of Commercialized Censorship in China. Retrieved from https://osf.io/preprints/socarxiv/wn7pr/, 2019.

[18] Jason Q. Ng. Blocked on Weibo - Search result logs and full list of banned words. Retrieved from http://blockedonweibo.tumblr.com/post/12729333782/search-result-logs-and-full-list-of-banned-words, 2012.

[19] Qiao Long. China's Social Media App WeChat Demands More Info From Users. Retrieved from https://www.rfa.org/english/news/demands-06142018124702.html, 2018.

[20] Lotus Ruan, Jeffrey Knockel, and Masashi Crete-Nishihata. We (can't) Chat: "709 Crackdown" Discussions Blocked on Weibo and WeChat. Technical report, Citizen Lab, University of Toronto, 2017.

[21] Lotus Ruan, Jeffrey Knockel, Jason Q. Ng, and Masashi Crete-Nishihata. One App, Two Systems: How WeChat uses one censorship policy in China and another internationally. Technical report, Citizen Lab, University of Toronto, 2016.

[22] Adam Senft, Jason Q. Ng, Seth Hardy, and Masashi Crete-Nishihata. Asia Chats: LINE keyword filtering upgraded to include regular expressions. Technical report, Citizen Lab, 2014.

[23] Steven S. Skiena. *The Algorithm Design Manual*. Springer Publishing Company, Incorporated, 2nd edition, 2008.

[24] Soku. FuzzyHitMatcher.java. Retrieved from `https://www.github.com/RyanTech/-java-soku/blob/eb7be3389885f517c7425b62e7be9677f28e8787/WEB-INF/java/com/youku/soku/shield/matcher/FuzzyHitMatcher.java`, 2016.

[25] Nart Villeneuve. Search monitor project: Toward a measure of transparency. Technical report, Citizen Lab, 2008.

[26] Tao Zhu, David Phipps, Adam Pridgen, Jedidiah R Crandall, and Dan S Wallach. The Velocity of Censorship: High-fidelity Detection of Microblog Post Deletions. In *Proceedings of the 22nd USENIX Conference on Security*, pages 227–240, 2013.