BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

AUTOMATIC TEMPLATE PATTERN RECOGNITION

AUTOMATICKÁ IDENTIFIKACE ŠABLONY GENERUJÍCÍ SPAM KAMPANĚ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. DAVID KOVAŘÍK**
AUTOR PRÁCE

SUPERVISOR **Mgr. Bc. HANA PLUHÁČKOVÁ**
VEDOUCÍ PRÁCE

BRNO 2018

Brno University of Technology - Faculty of Information Technology

Department of Intelligent Systems Academic year 2017/2018

Master's Thesis Specification

For: **Kovařík David, Bc.**
Branch of study: Intelligent Systems
Title: **Automatic Template Pattern Recognition**
Category: Security

Instructions for project work:

1. Prostudujte metody pro identifikování stejných a různých částí textů (tokenizace, lineární programování, grafy, konečné automaty atd.) a jejich použitelnost na velký počet krátkých textů generovaných podle jedné šablony (př. SMS spam kampaně).
2. Navrhněte algoritmus, který bude schopen automaticky rozpoznat šablonu, ze které je kampaň generována, a popsat ji formálním jazykem (např. regularním výrazem).
3. Implementujte daný algoritmus.
4. Otestujte algoritmus na datasetech od firmy Mavenir s texty generovanými ze stejné šablony.
5. Vizualizujte uživateli společné a rozdílné části v dané skupině textů. Výstupem aplikace je formát uchovávající tuto vizualizaci (obrázek, HTML soubor atd.) a nalezený formální popis (regulární výraz).
6. Analyzujte použitelnost a rozšíření algoritmu na další problémy a na jiný typ vstupních dat (např. Na dlouhé texty, na binární data atd.).

Basic references:

- Dle pokynů vedoucího práce.

Requirements for the semestral defense:
Body 1 a 2.

Detailed formal specifications can be found at http://www.fit.vutbr.cz/info/szz/

The Master's Thesis must define its purpose, describe a current state of the art, introduce the theoretical and technical background relevant to the problems solved, and specify what parts have been used from earlier projects or have been taken over from other sources.

Each student will hand-in printed as well as electronic versions of the technical report, an electronic version of the complete program documentation, program source files, and a functional hardware prototype sample if desired. The information in electronic form will be stored on a standard non-rewritable medium (CD-R, DVD-R, etc.) in formats common at the FIT. In order to allow regular handling, the medium will be securely attached to the printed report.

Supervisor: **Pluháčková Hana, Mgr. Bc.**, DITS FIT BUT
Beginning of work: November 1, 2017
Date of delivery: May 23, 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

Petr Hanáček
Associate Professor and Head of Department

Abstract

Spam does not occur as separate messages, but it is sent in so-called campaigns. They are usually generated by a certain template which allows producing large amount of semantically, but not syntactically, equivalent messages. The goal of this work is to design an algorithm able to reversely extract a template of a campaign from a set of concrete messages. The main focus is on spam in SMS communication. However, proposed method is general enough for wider use. The proposed algorithm is based on a method of aligning to sequences, which is used in bioinformatics to detect similar regions of protein strings. Resulting templates are represented as regular expressions. The tool is able to visualize extracted templates using HTML. Results of the tool were validated on more than three hundred real-world campaigns. In most cases extracted regular expressions were able to identify their source campaign.

Abstrakt

Spam se typicky nevyskytuje ve formě samostatných zpráv, ale často bývá sdružován do takzvaných kampaní. Ty bývají automaticky generovány pomocí šablon. Díky tomu jsou jednotlivé zprávy sémanticky, ale ne syntakticky, ekvivalentní. Cílem práce je navrhnout algoritmus schopný z množiny zpráv jedné kampaně zpětně extrahovat šablonu, ze které tyto zprávy byly generovány. Práce se zaměřuje na spam v SMS komunikaci, ale navržené postupy jsou dostatečně obecné pro širší použití. Algoritmus je postaven na metodě zarovnávání dvou sekvencí, používané v bioinformatice pro nalezení podobných oblastí proteinových řetězců. Výstupem je regulární výraz popisující šablonu dané kampaně. Součástí řešení je také nástroj pro vizualizaci šablony pomocí HTML. Řešení bylo ověřeno na přibližně třech stovkách skutečných kampaní z celého světa. V naprosté většině případů je poskytnutý výsledek postačující pro identifikaci kampaně.

Keywords

SMS, Spam, Spam campaigns, Template extraction, Regular expression induction, Regular expressions

Klíčová slova

SMS, Spam, Spamové kampaně, Extrakce šablony, Indukce regulárních výrazů, Regulární výrazy

Reference

KOVAŘÍK, David. *Automatic Template Pattern Recognition.* Brno, 2018. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Mgr. Bc. Hana Pluháčková

Rozšířený abstrakt

Spam se typicky nevyskytuje ve formě samostatných zpráv. Místo toho je distribuován v takzvaných kampaních. Jedná se o množiny spamových zpráv s podobnou strukturou a stejným účelem. Tím může být například propagace určitého výrobku nebo distribuce škodlivého softwaru. Kampaně bývají generovány automaticky pomocí šablon — abstraktního popisu struktury zpráv. Šablona obsahuje statické a proměnlivé části, které jsou v průběhu generování kampaně expandovány na různé hodnoty. Díky tomu mohou distributoři spamu produkovat velké množství sémanticky stejných, ale syntakticky odlišných zpráv. Tato práce se primárně zaměřuje na spamové kampaně ve službě SMS. Čísla z poslední dekády ukazují, že je ve světě stále více mobilních předplatitelů, s čímž roste i počet odeslaných zpráv. Přestože tato platforma představuje pro spam obrovský potenciál, nebylo mu věnováno tolik pozornosti, jako například e-mailové komunikaci.

Cílem práce je navrhnout algoritmus schopný z množiny zpráv jedné kampaně zpětně extrahovat šablonu, ze které tyto zprávy byly generovány. Tuto šablonu následně vhodně reprezentovat na výstupu. Zpětnou extrakci šablony je možné chápat také jako proces učení formálního jazyka z množiny jeho vět. Proto jsou na úvod práce shrnuty různé metody, které byly za tímto účelem navrženy. Primárně jsou popsány metody gramatické indukce, jejichž výsledkem je konečný automat popisující daný jazyk. Navíc jsou ukázány alternativní přístupy jako například zarovnávání sekvencí nebo genetické programování.

Jádrem navrženého řešení je Needleman-Wunsch algoritmus používaný v bioinformatice pro zarovnání dvou sekvencí proteinů nebo nukleotidů. Byl patřičně rozšířen tak, aby byl schopen zarovnat dva různé texty. Extrakční algoritmus provádí inkrementální generalizaci počáteční šablony tak, aby na jeho konci daná šablona popisovala všechny zprávy dané kampaně. První načtená zpráva slouží pro konstrukci počáteční šablony. Následně jsou načítány další zprávy. Pokud šablona akceptuje čtenou zprávu, přejde se k další. Při neshodě, dojde k zarovnání šablony a zprávy a jejich následnému sloučení. Tímto krokem vznikne nová šablona, která popisuje všechny dříve akceptované zprávy a aktuálně čtenou zprávu. Po zpracování všech zpráv je výsledkem šablona, která popisuje všechny zprávy dané kampaně. Výstupem algoritmu je regulární výraz popisující extrahovanou šablonu.

Součástí algoritmu jsou i další kroky. Při neshodě šablony a zprávy, musí být daná zpráva nejdříve převedena na lexikální jednotky, aby mohla být dále zpracovávána. Zdrojová kampaň může obsahovat určitý šum v podobě zpráv, které do této kampaně nepatří. Součástí extrakce je také detekce a eliminace takových zpráv. Extrakční algoritmus je implementován v rámci nástroje, který také umožňuje ověření kvality získané šablony. Šablonu lze také extrahovat pouze z podmnožiny kampaně, nebo náhodných vzorků. Výstupem validace je informace o tom, kolik zpráv kampaně šablona popisuje, a kolik ne. Lze také generovat výstup ve formátu HTML, který demonstruje obecné vzory, které se v kampani vyskytují.

Díky procesu postupné generalizace je výsledná šablona velmi specifická pro zdrojovou kampaň. Platí tedy, že celou kampaň lze identifikovat na základě šablony. Pokud je taková šablona přidána do spamového filtru, všechny následně přijaté zprávy, které se s šablonou shodují, mohou být označeny za spam.

Kvalita extrakčního algoritmu byla ověřena na přibližně třech stovkách kampaní z celého světa. Jelikož existuje potenciálně nekonečně mnoho výrazů, které popisují stejnou kampaň, a čitelnost je součástí ohodnocení, lze jen těžko přesně říct, co je to správný výsledek. Co však konstatovat lze je fakt, že v naprosté většině případů je výsledek dostačující pro identifikaci dané kampaně. Navíc, ve většině případů popisuje zdrojovou kampaň minimalistickým způsobem — čitelnost je tedy také velmi vysoká.

Automatic Template Pattern Recognition

Declaration

Hereby I declare that this master's thesis was prepared as an original author's work under the supervision of Mgr. Bc. Hana Pluháčková.

The supplementary information was provided by Mr. Petr Salomoun.

All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
David Kovařík
May 21, 2018

Acknowledgements

I would like express my gratitude to Mgr. Bc. Hana Pluháčková for her supervision of my work, as well as to Mr. Petr Salomoun for all the advice he provided.

Contents

Chapter 1

Introduction

Short Message Service (SMS) is one of the key and most popular services available on mobile phones. According to the annual report of the International Telecommunication Union, there were up to 6.1 trillion SMS sent in 2010 [14] (around 200 000 every second). A few years later, in 2015, there were up to 7 billion cellular subscriptions world-wide [15]. Even though the use of SMS has been decreasing in some parts of the world, it is still popular in others. Especially in developing countries. On the other hand, private companies now send SMS to their customers much more. Therefore, the amount of sent SMS is still high. These facts represent significant potential for immoral or unethical activities, such as spamming. However, protection against spam in SMS has not received enough attention by researches.

Spam is usually not sent as separate messages, but rather in so-called campaigns. Messages within a campaign have similar structure and the same purpose (advertisement, etc.). Campaigns are generated automatically using templates — a meta-description of message's structure. Therefore, each message can be more personalized and more difficult to detect. The goal of this work is to design an algorithm which is able to reversely extract a template from a set of campaign's messages and describe it with a formal model.

At the beginning, in Chapter 2, more detailed description of spam, with focus on spam in SMS communication, and prevention against it is provided. Relationship between campaigns and templates is also further explained and demonstrated. This part is based on research done within the Term project together with an initial draft of the extraction algorithm.

The task of template extraction can be seen as learning a formal description of a language. Chapter 3 provides exactly this point of view. First, very naive approaches are shown. Their only goal is to demonstrate, why more advanced methods are required. Then, several methods of language acquisition, such as automata induction and genetic programming, are presented. Lastly, an idea of using sequence alignment algorithms, adopted from bioinformatics for aligning DNA strings, as a method of learning a language is shown.

Chapter 4 contains detailed description of the proposed extraction algorithm. First, the overall view and the general idea is presented followed by further specification of each crucial step. Some of the implementation details, such as an overview of available output methods can be found in Chapter 5.

There were over 300 real-world campaigns used to validate quality of outputs of the algorithm. Chapter 6 provides overview of these results. There are experiments aiming on specific aspects of the template extraction, as well as over-all summary of algorithm's performance.

Chapter 2

Spam and Spam Campaigns

First part of this chapter contains a brief description of what spam is, what is it used for and how it affects the communication environment. Second part is dedicated to spam campaigns and fighting them.

2.1 Spam

The term *spamming* stands for the practice of using electronic communication systems to send unsolicited messages to users. It is mostly used for advertisement, spreading malware and sending fraudulent messages in order to obtain sensitive information (such as passwords) of recipients. There are many forms of spam. The most common is e-mail spam. However, there are other forms too — instant messaging or SMS spam.

Back in early 2000s, right after the boom of the Internet, the OECD summarized characteristic properties of spam. They can be split into two categories — *primary* and *secondary*. The *primary* category contains characteristics which would lead most people to identify a message as spam. The *secondary* properties are frequently associated with spam, but not necessarily [20]. These properties are shown in Table 2.1.

Primary characteristics	**Secondary characteristics**
Electronic message	Uses addresses collected without prior consent or knowledge
Sent in bulks	Repetitive
Unsolicited	Untargeted and indiscriminate
Commercial	Unstoppable
	Anonymous and/or disguised
	Illegal or offensive content
	Deceptive or fraudulent content

Table 2.1: Primary and secondary characteristics of SPAM defined by the OECD [20]

2.2 SMS Spam

According to the International Telecommunication Union (ITU) of the United Nations, there were up to 7 billions cellular subscriptions in the world in 2015 [15]. During the year of 2010, around 6.1 trillion SMS messages were sent — approximately 200 000 messages every second. Astonishing is that in 2007, ITU predicted only 1.8 trillions to be sent in

2010 [14]. The reality was three times more than what was predicted. These numbers represent huge potential for spammers and their fraudulent activities.

Spam in SMS and protection against it is, however, not as well-researched as e-mail spam. Filters for e-mail spam are much more sophisticated. Users also consider SMS to be a more reliable source (awareness of SMS spam is not as high as it is with e-mails). It results in higher response rate for SMS spam. Also, SMS spamming is, thanks to introduction of unlimited SIM cards, becoming more and more cost-effective for spammers. These facts are reasons why spammers are moving their attention to SMS service more and more.

The ratio of spam in SMS differs from region to region. In North America it is below 1%, while in Asia it can go from 20% to 30%. For example, according to [5], mobile subscribers in the world received over 200 billions spam SMS during a single week of 2008.

However, SMS spam is not just an annoying thing that could be easily ignored. Mobile subscribers can actually suffer from it financially. By simply responding to an SMS, a person can sign up for a certain subscription service, or call a premium number. An example of an SMS spam message is shown in Figure 2.1.

Figure 2.1: An example of an SMS spam message (Source: [17])

The previous description defines spam from user's point of view. However, from provider's point of view, a spam message can actually be solicited and important for its recipient. Private companies often abuse unlimited subscription plans designed for regular customers to send their commercial messages. They try to avoid using more expensive means designated for commercial purposes. Such behavior is often a violation of provider's terms of service. It is in their best interest to detect such behavior as it is a financial loss for them. Therefore, commercial messages sent via non-commercial channels can be seen as spam from provider's point of view. Even though these messages are solicited for its recipient.

We can say that it is the providers who suffer from SMS spam the most. Firstly, unsolicited messages may cause higher operating and customer-support costs. It can be harmful to their reputation too. Secondly, private companies abusing unlimited plans is financial loss for providers.

2.3 Spam Campaigns

Spammers usually do not send a single spam message. Instead, they put the effort into so-called campaigns. It is a set of messages with similar structure focusing on a particular goal, such as selling a product, malware distribution, financial fraud, etc. Unfortunately, SMS spam and campaigns are not covered much in the literature. However, the core principles are the same or at least similar to e-mail spam campaigns where much more work has been done.

Campaigns may be executed in a single run, or repeatedly. The frequency of sending messages may vary in time. The distribution platform is often reused by multiple campaigns (especially with e-mail spam campaigns, where the same botnet infrastructure is reused). For sending SMS spam, so-called *SIM farms* — a device grouping many SIM cards connected to a server. They can be controlled to send messages, dial numbers, etc.

To demonstrate how advanced spam campaign business is, we can briefly look into Cutwail botnet on which authors in [13] focus. They describe how sophisticated is the customer support provided by the owners of the botnet. Even a manual on how to make a "successful" campaign is provided to customers. The manual provides guidelines in three categories:

- **Text guidelines** — In order to lure as many victims as possible, a spam message needs to be well-crafted. It needs to be convincing enough, so its victims think it is a genuine message.
- **Email address database guidelines** — Helps customers to overcome issues related to non-existing addresses, distribution of addresses among bots, ...
- **Technical guidelines** — How to increase performance of the botnet, and setting up the campaign duration (the shorter, the lower chance of being detected by spam filters).

To make a campaign efficient, a target list (email addresses, phone numbers, etc.) is necessary. Spammers can get this list by many means — crawling the Internet, malware or even purchase them.

2.4 Templates

A key component of a campaign is a *template* — a meta-description used to craft concrete messages for each recipient. It contains variable fields which are expanded during generation process. It allows a concrete message to be more personalized, as well as it makes messages more difficult to be detected or grouped with other messages of the same campaign. An example of a template could be:

Dear {John Doe|Jane Doe}, we have a special offer for you. Call 123-223-111, report a code {RAND_NUMBER} to get your $100 Walmart gift card.

The grouping attribute of a template is important for spam prevention. Not only that a template can be used to generate a campaign, it can also be used to identify it. Therefore, if we are able to reconstruct a template from a set of concrete messages of a campaign, we can add this template to a spam filter. Any message matching the template can be associated with the campaign and marked as spam.

Another reason is that the volume of spam is too high. Analyzing each message would be impossible. Merging single messages into larger sets (campaigns), while preserving their characteristic properties, reduces the amount of data that needs to be analyzed.

2.5 Fighting SMS Spam

One way of fighting SMS spam is so-called *content-based* filtering where the content of each message is analyzed (typical for e-mail spam prevention). However, due to the restricted length of SMS messages, there is not much "material" to work with. Also, the language used in SMS is a bit different — abbreviations, slang, phonetic contractions, emoticons, as well as intentional incorrect spelling are more common. All of these factors make content-based filtering more difficult and less efficient.

Another type of methods is based of generating signatures (or a fingerprints) from the content of known spam messages. This signature is distributed to subscribers. When a subscriber receives a message, its signature is computed and it is checked against all of the known spam signatures. If a matching signature is found, the incoming message is marked as spam.

However, in order to get around this filtering method, spammers started inserting variable parts into their messages. They use templates to create multiple versions of semantically, but not syntactically, equivalent messages. Therefore, the signature of each message is different. It is even common to spot completely random sequences in each message — typically numbers. This practice is called *textual polymorphism*.

Chapter 3

Methods for Template Extraction

Even though the purpose of this work is SMS spam analysis, the core idea is more general. The goal is to develop an algorithm for reversely extracting a template from a set of messages and describe it using a formal language. The core idea, however, can be seen as learning a formal language from a finite set of its sentences.

In this section, we are going to describe several methods which can be used for extracting a formal description of a language, what is their output, pros and cons. For example, some of the discussed methods use finite automata as their model, or regular expressions.

Many of the examples presented in this section are going to be real-world spam campaign. Any sensitive information (names, phone numbers, dates, etc.) contained in them will be replaced and randomized.

3.1 Naive Approach

In this part, we are going to describe a very simple approach for template extraction. It provides good results only in very specific cases and it fails on more complex data. However, it is good for demonstrating why more advanced methods are required to get satisfying results. The method does not require any advanced tokenization and basic string operations are sufficient. Therefore, it is a very fast approach. The process is demonstrated in Algorithm 1.

As mentioned before, this is a very trivial approach and it will provide good results only in very specific cases. It uses the first message to construct a template by splitting it into lexical tokens. Then, it goes through each message and tokenizes it. For each token in each message it is checked whether there is the same value on the same position in the template. If yes, nothing happens. If not, the value in the template is set to `Null` (to indicate a variable part).

After that, all sequences of `Nulls` are merged into a single one and the resulting regular expression (RE) is constructed. It goes through each token in the template — if it is a string literal, it is appended to the result. If a `Null` value is found, an expression "`(.*?)`" is added to cover values in the variable part.

The algorithm is very abstract, it does not handle white space characters, possible overgeneralization, and so on. It could also be extended to remember which concrete values are present in variable parts and list them in the result. The key issue is that it only works for messages which have the same amount of tokens (no matter how is the tokenization process defined).

Algorithm 1: Naive template extraction for trivial cases.

Input: Set of messages
Output: Template in form of a regular expression

```
temp ← tokenize the first message
foreach msg in messages do
    tokens ← tokenize msg
    for (i=0 ; i < length(res) ; i++) do
        if temp[i] ≠ tokens[i] then
            temp[i] ← Null

if res contains sequences of multiple Null values then
    Reduce the sequence into a singe Null

regex ← ""
foreach token in res do
    if token is Null then
        Append "(.*?)" to regex
    else
        Append token to regex

return regex
```

It is very common that messages of the same campaign are different in length. Therefore this approach is going to fail. However, its purpose is to demonstrate the simplest way to achieve the goal. On the other hand, there are also real-world campaigns, where this approach would work perfectly. Here is an example of such spam campaign from Argentina:

Estimado/a A B le recordamos que su cuota vencio el 15/09/2016. Puede abonarla en Sucursal. Si pago desestime el sms. Ribeiro S.A.
Estimado/a C D le recordamos que su cuota vencio el 20/09/2016. Puede abonarla en Sucursal. Si pago desestime el sms. Ribeiro S.A.

The corresponding template generated by this method is:

Estimado/a `(.*?)` *le recordamos que su cuota vencio el* `(.*?)`*. Puede abonarla en Sucursal. Si pago desestime el sms. Ribeiro S.A.*

Another example of a naive approach is shown in Algorithm 2. It simply finds tokens which occur in every message and then concatenates them together in proper order. Variable parts are, again, represented as "`(.*?)`". This method will be also very fast and it will produce relatively good results for campaigns with simple structure. However, it will fail on more complex sets.

Proposed naive approaches do not aim to solve the problem. They are useful in understanding why more advanced methods are necessary in order to get good results from real-world (more complex and unpredictable) examples.

Algorithm 2: Another method of naive template extraction.

Input: Set of messages
Output: Template in form of regular expression

global_tokens ← tokenized first message
foreach *msg in messages* **do**
    tokens ← tokenize msg
    global_tokens ← global_tokens ⋂ tokens

ordered ← global_tokens ordered by occurrence in the first message
regex ← put the placeholder `(.*?)` between tokens and concatenate it all together
return *regex*

3.2 Inferring a Formal Description

The procedure of constructing a template of a given spam campaign can be seen as a problem of extracting a formal description of a language (a grammar, automaton, etc.) from a set of positive examples.

Grammatical Inference is a task of discovering common patterns in examples which were generated by the same process. Result is a finite model in which these patterns are appropriately described. The inference process can be done using either *positive data* only, or from *complete representation* (both positive and negative samples). Positive samples are sentences of a language, while negative samples are sentences which do not belong to the language. Majority of algorithms related to grammatical inference proposed in the past are based on Gold's work — *Language identification in the limit* [11].

We are going to focus only on the class of *finite languages*, since we can see a spam campaign as a language and each concrete message as a valid sentence of the language. This fact helps us a lot, because Gold proved in [11] that only the class of finite languages can be learned from positive samples only. Other classes require negative examples. Even the class of regular languages. A summary of learnability of each class of languages is shown in Table 3.1.

Learnability model	**Class of languages**
Informant	Primitive recursive
	Context-sensitive
	Context-free
	Regular
Text	Finite-cardinality languages

Table 3.1: Table shows which classes of languages are learnable by which means. *Text* model provides the learner only with positive examples of a language. *Informant* gives the learner information whether each example belongs to the language or not. (Source: [11])

When it comes to induction of regular languages, algorithms developed in the past can be divided by the type of model they work with — finite state automata, grammars, regular expressions. Probably the most popular formal model is finite state automata as majority of methods use it for the inference. Therefore, we are also going to focus mostly on it.

3.2.1 Automata Induction

One of the formalism that can be used to learn regular languages is a finite automaton. This model is also useful for understanding, why finite languages can be learned from positive examples only. Intuitively, we can construct an automaton in which each "branch" accepts one sentence of the language. Such automaton is called *Maximal Canonical Automaton (MCA)*. An example is shown in Figure 3.1. However, such model does not produce an efficient RE, because it is merely a union of sentences of the given language. Another representation used as a starting point for induction is *Prefix Tree Acceptor (PTA)* — a tree structured automaton in which each sentence of the input dataset is accepted. If two sentences have the same prefix, they share transitions of the automaton for this prefix. An example is shown in Figure 3.2

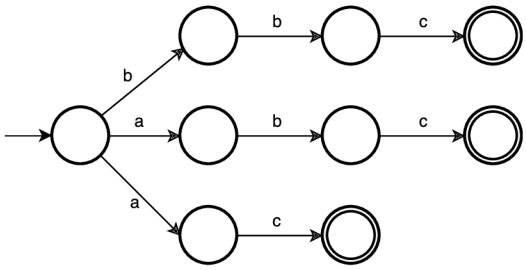

Figure 3.1: An example of a *Maximal Canonical Automaton* for the language $\{abc, ac, bbc\}$.

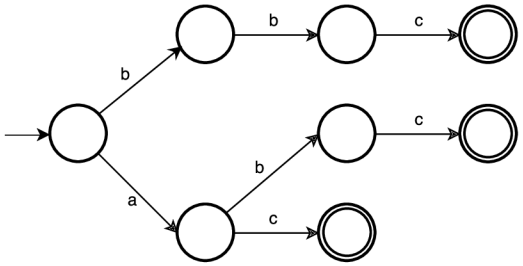

Figure 3.2: An example of *Prefix Tree Acceptor* for the language $\{abc, ac, bbc\}$.

State-Merging Approach

After establishing the initial model, sequential merging of states, as a way of generalizing the model, is performed. The order in which states are merged is important. One of the first such algorithms proposed was *RPNI (Regular Positive and Negative Inference)* [21]. Given a set of positive and negative sentences, the algorithm retrieves a Deterministic Finite Automaton (DFA) consistent with the language (it accepts all positive and rejects all negative sentences). It constructs a PTA from positive samples. Then, it recursively merges states in lexicographical order (a state and its predecessor), until the condition to accept every positive and reject every negative sample is satisfied.

An extension to RPNI, called *RPNI2* was proposed in [8]. The main difference between them is that when two states cannot be merged in RPNI, RPNI2 tries to find an inclusion relation between them in order to predict whether the prefixes of the data belong to the language, or its complement. Both RPNI and RPNI2 produce a DFA on the output.

However, non-deterministic finite automata (NFA) are generally smaller, more compact description of a language. Also REs generated from NFAs are less complex. Algorithms producing NFA have been proposed it the past. One of the well-known is *DeLeTe2*[6]. Another method, independent on the order of merging states, producing NFA on the output has been proposed in [9].

The problem with discussed methods is that they all require a set of negative examples to guide the merging algorithm. Whenever two states are merged it is checked, whether the new automaton accepts all positive and rejects all negative samples. It allows the algorithm to be more precise. Using these algorithms, when only positive samples are available, would require reducing the consistency check, which would reduce accuracy of the result.

Using an automaton as an output of template induction has several drawbacks. Firstly, it is more difficult for humans to read it. Therefore, it would be necessary to provide users with a way of transforming it to, for example, REs (which are easier to understand for humans). However, even though REs and finite state automata are equivalent, algorithms for generating REs from automatons produce very complicated, hard-to-read results. Especially when an automaton contains loops. This issue is called *exponential blow-up* [12].

3.2.2 Align-Based Approach

There are not many algorithms designed to generate REs directly as a result of grammar induction. Most of them focus on constructing DFA/NFA. However, one of such methods was proposed in [7]. It consists of several steps. Samples are first split into blocks — sequences of same characters. Then, all of the samples are aligned from the left. Let's assume an example (from [7]) of the language $\{ababb, aabb, ababa, abc\}$, the alignment can be seen here:

(a)	(b)	(a)	(bb)	
(aa)	(bb)			
(a)	(b)	(a)	(b)	(a)
(a)	(b)	(c)		

Authors use the left-most alignment, because finding "the best" alignment for multiple sequences is an NP-hard problem. Finding it for several hundreds or thousands of samples in reasonable time is, therefore, not possible.

As the next step, a tree-shaped NFA (shown in (A) of Figure 3.3) is constructed from this alignment where, initially, block values are used for edges. If one column contains two different block values, the tree will contain an edge for both of them. Note that it does not matter how long concrete blocks are, but from which symbols they are composed from. As the next step, sequences of characters are used to construct loops within the automaton (shown in part (B) of Figure 3.3).

As mentioned before, the main drawback is the necessity to align all of the samples together. That might be problematic with large data sets. Another problem is that the left-most alignment would not be sufficient for more complex sentences. Finding a better alignment of multiple sequences together is too difficult to compute in reasonable time, though. More on this issue is described in following section.

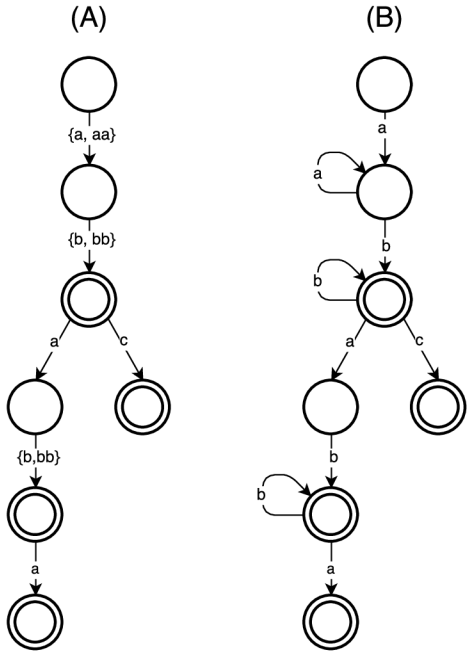

Figure 3.3: NFA (A) and DFA (B) generated by the method proposed in [7].

3.3 Sequence Alignment

In this section we are going to dive deeper into methods of aligning two sequences based on similarity of their parts. It is most commonly used in bioinfromatics for arranging DNA, RNA or protein sequences. Its goal is to find similar regions in two or more sequences, which may indicate some sort of relationships in the structure, and put these regions under each other. However, the scope of usage of these methods is beyond just bioinformatics. It is used in natural language processing or financial data processing.

Alignments of short and simple messages can be done easily by hand. However, the more complex they are, the more difficult it gets. Instead, automatic alignment algorithms have been designed. There are two types of alignment methods — *global* and *local*. Global alignment takes into consideration whole length of sequences in order to find the best alignment. Therefore, it is more suitable for sequences with similar length. Local alignment is better for sequences that differ in length. With local alignment we can find regions with high level of similarity in sequences which are not very similar in general.

Another classification of these methods is based on how many sequences they try to align. *Pairwise* methods, as the name suggests, attempt to find alignment of only two sequences. *Multiple Sequence Alignment (MSA)* methods are designed to find an alignment among three and more sequences. Computing MSA is very expensive in terms of both time and space, therefore these methods are usually used to find an alignment among three or four sequences.

Here is an example demonstrating why an alignment of two sequences might be useful. It makes further analysis easier and reveals patterns in a campaign:

Pelanggan	00000,	Kamu dapat 1 Pesan di Inbox-mu.
Plgn	Yth 11111,	Kamu dapat 1 Pesan Chatting.

As the example demonstrates, if corresponding segments are aligned under each other, it is easier for both humans and an algorithm to recognize a pattern within a campaign. One of the possibilities is to build a finite automaton based on this alignment. An example is shown in Figure 3.4. Also, a RE can be easily generated from such automaton.

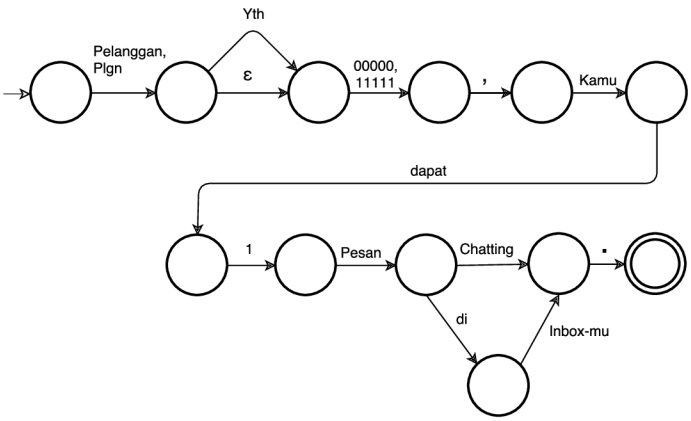

Figure 3.4: An example of a finite automaton generated from aligned messages.

3.3.1 Needleman-Wunsch Algorithm

Needleman-Wunsch Algorithm (NWA) [18] is designed to find a global alignment of two protein or nucleotide sequences using dynamic programming. However, the algorithm can be extended to make the procedure more general. This way the algorithm could support sequences of custom objects, rather than just protein/nucleotide sequences. In this section, the original version of the algorithm is presented. Both space and time complexity of this algorithm is $O(mn)$, where m/n is the length of the first/second sequence.

The algorithm uses two matrices — *score matrix and trace-back matrix.* It puts both sequences to headers of these matrices, each on one axis. It consists of three steps: (i) initialization of the score matrix (ii) computing scores and filling the trace-back matrix (iii) deducing the alignment from the trace-back matrix. The basic idea is to compare each possible pair, compute their scores (or quality), and find an alignment based on the best scores.

Initialization Step
First, several execution parameters have to be defined — *match reward, mismatch penalty and gap penalty.* If symbols in both row and column are the same, match reward is added. Otherwise mismatch penalty is used. Gap penalty is applied when a symbol aligns to a gap in the other sequence.

In the following example the match reward is $+1$, mismatch penalty -1 and gap penalty -2. Next, the score matrix T has to be constructed — one sequence is put into the first column, the other one in the first row as headers. The first value to be put into the matrix is 0 into the first empty cell $T(0,0)$ (excluding headers). Let us demonstrate the algorithm on two sequences $s_1 = TGGTG$ and $s_2 = ATCGT$. To access a symbol on the i-th position of a sequence s, the notation $s_x(i)$, where $x \in \{1,2\}$ is used.

Filling Score and Trace-back Matrices
The matrix is filled by traversal of the matrix row by row, computing the value of each cell at a time. A cell value is defined as the highest value computed from existing values of the left, top and top-left neighbor. Values coming from top or left represent gaps in the alignment. When it comes from the diagonal neighbor it represents the alignment of those two symbols. Here is a more precise definition of cell value computation:

$$T(i,j) = max \begin{cases} T(i-1,j-1) + \sigma(s_1(i), s_2(j)) \\ T(i-1,j) + gap\_penalty \\ T(i,j-1) + gap\_penalty \end{cases}$$

Where σ is a scoring function returning the *match_reward* if given symbols are the same, otherwise returns the *mismatch_penalty*:

$$\sigma(s_1, s_2) = \begin{cases} \text{match\_reward} & if\ s_1 = s_2 \\ \text{mismatch\_penalty} & otherwise \end{cases}$$

The first row and column are special cases. Their left (or top) and diagonal neighbors are not defined. Therefore, they can be easily pre-filled with multiplications of the gap penalty, as shown in Table 3.2a. The completely filled matrix is shown in Table 3.2b.

		T	G	G	T	G
	0	−2	−4	−6	−8	−10
A	−2					
T	−4					
C	−6					
G	−8					
T	−10					

(a)

		T	G	G	T	G
	0	−2	−4	−6	−8	−10
A	−2	−1	−3	−5	−7	−9
T	−4	−1	−2	−4	−4	−6
C	−6	−3	−2	−3	−5	−5
G	−8	−5	−2	−1	−3	−4
T	−10	−7	−4	−3	0	−2

(b)

Table 3.2: Filling of the score table. Sub-table (a) shows the pre-filling of the first row and column.

Each cell takes values from three neighbors, adds proper reward or penalty and then chooses the highest of them. For extracting an alignment, it is important to keep track from where each cell got its value. That is what the trace-back matrix is for. For each cell in the score matrix, it holds the information about the origin of its value. If two neighbors provide the same value, both sources are stored. This represents two equally good alignments. Table 3.3a shows from where each value in the score matrix comes from.

Tracing Back
The last step of producing an alignment is the trace-back phase. First, a path with the highest scores connecting the last (bottom-right) and the initial (top-left) cells is found. It is done by simply following the arrows from the last cell. If a cell on this path contains two or more arrows, more equally good alignments can be found. The path is shown in part (B) of Figure 3.3a. The alignment is constructed by following the path in reversed direction and applying these rules:

- *Diagonal arrow* – Symbols on corresponding positions from both sequences are aligned under each other. It is not important, whether there was a match or not.

		T	G	G	T	G
	x	←	←	←	←	←
A	↑	↖	←↖	←↖	←↖	←↖
T	↑	↖	↖	←↖	↖	←
C	↑	↑	↖	↖	←↖	↖
G	↑	↑	↖	↖	←	↖
T	↑	↖↑	↑	↖↑	↖	←

(a)

		T	G	G	T	G
	x	←	←	←	←	←
A	↑	↖	←↖	←↖	←↖	←↖
T	↑	↖	↖	←↖	↖	←
C	↑	↑	↖	↖	←↖	↖
G	↑	↑	↖	↖	←	↖
T	↑	↖↑	↑	↖↑	↖	←

(b)

Table 3.3: Sub-table (a) shows the trace-back table, where each cell keeps information about where it got its value from. Sub-table (b) shows the path from the last to the initial cell, which is used to create the final alignment.

- *Left arrow* – There will be a gap inserted to the sequence written on the side.
- *Vertical arrow* – There will be a gap inserted to the sequence written on the top.

Based on these rules, the following alignment of sequences s_1 and s_2 from the trace-back matrix is constructed:

\- T G G T G
A T C G T -

3.4 Genetic Algorithms

Another method of grammar induction is *genetic programming (GP)* — an evolutionary computing paradigm which performs a search in space of possible candidates trying to find the best.

It consist of a set of possible candidate solutions (called individuals). The whole set of individuals is called a population. Quality of each individual (how well it solves the problem) is computed by a *fitness function*. GP performs stochastic and heuristic solution-space exploration in order to find an individual with optimal fitness. In order to do so, an initial population is necessary. It usually consists of randomly generated individuals. GP execution is an iterative procedure in which (i) new individuals are created from the existing population (there are specific genetic operators such as crossover and mutation), (ii) adding new individuals to the population, (iii), eliminating individuals with the lowest fitness. This whole procedure is repeated until a certain condition is satisfied (number of iterations, an individual with sufficient fitness is found, etc.).

In case of grammar induction a candidate is an instance of desired language representation (a grammar, automaton, RE). Its quality (fitness) is based on how many samples of the input dataset it accepts and how many it rejects.

Many GP-based methods for grammar induction have been proposed in the past. Authors in [4] present an automatic method for RE extraction, where the input is only positive samples. Individuals in this case represent a candidate RE. They are encoded as trees, where each node represents a sub-part of the final RE. When creating a new candidate by crossover operator, some sub-trees are swapped. Similarly for the mutation operator, a certain

sub-tree is randomly modified, in attempt to access different parts of the searched solution space.

Whenever an individual which is not a valid RE generated it is removed from the population. Authors performed an interesting comparison between 70 human annotators and the proposed automatic extraction method. Based on their results they claim to be *highly competitive* with human capabilities. The implementation of the method is available as an online tool, together with several datasets and extraction tasks at [2]. A preview can be seen in Figure 3.5.

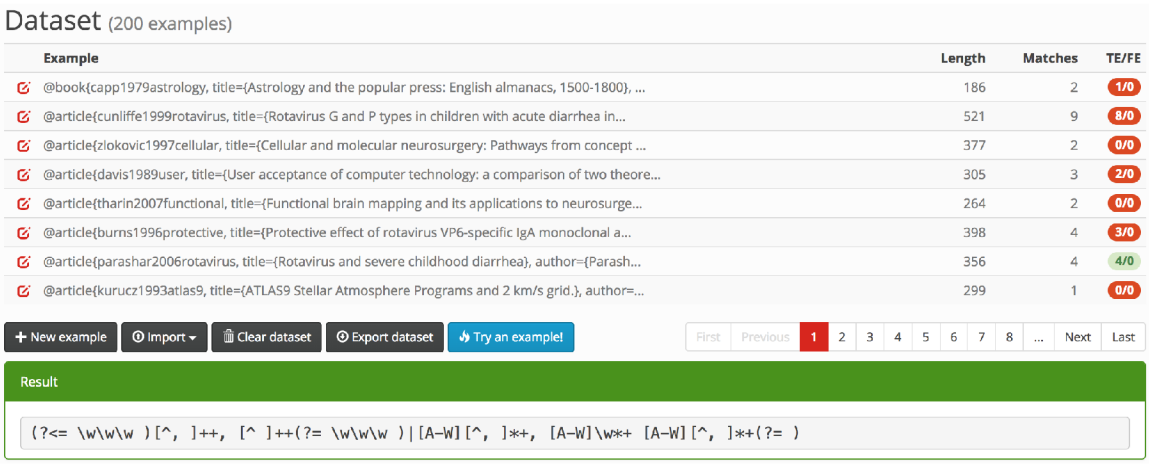

Figure 3.5: Online tool for GP search for learning regular expressions from examples (Source: [2]).

An advantage of GP is that it can be used to learn grammars more complex than regular. There is a method of learning context-free grammars from language examples in [23]. The method is successful to recognize patterns in parenthesis pairs and reflect it on the output. However, it is successful only with very basic examples. To move from the scope of regular languages even further, GP is also used in attempts to extract grammars from natural languages, as presented in [16].

To evaluate usability of GP — with respect to the goal of this work — there are several drawbacks. Firstly, it may take a long time to get results. In case of the method presented in [4], it took over 2 hours to get a result, when working with 200 instances of BibTeX entries. Secondly, the method requires many walk-through of the data set. To compute fitness of an individual, it needs to be compared with every data sample. The result of GP is just the best solution found, it is not guaranteed that it would produce a RE matching every message of a spam campaign. Another issue is the possible over-generalization. As it can be seen in Figure 3.5, the result is rather too general. Our goal is to propose an algorithm, which would be as specific to a certain campaign as possible.

Chapter 4

Design of the Algorithm

In this chapter we are going to detailly describe the proposed algorithm. But first, we are going to further specify the task it is meant to solve, and address issues it has to overcome. Then, we will focus on each crucial component of the algorithm and how they all cooperate in order to achieve the desired goal. In the end, we present overall architecture of the whole system.

4.1 Task Description

The goal is to design an algorithm which is able to reversely extract a template of a given SMS spam campaign, when only samples of the campaign are available. A template must be represented as a formal model — regular expressions (RE) are preferred. The length constraint of SMS messages (160 characters) can be taken into consideration for the final design.

A resulting template should match every message of a given campaign, but not messages from other campaigns. In case that not every message can be matched, a measure of quality of the output should be provided at the end (chance of mismatching a message from the same campaign).

The algorithm must be able to process large amount of data, as the volume of messages in campaigns can be very high. It should reliably process texts with different typing conventions, separators, etc.

4.2 Addressing the Challenges

There are several issues that need to be overcome in order to provide sufficient results. From high volume of data, to the fact that messages are intentionally crafted to be highly variable. Here is a list of these challenges.:

- **Volume and variety** — To design the algorithm and evaluate it, around 300 real SMS spam campaigns from around the world were used. The smallest campaign contains around 800 messages. The largest one has over 27 000 samples.
- **Over-generalization** — Produced templates should be able to identify campaigns from which they were extracted. Therefore, an extracted template should be as campaign-specific, to the source campaign, as possible. Consider an extreme case where the resulting RE describing a campaign would be `(^\w+ \w+ \w+ \d+$)` (a

sequence of three words and a digit separated by white-space characters). Even though it may match every message of the source campaign, it could not be used to identify the campaign.

- **Static/semi-static parts** — There might be parts of messages which appear to be static at the beginning, but turn out to be variable later (date, time, etc.).
- **Only positive data** — There are only genuine examples of each campaign available. The learning algorithm is not shown any counter-examples. The input file can, of course, contain noise (messages that do not belong to the campaign) and messages of other campaigns. However, they cannot be treated as negative examples, as none of them is helpful in defining structure of the language.
- **Quality of the output** — There are multiple ways in which a campaign can be formally described. Even if we assume to use only REs as the output, there are multiple semantically-equivalent representations. The goal is to provide campaign-specific, easy-to-read results.
- **Intentional confusion** — The algorithm has to assume that spammers put a lot of effort into confusing spam filters (intentional misspelling of words, etc.). These practices make extraction more difficult and they have to be overcome.

4.3 Output Format

We decided to use *regular expressions (RE)* as the mean of template representation. There are multiple reasons for that. As we consider a campaign to be a finite (therefore regular) language, REs are an appropriate tool. They are easier to read and manipulate with than finite automata. Thanks to their usage in many programs (such as *grep, awk, lex*), they are very well-known. In terms of further processing, REs are supported in almost every programming language.

There are, especially within programming languages, multiple implementations of REs with different capabilities. For example, REs implemented in UNIX `grep` overcome — in terms of descriptive power — their formal definition. Therefore, we need to specify which constructions are going to be used. As we want to make campaign descriptions simple and readable, the algorithm is restricted to use following constructions only:

- String literal — it can be wrapped within a RE group (parenthesis), or not
- Selection of string literals — such as `(a|b|c)`, also called a *choice group*
- Optional group indicator — symbol "`?`" indicating that the value can, but does not have to, be present
- Wildcard group — "`(.*?)`", non-greedy variant "consuming" as little as possible
- Type-specific wildcard groups — such as an integer wildcard "`(\d*)`", etc.

4.4 Extraction Method Design

In this section the general idea of the proposed template extraction algorithm is described. It is based on finding an alignment of two sequences. The alignment algorithm is a modified

version of Needleman-Wunsch algorithm described in Section 3.3.1. The modifications are explained in Section 4.4.4. Besides the alignment, there are several more key steps, such as tokenization and RE construction. All of them are also detailly described in this section.

4.4.1 Templates and Segments

Before demonstrating the proposed algorithm, two terms need to be clearly defined — *a template* and *a segment*. A template is an abstract representation of characteristic properties of a campaign. It is not a certain formal model, but models can be constructed from it. Therefore, it cannot be used to decide whether a message belongs to a campaign or not. It needs to be transformed first. The transformation process is not restricted in any way. Even multiple formal models (such as a RE and a finite automaton) can be constructed from a single template.

A template is an ordered collection of segments — typed sets of lexical tokens (described in 4.4.3). Segments are filled with data during the alignment process. Their types are helpful with deciding, where each token should be put. For example, a number in a message is more likely to be aligned with a numeric segment, rather than with a punctuation segment.

When a template is transformed to a formal model, each segment is transformed with it. Segment's type helps with handling the data inside (a number is transformed in a different way than a date). Here is a list of currently supported segment types:

- `TextSegment`
- `PunctuationSegment`
- `NumericSegment`
- `TimeSegment`
- `DateSegment`

Type of a segment is determined by its tokens. If a new token is added to a segment, but it is incompatible with other tokens in it, the segment is replaced by a general `TextSegment`.

4.4.2 Top-Level View on the Algorithm

The extraction procedure is demonstrated in Algorithm 3. Input of the algorithm is a list containing messages of a single campaign (clustering of messages into campaigns is not part of this work). The extraction procedure consists of several major steps — *tokenization, alignment, outlier detection, alignment merging and RE construction.* These steps will receive more attention in following sections.

An initial template is constructed from the first message of a campaign. It matches just the first message. Then, the algorithm performs incremental generalization of the template. Each message updates the template, so it matches all of the previously matched messages and the current message itself. After going through the whole campaign, a template (potentially) matches every message of the campaign.

Optionally, messages can be sorted decreasingly by the number of tokens they consist of, prior to the execution. It improves extraction results in certain cases. However, it requires to tokenize the whole campaign. Even messages which would not be tokenized otherwise. This feature is described more by an experiment in Section 6.3.2.

Algorithm 3: An incremental method of message template extraction

```
Tokens ← tokenize the first message of the data set
Template ← create template from Tokens
TemplateRegex ← generate regex from the template
foreach msg in the campaign do
    if msg matches TemplateRegex then
        continue

    Tokens ← tokenize msg
    Alignment ← align(template, tokens)
    Similarity ← compute similarity of the message and the template

    if Similarity < THRESHOLD_VALUE then
        Discard the alignment
        continue

    MergedAlignment ← mergeAlignment(Alignment)
    Template ← create new template from the merged alignment
    TemplateRegex ← generate regex from Template

OptimizedRegex ← Optimize template regex for presentation
return OptimizedRegex
.
```

During the generalization process, one message at a time is read from the input (whether it was presorted, or not). If the message matches the current template, nothing is changed and the message is skipped, because it is already covered by the template. Otherwise, an alignment between the template and message's token string is computed. The alignment tries to identify similar or related parts in the template and the message and pair them together. The alignment process is further described in Section 4.4.4.

Of course, some noise (a message which does not belong to the campaign) can be present in the campaign file. These messages have to be identified and removed, otherwise they would degenerate the final template. Similarity of the template and the current message is computed. If it is lower than a certain threshold, the alignment is discarded and the message is skipped. Otherwise, the message is accepted and alignment pairs are merged together — a new template is constructed. More on detecting outliers can be found in Section 4.4.6.

By merging an alignment we mean putting tokens of a message into template's segments to which they were aligned to. This step produces a new template. The new template is a generalized version of the old one in a way that the new one covers the previously uncovered message. A new RE is constructed from the updated template, so the next input message is compared with an accurate template. RE construction is described more in Section 4.4.8. By repeating this procedure for all messages, the final template will eventually match every message of the source campaign (except those excluded).

4.4.3 Tokenization

A token is a lexical unit of a message. It encapsulates a string literal (a certain part of a message), its type and possibly more meta-data. It is a unit of extraction (the smallest data instance). Every extraction-related algorithm works with tokens or higher structures composed from them (such as segments).

Each token type is defined by a RE (e.g. `\d+` for numbers). Tokenization is a process of splitting an input message into an ordered collection of tokens. However, considering the character of the data (in sense of our task), it is very difficult to define clear rules specifying token types. For example, let us assume the string `1011-2365`. Does "`-`" indicate a subtraction of two numbers, making it three separate tokens `"1011"`, `"-"` and `"2465"`, or is the whole sequence a single token indicating a serial code? In fact, both of these options can be valid, even within the same campaign.

It is impossible to define rules which would be valid in every possible case. Therefore, we tried to extract the most characteristic constructions occurring in real-world campaigns. Here is a list of rules that are used to extract tokens:

- numbers — Several types, such as decimal, integer, comma/dot-separated, etc.
- URLs — Basic representation of URLs and e-mails.
- punctuation
- time — It has a relatively consistent representation (there are not that many time formats).
- date — Only a subset of possible date formats, as there are too many.
- slugs — Constructions such as "*ab1-b4-33*"
- special characters — For example, Unicode characters (©, etc.).
- "any" — Any sequence of alphanumeric characters.
- white characters — They are not part of the extraction as the other types are, but it is important to keep the information, whether a token was preceded by a space, or not.

However, not each of these rules produces its own token type. For example, rules defined as "slugs" and "any" in the list, produce tokens of the same type. All types of tokens are shown in Figure 4.1 in form of a class diagram.

Spaces and other "white" characters do not make separate tokens. They are, however, extracted from the source file. Each token contains information whether it was preceded by a white-space character or not. Having this information is crucial for generating correct REs. A single missing space could make the whole RE incorrect.

4.4.4 Alignment of a Template and a Message

This is a crucial part of the algorithm. It aligns a template and an input message to identify their related parts, even though they are lexicographically different and in different positions. As a result, it helps detecting a general pattern of a campaign, such as static and variable parts.

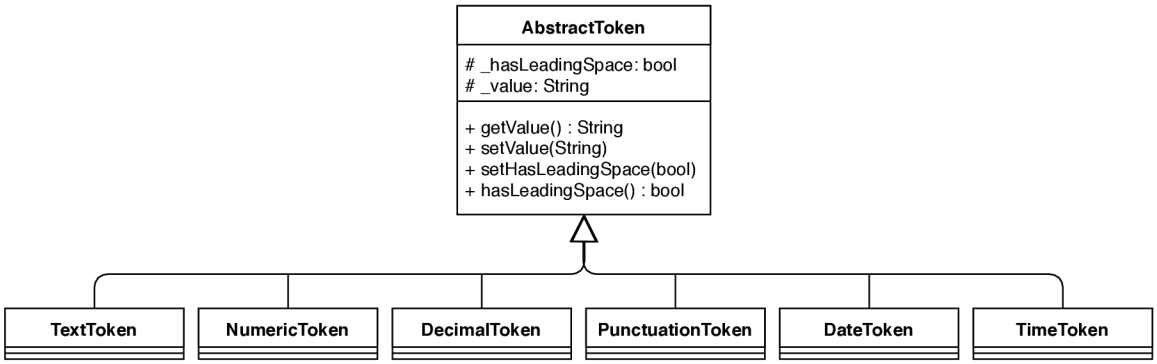

Figure 4.1: Types of tokens and their hierarchy.

An alignment is basically two sequences of the same length. Items on the same positions are identified to be related. Some items of these sequences can be empty, which indicates a gap in the alignment. An example of an alignment is shown in Figure 4.2. Finding alignments is performed using Needleman-Wunsch algorithm described in Section 3.3.1. However, in order to make it applicable for template extraction, it had to be modified in several ways.

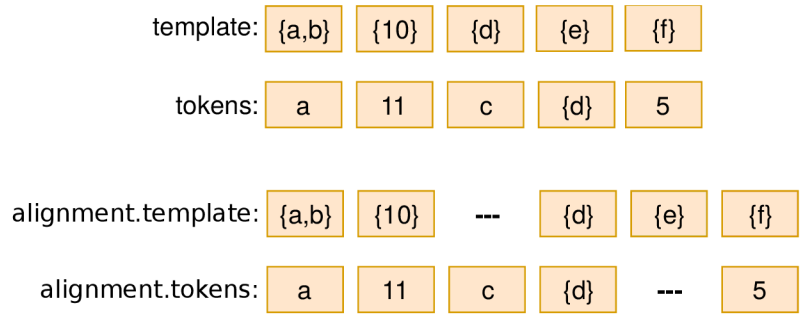

Figure 4.2: Representation of an alignment as a structure.

There are two major differences — the algorithm does not align just characters, and lexicographical variability is common, when spammers try to confuse spam filters. Instead of just characters, the algorithm aligns a template (a collection of sets of tokens) and a collection of tokens (strings with assigned types). However, the core principle of the method remains the same. Score and trace-back matrices are used, but a more sophisticated score-computing function had to be introduced.

The function has to be able to assign a score to *(segment–token)* pairs. Since a segment is a set of tokens, the score is computed as the maximum score of each *(segment-token–token)* pair. Formally defined as $score(Segment, token) = max(Subscores)$, where $Subscores = \{score(segmentToken,\ token) \mid segmentToken \in Segment\}$. The function *score* is defined in Algorithm 4.

The original algorithm works with three constants — *match reward, mismatch penalty and gap penalty.* However, when working with strings, it is more appropriate to use a measure of similarity, rather than just binary equality concept. A difference in a single character is not a reason to apply the full gap penalty. Especially when the variability of messages is intentionally high. Therefore, the proposed algorithm works with only two constants — *gap penalty* and *full match reward.*

Algorithm 4: Simplified version of computing scores for the score matrix in modified Needleman-Wunsch algorithm.

```
Function score(segment, token):
    subscores ← [ ]
    foreach segtkn in segment do
        if neither is TextToken and both are of the same type then
            Append EXACT_MATCH_VALUE to subscores
        else
            sim ← similarity(segtkn, token)
            Append EXACT_MATCH_VALUE*sim to subscores
    return max(subscores)

Function similarity(s1, s2):
    return 1 − normalized_levenshtein_distance(s1, s2)
```

The meaning of the gap penalty remains the same. Full match reward is the value applied when two strings are equal. This value is then multiplied with the similarity score, to reflect how much are those two strings different or similar. More similar pairs, therefore, will be rewarded more. The similarity of two tokens is computed using *Levenshtein similarity*[10][1].

Next to the similarity measure, token types are also taken into consideration. If two *non-textual* tokens are of the same type, their score is equal to the full match. This, for example, reflects relations between numbers. Two numbers can be lexicographically completely different (therefore low similarity), but semantically they should be aligned together. Another example is punctuation. Characters like `"."` and `","` are completely different, yet they can have the same meaning within a message. Also applies that *non-textual* tokens are more significant for alignments. If there is a punctuation token in both message and template (relatively close to each other), it is more likely that the best alignment is to put them under each other and fill the remaining space with gaps. These relations need to be emphasized by the score-computing function.

The relationship between *full match reward* and *gap penalty* is following. If their difference is too low, the algorithm prefers to create gaps in the alignment, or it puts unrelated, highly different tokens into the same segments, even though there is a better option a bit further in the sequence. If the difference is too high, the algorithm tries to align tokens and segments which are too far from each other in the alignment. This may result in too many gaps and optional segments. By experimenting, we discovered that the configuration $gap\_penalty = 1$ and $full\_match \in \{4, 5\}$ provides good results on real-world campaigns.

If a segment of a template is aligned with a gap, the segment is set to be optional. Tokens of this segment will not have to be present in every message. To properly catch and handle this situation is very important. Having an optional segment set as mandatory could lead to mismatching many messages, which would be matched otherwise.

[1] Similarity is computed from Levenshtein distance by formula: $similarity = 1 - normalized\_distance$

4.4.5 Outliers Detection

Input of the algorithm should be a file containing a single campaign. However, it would be naive to expect this requirement to be satisfied in any case. Some noise can always be present. For this reason the algorithm contains an outlier detection mechanism.

The idea is to compute overall similarity between the aligned template and the token string of current message. Algorithm 5 shows how the general similarity is computed. It is very similar to the approach shown in Algorithm 4 with several differences. It does not multiply the similarity measure with any constant value and it "punishes" type incompatibilities and gaps more.

If a computed similarity is less than a certain value (`SIMILARITY_THRESHOLD`), this message is discarded and skipped. The threshold is set — based on experiments — to 0.2. However, it can be easily modified.

Algorithm 5: The algorithm for computing general similarity of an alignment.

```
similarities ← [ ]
foreach (segment, token) pair of the alignment do
    if segment or token is a gap then
        Append −1 to similarities
    else
        Append similarity(segment, token) to similarities
OverallSimilarity ← sum(similarities)/size(similarities)

Function similarity(segment, token):
    subscores ← [ ]
    foreach segtkn in segment do
        if neither is TextToken and both are of the same type then
            Append 1 to subscores
        else if segtkn and segment have different types then
            Append −1 to subscores
        else
            Append levenshtein_similarity(segtkn, token) to subscores
    return max(subscores)
```

Detection of outliers can be very unclear and subjective. See the example shown in Figure 4.3 with focus on the 9th line. Majority of the message matches perfectly with other messages (green parts) or follow similar patterns (e.g. aligned numbers). However, by the end there are some parts of the message missing, text tokens begin to differ and it even puts a number where a text literal is expected. The decision is, as it was said, very subjective — is this message an outlier, or not? In the example shown, there is only one such message, but in the whole campaign this pattern repeats (less frequently than the major pattern).

A clearer example of an outlier is shown Figure 4.4. Here it is more obvious that a high-scoring alignment cannot be found. There are only few parts that match precisely, compared to all the parts that are not similar at all.

\(a\) (.*?) \, (Ainda|ainda) nao recebemos seu contato para tratar de seu beneficio disponivel (\,|\.) (entre|Entre) em contato no 0800 ([+-]?\d+) (\,|\.)? (aguardamos|Aguardamos)? (ou|o) (seu|Whats) (contato|1111122222) \.														
(a)	N	,	ainda	nao recebemos seu contato para tratar de seu beneficio disponivel	,	entre	em contato no 0800	1111	,	aguardamos	o	seu	contato	.
(a)	C	,	Ainda	nao recebemos seu contato para tratar de seu beneficio disponivel	.	Entre	em contato no 0800	7777	.	Aguardamos	o	seu	contato	.
(a)	J	,	Ainda	nao recebemos seu contato para tratar de seu beneficio disponivel	,	entre	em contato no 0800	8888	.	Aguardamos	o	seu	contato	.
(a)	O	,	Ainda	nao recebemos seu contato para tratar de seu beneficio disponivel	,	entre	em contato no 0800	8888	.	Aguardamos	o	seu	contato	.
(a)	M	,	ainda	nao recebemos seu contato para tratar de seu beneficio disponivel	,	entre	em contato no 0800	1111	,	aguardamos	o	seu	contato	.
(a)	J	,	Ainda	nao recebemos seu contato para tratar de seu beneficio disponivel	.	Entre	em contato no 0800	7777	.	Aguardamos	o	seu	contato	.
(a)	E	,	Ainda	nao recebemos seu contato para tratar de seu beneficio disponivel	,	entre	em contato no 0800	2222	.	Aguardamos	o	seu	contato	.
(a)	I	,	Ainda	nao recebemos seu contato para tratar de seu beneficio disponivel	.	Entre	em contato no 0800	7990	.	Aguardamos	o	seu	contato	.
(a)	L	,	Ainda	nao recebemos seu contato para tratar de seu beneficio disponivel	.	Entre	em contato no 0800	4444			ou	Whats	1111122222	.
(a)	N	,	Ainda	nao recebemos seu contato para tratar de seu beneficio disponivel	.	Entre	em contato no 0800	7777	.	Aguardamos	o	seu	contato	.

Figure 4.3: Demonstration of subjectivity of outliers detection. Does the 9th message belong to the campaign, or not?

	Bpk\/Ibu Yth\,trima kasih servis Toyota (B2|B4|B3|B1) tgl 14\.05\.2016 di Auto2000 (G|R|B)? (S|B) \. Jika ada keluhan\, hub (M)? (J|S|R) (P|A)?								
1	Bpk/Ibu Yth,trima kasih servis Toyota	B1	tgl 14.05.2016 di Auto2000	G	S	. Jika ada keluhan, hub		S	P
2	Bpk/Ibu Yth,trima kasih servis Toyota	B2	tgl 14.05.2016 di Auto2000	B	B	. Jika ada keluhan, hub		J	P
3	Bpk/Ibu Yth,Toyota BX akan jatuh tempo utk Service Berkala 10000km. Nikmati CC CC J di Auto2000.S&K* berlaku. Utk Booking hub 061-8888000								
4	Bpk/Ibu Yth,Toyota BX akan jatuh tempo utk Service Berkala 30000km. Nikmati CC C J di Auto2000.S&K* berlaku. Utk Booking hub 061-8888000								
5	Bpk/Ibu Yth,trima kasih servis Toyota	B3	tgl 14.05.2016 di Auto2000	R	B	. Jika ada keluhan, hub		R	A
6	Bpk/Ibu Yth,trima kasih servis Toyota	B4	tgl 14.05.2016 di Auto2000		S	. Jika ada keluhan, hub	M	S	

Figure 4.4: A clearer example of a campaign containing outliers.

4.4.6 Merging an Alignment

After it is decided that the currently processed message belongs to the campaign, the current template can be appropriately generalized, so it matches the message. The generalization is done by merging the previously computed alignment.

Merging of an alignment is a simple process of extending template's segments with tokens they were paired with. Since segments are sets, no duplicates are added. There are three situations that need proper handling:

- Gaps within alignments:
  - In the message line — the corresponding segment is set to be optional.
  - In the segment line — new segment (of appropriate type) containing the corresponding token is created and inserted on the specific position.
- Type-compatibility of segments:
  - Newly added token is type-compatible with the segment to which it is inserted — nothing special happens.
  - Token and segment are type-incompatible — new segment of a more general type is created and inserted on the correct position. All of the old and new tokens are inserted to this segment.
- Token's attributes — if a token of the same value is already in the segment, information about preceding spaces needs to be properly copied.

By merging the alignment the algorithm approximates construction of *Multiple Sequence Alignment*, which is otherwise very difficult to compute, as described in Section 3.3. Visual demonstration of the merging process is shown in Figure 4.5. It shows all possible cases of merging a segment and a token (or a gap). Each case is highlighted by a different color.

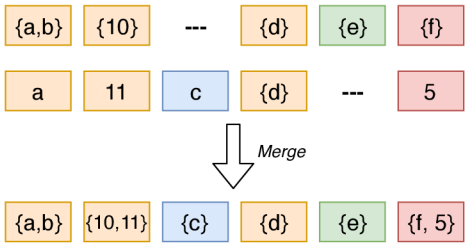

Figure 4.5: A diagram demonstrating the merging process of segments and tokens. There are following possible cases of merging: segment-token (both type compatible/incompatible), segment-gap, gap-token.

4.4.7 Regular Expression Construction

A template is just an abstract representation of a campaign and its characteristic properties. Before it can be used to decide whether it matches a message, or not, it needs to be transformed to a certain formal model — a RE expression in our case. However, REs are not generated directly from templates. A mid-product is introduced — so-called `RegexGroups`. Each segment corresponds to a single `RegexGroup` instance. Their goal is to simplify manipulation with segments during the transformation, by reducing the number of distinct values and types which segments contain. Also segments' attributes, such as optionally and leading spaces, are reflected to newly created groups. The transformation proceeds by following rules:

- If a segment has only one token → construct a `SingletonGroup`.
- Else if the number of distinct tokens is smaller than a given limit → construct a `ChoiceGroup`.
- Else (the number is higher than the limit) → build a `WildCardGroup`.

However, it is not groups' responsibility to transform themselves into REs. This responsibility is given to so-called `RegexGenerators`. They take each group and build a corresponding RE from it. The final RE is constructed by merging each groups' RE. Generators need to properly reflect optionality and leading spaces of each segment. An invalid RE could be generated otherwise. A class diagram of the RE construction sub-system is shown in Figure 4.6.

4.4.8 Regular Expression Optimization

The RE optimization process is present to increase readability of the result for the user. Therefore, it only affects *"visual"* aspects of REs. Considering that every segment gets transformed to a corresponding group and then a RE, a situation where two REs cover each other can occur. Therefore, it is unnecessary to have both of them in the result.

Let us assume an example of the "`(.*?)`" wildcard. It is applied whenever a segment contains too many distinct tokens. Obviously, two consequent segments can be transformed this way, giving "`(.*?)(.*?)`" as a result. However, in the semantics of REs, the latter wildcard completely covers potential matches of the first one, as it is non-greedy. Therefore, it can be removed.

There are three main optimization procedures aiming to improve overall readability of REs:

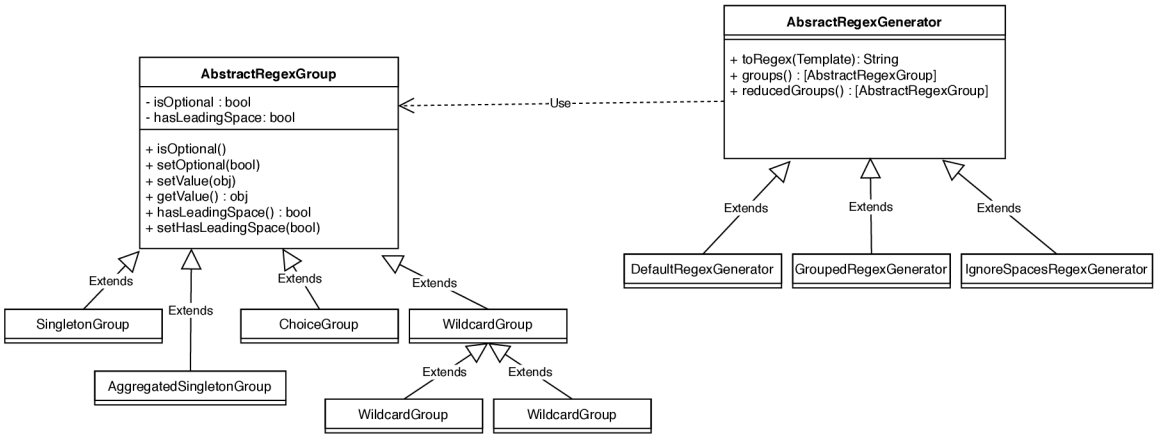

Figure 4.6: Class diagram of RegexGroups and RegexGenerators.

- Merging sequences of `(.*?)` wildcards,
- merging sequences of `(.*?)` wildcards interspaced by optional groups,
- merging sequences of static singletons (mandatory single-token segments).

These three optimization steps are demonstrated in Figure 4.7. The original RE got simplified from 9 to only 4 groups while maintaining the same semantics. However, it is not just readability that the optimization process improves. It also helps visualization of results. The `(.*?)` wildcard is non-greedy (it "consumes" as little as possible). Therefore, in a sequence of two such wildcards, the first one would match an empty string. Having such constructions right next to each other is unnecessary.

Figure 4.7: Demonstration of RE optimization.

There are two more RE optimization steps present. They are described separately, because they are left optional in the tool's implementation. Implicitly, they are turned off. They are also related to wildcard groups. This time, however, they consume optional groups right in front and behind them, until a mandatory group is found.

The need for these optimizations came up during experimenting with real-world campaigns. Personal names appear in most of the campaigns. Sometimes they consist of 2 parts, but sometimes they can have up to 4 or 5 parts. Let us assume an example from Figure 4.8. Most names consist of two parts and thanks to their variety, a wildcard is constructed. However, if some messages contain a name with three parts, the extra part will remain as a separate group in the result. In this case an extra optimization is in place (shown in the bottom part of Figure 4.8).

However, there are cases in which these second-step optimizations are rather harmful. An example of the same, but slightly modified, campaign is shown in Figure 4.9. The period character from the second static part was removed from some messages, which makes it a separate optional RE group. This group is directly attached to the wildcard

	tgl 14\.05\.2016 di Auto2000	(E|I|A|C)?	(.*?)	(Z|G|K|M)?	\. Jika ada keluhan\,
1	tgl 14.05.2016 di Auto2000	A	B		. Jika ada keluhan,
2	tgl 14.05.2016 di Auto2000	C	F		. Jika ada keluhan,
3	tgl 14.05.2016 di Auto2000	E	FF	G	. Jika ada keluhan,
4	tgl 14.05.2016 di Auto2000		H	Z	. Jika ada keluhan,
5	tgl 14.05.2016 di Auto2000	I	J	K	. Jika ada keluhan,
6	tgl 14.05.2016 di Auto2000		L	M	. Jika ada keluhan,

	tgl 14\.05\.2016 di Auto2000	(.*?)	\. Jika ada keluhan\,
1	tgl 14.05.2016 di Auto2000	A B	. Jika ada keluhan,
2	tgl 14.05.2016 di Auto2000	C F	. Jika ada keluhan,
3	tgl 14.05.2016 di Auto2000	E FF G	. Jika ada keluhan,
4	tgl 14.05.2016 di Auto2000	H Z	. Jika ada keluhan,
5	tgl 14.05.2016 di Auto2000	I J K	. Jika ada keluhan,
6	tgl 14.05.2016 di Auto2000	L M	. Jika ada keluhan,

Figure 4.8: An example of a real campaign in which more optimization is in place (upper part). And how does it look after optimizing optional groups before/after wildcards.

group and, therefore, it gets consumed during the optimization. In this case it would be more appropriate to exclude this group from the optimization.

	tgl 14\.05\.2016 di Auto2000	(.*?)	Jika ada keluhan\,
1	tgl 14.05.2016 di Auto2000	A B	Jika ada keluhan,
2	tgl 14.05.2016 di Auto2000	C F	Jika ada keluhan,
3	tgl 14.05.2016 di Auto2000	E FF G	Jika ada keluhan,
4	tgl 14.05.2016 di Auto2000	H Z.	Jika ada keluhan,
5	tgl 14.05.2016 di Auto2000	I J K.	Jika ada keluhan,
6	tgl 14.05.2016 di Auto2000	L M.	Jika ada keluhan,

Figure 4.9: An example of a real campaign in which the second set of RE optimization can be rather harmful. The period character would be, in this case, better as a standalone optional group.

Descriptive power of REs before and after optimizations is the same. Therefore, it is not necessary to perform them in every step of template generalization. Doing so right before the resulting RE is presented to the user is sufficient. Also, it can be completely turned off without affecting the number of matched/mismatched messages.

4.4.9 Properties of the Design — Pros and Cons

There are several major advantages, as well as drawbacks, to this approach of template extraction. It is important to mention that the properties described bellow apply mostly on real-world examples. It is always possible to construct an artificial example in which the listed advantages will not occur.

Firstly, it is important to understand that correctness of a RE on the output is difficult to measure. There is, potentially, an infinite number of REs describing the same campaign. Our goal is to provide a simple, easy-to-read result. However, readability can be very subjective. Therefore, correctness of the result is also very subjective. Sometimes it might be better to exclude some messages from matching the RE, in exchange for better readability. It is impossible to draw a clear line here. What we can say is that REs on the output do not suffer from so-called *exponential blow-up*, as they do when they are generated from finite automata.

The algorithm is designed in a way that complex and time-consuming operations (e.g. finding an alignment) are executed only when the current message does not match the template. Mismatches are expected to happen more frequently at the beginning of execution. However, as the template becomes more general, it begins to match more and more messages in the campaign (mismatches become less frequent). This way even large campaign (with tens of thousands of messages) can be processed relatively fast.

Since the template is generalized only when a mismatch happens and it is modified only to match the current message, the resulting template is very specific to the source campaign. Therefore, an extracted template can be (in most cases) used to identify the campaign from which it was generated. It also makes adding new messages to a campaign possible without recomputing the whole template.

From this advantage — the incremental template generalization — comes the biggest drawback. The consequence is, that the first message read has the biggest influence on the output. If the first message happens to be a noise, the result will probably turn out incorrect. This attribute of the algorithm should be removed in the future. One of the solutions would be extracting multiple templates from a single campaign. This way unwanted messages would cluster in their own templates, while the majority of messages would be matched to the main template.

A token is a unit of data manipulation in the algorithm. It allows very fine-grained template extraction. However, in combination with the inner representation of templates, it leads to loss of information about token sequences. For two neighboring segments containing multiple values, it is difficult to say which combinations of tokens (if any) occurred on the input. This issue is demonstrated in Figure 4.10. Part (1) shows a case in which a RE `"(sr(a)|sr\ a)\."` would be a better, more readable result. Case (2), on the other hand, shows a situation where the fine-grained approach is beneficial. It prevents replacing the whole part of the message by a wildcard (because there would be too many distinct values) and it provides more readable result.

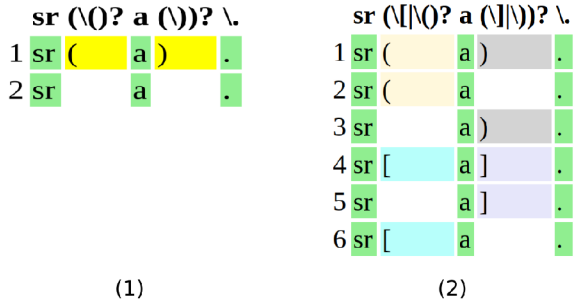

Figure 4.10: Demonstration of the token sequence disconnection problem. In case (1) fine-grained result decreases readability of the RE. In case (2) it, on the other hand, makes clearer result and prevents replacing this part with a wildcard.

The algorithm will always, by its nature, learn a template, because there can always be an alignment found. Even though it is incorrect and does not reflect real characteristics of the campaign, some alignment be can always found. Therefore, the process has to be restricted by the outlier detection mechanism. Otherwise, in case of mixed campaigns, the result would be a combination of both campaigns.

Segments with higher volume of distinct tokens are mostly replaced with the "`(.*?)`" wildcard. It is not an approach helping REs to be as campaign-specific as possible. Wildcard

groups are a trade-off between readability and accuracy. It also makes REs more flexible in case of unexpected value.

Lastly, the option of presorting messages by the number of their tokens helps to improve results in some cases. However, It requires all the messages to be tokenized prior to the extraction process. Even messages which would be covered by the template and, therefore, skipped. It also makes slightly less obvious how did each message affect the template, because messages are not processed in the visible order.

4.5 Architecture of the Tool

The template extraction process is an essential part of the tool, but it is not the only thing. After a template is retrieved, a validation process and result presentation follows. However, these processes can be logically separated. Therefore, their architectures are also presented separately, even though they share multiple components. The top-level architecture of the template extraction sub-system, in form of a class diagram, is shown in Figure 4.11.

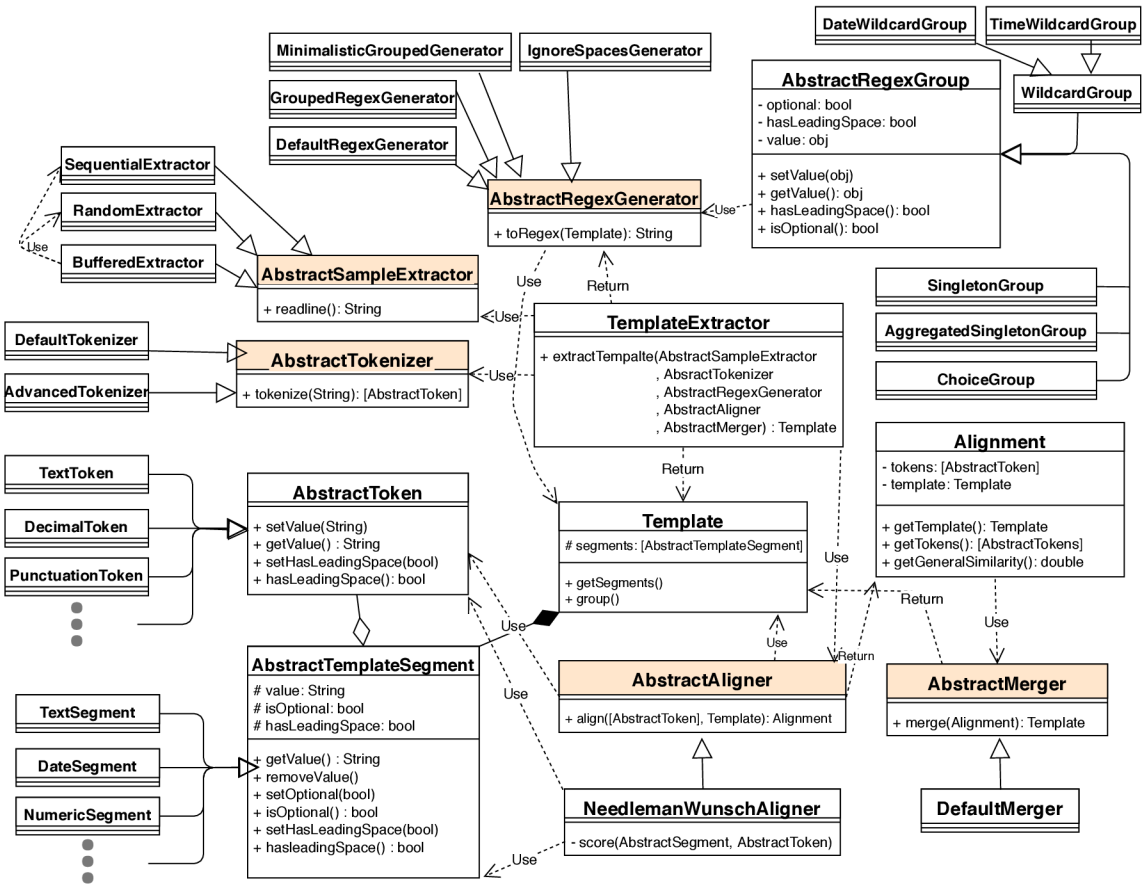

Figure 4.11: Class diagram of the template extraction sub-system. Highlighted classes are essential components for extraction. It demonstrates how customizable the sub-system is, because every concrete component is hidden behind an interface/abstract class.

The main class is `TemplateExtractor` with its `extractTemplate(...)` method. The diagram shows, how customizable the system is. Highlighted classes represent crucial components — sample extractor, tokenizer, aligner, merger and RE generator. All of them

are abstract and contain minimal interface required from concrete classes. Thanks to this design, it is very easy to modify behavior of `TemplateExtractor` or to extend the whole sub-system. All it takes is to implement the desired behavior in a new class which implements the necessary interface. Then connecting an instance of this class to the system. Interfaces guarantee compatibility with other components.

4.5.1 Validation Sub-System

Validation is a standalone sub-system within the whole tool. It takes a RE (as a string) on the input. On the output it tells how many messages of given campaign match the template and how many do not. Optionally, other output-generating components can be connected to produce additional outputs.

Extraction and validation sub-systems share some components (such as sample extractors), but they do not depend on each other in any way. A template can be extracted without further validation, as well as an existing RE can be just validated on a campaign.

Architecture of the validation sub-system (again in form of a class diagram) is shown in Figure 4.12. Also this sub-system is easy to modify or extend. There are two built-in validators available — *sequential* and *random*. Both of them use `SampleExtractor` classes to retrieve a message from a campaign.

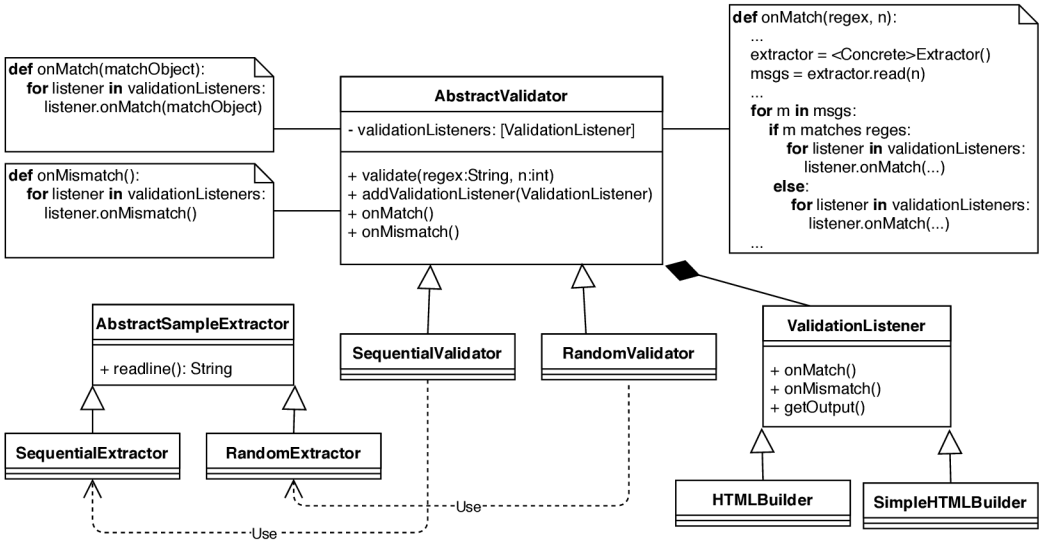

Figure 4.12: Architecture of the validation sub-system.

`ValidationListeners` are components which can be connected to the validation process. Every time a match or a mismatch occurs, these components are notified about the result. Each `ValidationListener` can react to the result differently. So far, `HTMLBuilder` has been introduced. It constructs HTML output in order to visualize validation results. Most of the examples presented in this document are generated using this component. Each time a message mis/matches the template, a mis/match callback method of each listener is executed. More Validation listeners can be introduced in the future for different visualization methods, or data processing in general.

4.5.2 Extending the System

The architecture makes extending system's behavior relatively simple. Each crucial part of the extraction sub-system, can be implemented in a different way, because all of them are hidden behind interfaces. One of the examples is adding support for binary sequences. They differ significantly from regular messages, therefore a new tokenizer, which respects patterns in binary data, would have to be implemented (potentially new token types). If it requires updates in alignment rules, a new aligner can be created and plugged to the existing environment.

As long as the new component respects the corresponding interface, there should be no problem in extending the system. Interfaces guarantee compatibility of components and make them all work together.

Chapter 5

Implementation

In this chapter, we briefly describe implementation of the template extraction tool based on the algorithm proposed in Chapter 4. We are going to focus on technologies used to develop the tool and its features.

The "working" name of the tool is `tempex` (it stands for Template Extractor). It is implemented in `python 3` (version `3.5` was used for development). One of the main reasons why python was selected is that it makes text processing, which is a major portion of this work, much easier than it is in other (mostly statically typed) languages. It also allows fast development, quick prototyping, supports *OOP paradigm*[1] and has cross-platform support. There are several other dependencies:

- `numpy`[19] — A scientific computing library used for basic matrix support.
- `PLY`[1] — Implementation of lex and yacc parsing tools for Python. However, only `Lex` part is used to implement input message tokenizer. Support of this library is only required when `AdvancedTokenizer` is used to tokenize messages.
- `distance` library[2] — Contains python implementation of Levenshtein distance (developed using v0.1.3).
- `PyQt5 (v5.10.0`[3]`)`[3] — Binding of the `Qt` GUI toolkit for python. It is used to build a simple graphical interface for viewing and modifying outputs of the core program.

5.1 Modules and Packages

There are several modules and packages from which the program is built. Each of them has its own purpose. This section provides a brief description of all of them. If some parts need to be described in greater detail, there will be a separate section provided for them.

Here is a list of modules in (partial) logical order of how they are used during the template extraction process:

- `tempex` — Separates the class in control of the template extraction process. The class is called `TemplateExtractor`.

[1] OOP stands for *Object Oriented Programming.*

[2] `https://pypi.org/project/Distance/`

[3] Note that there were some issues considering HTML rendering when using versions slightly higher.

- `redef` — This module simply contains definitions of several REs which are used by multiple modules.
- `io.input` — It wraps methods of input message extraction. Two main approaches are *sequential* and *random*. Sequential reads given number of message from the beginning of the source file, while random selects samples randomly.
- `tokenize` — It contains classes for representing tokens (such as *numeric, textual, punctuation, time, date*) and so-called *tokenizers*, classes dedicated for turning input messages into collection of tokens.
- `template` — It encapsulates implementation of *template* and its *segments*.
- `align` — The module contains methods for aligning a template and a collection of tokens, as well as tools for merging the aligned sequences together.
- `regex` — A set of classes for generating valid REs from abstract templates. There are multiple methods for different REs present.
- `validation` — A module designed for validation of extracted REs.
- `io.output.html` — Provides an interface for generating HTML output during the validation process.

5.2 Regular Expression Generators

Output of the extraction process is an abstract representation of campaign's characteristics. It is not a specific formalism. Thanks to that, it is not limited in the output presentation. Even though the tool provides only REs for template presentation, there are several types of REs constructed from the same templates by different `RegexGenerator` classes. Here is their list and explanation:

- `DefaultRegexGenerator` — It creates a valid RE, but it does not put non-variable parts into python RE group (parenthesis syntax). These parts do not get extracted during validation. Therefore, it is not a good idea to use a RE, generated by this generator, for validation with HTML output. Static parts will not be shown. However, the overall readability is better, REs are more compact and good for presentation to humans.
- `IgnoreSpacesRegexGenerator` — It has similar rules as `DefaultRegexGenerator`, but ignores leading spaces of each segment. Therefore, the output might be incorrect for validation. It is used, for example, in HTML table headings, where additional spaces make it more confusing.
- `GroupedRegexGenerator` — Every part of a message is captured in a named RE group (in Python `(?P<name>RE)` syntax). This makes REs difficult to read, but crucial for validation of a campaign with HTML output, because every part of a message is captured in one of the groups.
- `MinimalisticGroupedRegexGenerator` — Similar to `GroupedRegexGenerator`, but it uses RE groups without names (simply "`(...)`" syntax).

5.3 Output Methods

A support tool would be useless without a way to present its results. Our extraction tool offers several ways of presenting results to users. The tool is designed to be mostly used in a command line, where users can use variety of parameters, through which they can modify behavior of the program. See Appendix A for details on program execution.

For this reason, we focused on presenting results using the *command line interface (CLI)*. It is demonstrated in Figure 5.1. It is very simple and it provides users with an extracted template and information about how many messages of a campaign does the template match.

```
^(.*?)\,\s*confira\s*a\s*CAMPANHA\s*PRIMAVERA\s*de\s*DESCONTOS\s*(ESB|ESBELTY|JB|HOT)?\s*(CR|POINT|PRO)?\.\s*
Condicoes\s*INCRIVEIS\s*p\/\s*liquidar\s*seu\s*contrato\!\s*ligue\s*0800\s*111\s*1234\s*ou\s*WhatsApp\s*11\s*
(45678\-55555|99999\-3333)\.\s*$
Read: 16
Matched: 16
```

Figure 5.1: The basic output method — command line interface. It provides users with the extracted template and information about how many messages of the campaign it matched.

CLI can be also used to generate HTML output, which is a visualization of the validation process. Each column represents a RE group. Groups have different colors to make patterns in campaigns more obvious. All static groups (parts that are present in every message) are green. Optional single-token parts are bright yellow. In choice groups, each option has its own color assigned. And finally all wildcard groups are red. HTML output of validating a real-world campaign is shown in Figure 5.2.

Regex: ^(.*?)\, confira a CAMPANHA PRIMAVERA de DESCONTOS(JB|ESBELTY|HOT|ESB)?(CR|POINT|PRO)?\. Condicoes INCRIVEIS p\/ liquidar seu contrato\! ligue 0800 111 1234 ou WhatsApp 11(99999\-3333|45678\-55555)(\.)?\s*$

	(.*?)	\, confira a CAMPANHA PRIMAVERA de DESCONTOS	(JB|ESBELTY|HOT|ESB)?	(CR|POINT|PRO)?	\. Condicoes INCRIVEIS p\/ liquidar seu contrato\! ligue 0800 111 1234 ou WhatsApp 11	(99999\-3333|45678\-55555)	(\.)?
1	ALEX	, confira a CAMPANHA PRIMAVERA de DESCONTOS	JB	CR	. Condicoes INCRIVEIS p/ liquidar seu contrato! ligue 0800 111 1234 ou WhatsApp 11	45678-55555	.
2	VAND	, confira a CAMPANHA PRIMAVERA de DESCONTOS	JB	CR	. Condicoes INCRIVEIS p/ liquidar seu contrato! ligue 0800 111 1234 ou WhatsApp 11	45678-55555	.
3	ELIS	, confira a CAMPANHA PRIMAVERA de DESCONTOS	ESB		. Condicoes INCRIVEIS p/ liquidar seu contrato! ligue 0800 111 1234 ou WhatsApp 11	45678-55555	.
4	WER	, confira a CAMPANHA PRIMAVERA de DESCONTOS	JB	CR	. Condicoes INCRIVEIS p/ liquidar seu contrato! ligue 0800 111 1234 ou WhatsApp 11	45678-55555	.
5	GRA	, confira a CAMPANHA PRIMAVERA de DESCONTOS		PRO	. Condicoes INCRIVEIS p/ liquidar seu contrato! ligue 0800 111 1234 ou WhatsApp 11	45678-55555	.
6	REN	, confira a CAMPANHA PRIMAVERA de DESCONTOS		PRO	. Condicoes INCRIVEIS p/ liquidar seu contrato! ligue 0800 111 1234 ou WhatsApp 11	45678-55555	.
7	ADA	, confira a CAMPANHA PRIMAVERA de DESCONTOS	JB	CR	. Condicoes INCRIVEIS p/ liquidar seu contrato! ligue 0800 111 1234 ou WhatsApp 11	45678-55555	.
8	JUC	, confira a CAMPANHA PRIMAVERA de DESCONTOS		PRO	. Condicoes INCRIVEIS p/ liquidar seu contrato! ligue 0800 111 1234 ou WhatsApp 11	45678-55555	.
9	DON	, confira a CAMPANHA PRIMAVERA de DESCONTOS		PRO	. Condicoes INCRIVEIS p/ liquidar seu contrato! ligue 0800 111 1234 ou WhatsApp 11	45678-55555	.
10	LES	, confira a CAMPANHA PRIMAVERA de DESCONTOS	JB	CR	. Condicoes INCRIVEIS p/ liquidar seu contrato! ligue 0800 111 1234 ou WhatsApp 11	45678-55555	.
11	NID	, confira a CAMPANHA PRIMAVERA de DESCONTOS	JB	CR	. Condicoes INCRIVEIS p/ liquidar seu contrato! ligue 0800 111 1234 ou WhatsApp 11	45678-55555	
12	JUN	, confira a CAMPANHA PRIMAVERA de DESCONTOS	ESB		. Condicoes INCRIVEIS p/ liquidar seu contrato! ligue 0800 111 1234 ou WhatsApp 11	45678-55555	.
13	ALI	, confira a CAMPANHA PRIMAVERA de DESCONTOS	HOT	POINT	. Condicoes INCRIVEIS p/ liquidar seu contrato! ligue 0800 111 1234 ou WhatsApp 11	99999-3333	.
14	WEL	, confira a CAMPANHA PRIMAVERA de DESCONTOS	HOT	POINT	. Condicoes INCRIVEIS p/ liquidar seu contrato! ligue 0800 111 1234 ou WhatsApp 11	99999-3333	.
15	WIL	, confira a CAMPANHA PRIMAVERA de DESCONTOS	ESBELTY		. Condicoes INCRIVEIS p/ liquidar seu contrato! ligue 0800 111 1234 ou WhatsApp 11	45678-55555	.
16	SIL	, confira a CAMPANHA PRIMAVERA de DESCONTOS	JB	CR	. Condicoes INCRIVEIS p/ liquidar seu contrato! ligue 0800 111 1234 ou WhatsApp 11	45678-55555	.

Figure 5.2: HTML output of the tool. It shows both the extracted template and the validation process visualization. Green columns are static and do not change in any message, multi-color columns are choice groups, where each option has its own color, and red ones are wild cards.

5.4 Graphical Interface

The tool also provides a very simple user interface (UI) implemented with `PyQt5`. Both template extraction and validation processes can be performed in it. Probably the biggest advantage of using the UI is the possibility to modify the extracted RE, re-execute validation process and see the difference. The UI and modification of the RE is demonstrated in Figure 5.3. In the example, it is obvious that the first group should be consumed by the following wildcard. However, this process is difficult to automatize, because in some cases this behavior is desired. Therefore, there is a tool for easy modification of the final RE.

The UI simply visualizes HTML generated during the validation process, similarly as in Section 5.3. However, there are couple of drawbacks. All of them are caused by the fact that the UI is intended to work with purely textual RE, not an abstract template. Therefore, there is less information to use. The most obvious drawback is that the group-coloring mechanism is less sophisticated. Here, each group gets its own color, without checking if it is a static group, a choice group, etc.

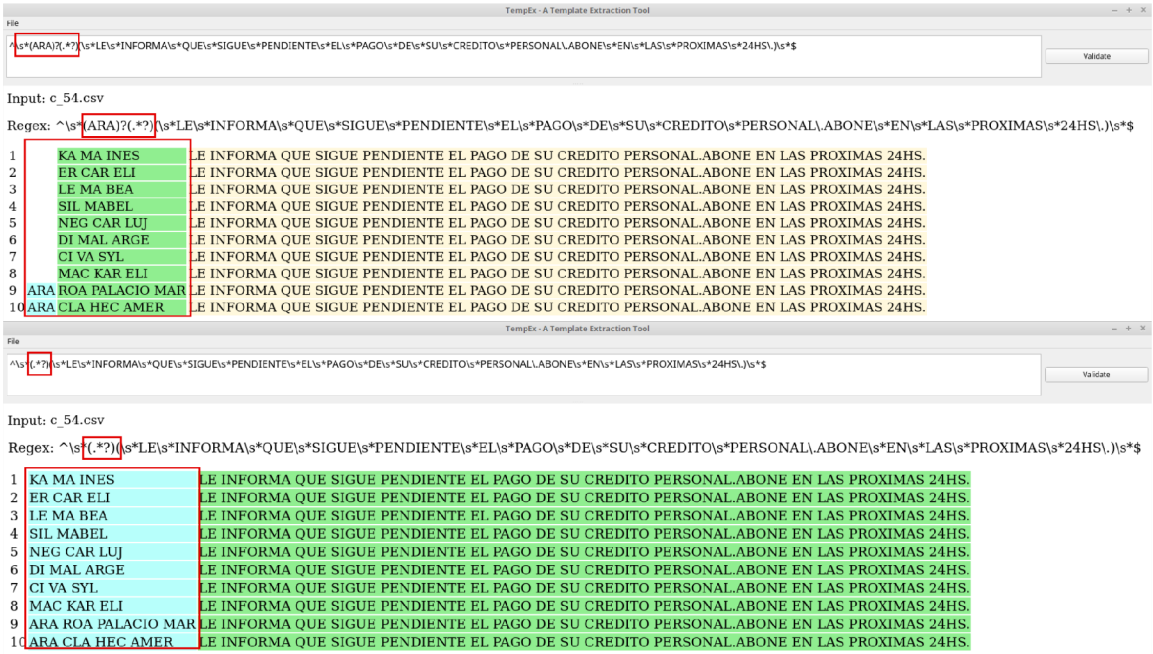

Figure 5.3: Simple GUI provided with the tool plus demonstration of the possibility to modify the regular expression and re-run validation.

As it was mentioned before, the UI is very simple. It has never been intended to be the main output method of the tool. It mostly serves as a support mechanism for the extraction/validation process.

Chapter 6

Method Evaluation and Experiments

The main focus of this chapter is to demonstrate how well does the proposed extraction algorithm perform on real-world data, what is the quality of its outputs, and what are the benefits and drawbacks. Before that, the datasets used during development and experimenting are described.

6.1 Used SMS Spam Datasets

For all phases of algorithm design and result evaluation, real-world examples of SMS spam campaigns were used. All together, there were 300 campaigns — 100 from Brazil, 100 from Indonesia and 100 from Argentina. They often contain sensitive data. Therefore, if there are examples presented, such information is modified to get randomized character.

Variety of campaigns that can be found in the datasets is really high. From campaigns with several hundreds of messages, to campaigns with almost 30 000 messages. Really simple ones with low variety among messages, as well as campaigns so complex that it would be very difficult for a human to discover a pattern in it. Some are purely ascii-based, some contain special Unicode characters. Most of them are written in a natural language, others are closer to formal languages.

What is common to all of them is that every message of every campaign is somehow different. There are multiple mechanisms to achieve this intra-campaign variability. Here is a list of the most common practices:

- *Variable data* — Each message contains a different value for a certain small part (typical for names).
- *Variable length* — Messages consist of different number of tokens.
- *Missing parts* — A subset of the previous case. Certain parts are completely missing in some messages.
- *Lower/upper-case* — Mixing upper-case and lower-case notation.
- *Different formats* — Messages use multiple formats to display the same type of data (typical for time, dates, etc.)

Most common or interesting practices, as well as overall evaluation of tool's performance, are shown in following sections. Out of those three country-based campaign sets, the Brazilian and the Argentinian provide the best results. The Indonesian set contains campaigns, which often lack structure, patterns are very weak, and they contain a lot of noise.

When an example is shown in this chapter, only a small portion of the whole campaign is shown. This way the results are compact enough for presentation. However, some information my be missing. For example, not every token, which appears in the RE, can be shown in the messages. Of course, we try not to omit any important detail, but it is not possible to show everything.

6.2 Template Extraction Performance

The algorithm is designed in a way that the most time-consuming algorithms are executed only when the current template does not cover the currently processed sample. The more general the template becomes, the less mismatches occur. The goal of this practice is to significantly reduce time necessary for the extraction. It is important to understand that the amount of samples needed to extract a template depends on complexity of a campaign.

Firstly, 10 large campaigns were selected (10 000 messages and more), without taking their complexity into consideration. Here are the observed results:

Campaign name	**Used samples**	**Campaign's size**
Argentina/c_0	13	28 760
Argentina/c_1	12	24 974
Argentina/c_8	5	11 706
Brazil/c_0	5	11 740
Brazil/c_1	5	11 121
Indonesia/c_0	15	16 522
Indonesia/c_1	13	12 625

Table 6.1: For 7 selected large campaigns, it shows how many samples were used for template extraction and how many samples are there in total.

Here average results for all campaigns [average number of samples used / average size of the campaign]:

- *Brazil* – 7.72/2726
- *Argentina* – 7.62/4364
- *Indonesia* – 13.02/2697

We can see that there are very few samples (compared to campaigns' sizes) used to extract a template. It significantly reduces time necessary for extraction. Especially finding an alignment is a very "expensive" procedure. Eliminating it to so few executions is a great benefit. It is also common that these characteristic samples are often spread somewhere close to the beginning of a campaign. Therefore, it is usually possible to use first, let us say, 100 samples to get a template matching up to 99% percent of the whole campaign.

6.3 Template Quality on Specific Examples

This section is dedicated to presenting several examples of real-world SMS spam campaigns with certain characteristic properties. The goal is to show how does the extraction tool handle these situation and to justify certain mechanisms built into the tool, such as RE optimization and data pre-sorting. Besides showing real-world data only, there are also several examples crafted precisely to target certain aspects of the tool.

6.3.1 Campaigns with Strong Patterns

In the beginning, we are going to present the type of campaigns for which a good result is provided without any problem. Those are campaigns in which large portion of tokens is static (does not change). They can be used as reference examples for further, more complex, campaigns.

A trivial example is shown in Figure 6.1. The pattern in this case is very strong. There are only two variable parts. One results in a wildcard group, because the amount of distinct values is too high. The other ends up as an integer wildcard group, because only numbers are present in there.

(.*?)	\, A LEAL assessoria Bradesco te convida p\/ regularizacao da sua pendencia\. Ligue\: 0800055555\/40222333 ou Whats 11 99999\-8888\. Cod ([+-]?\d+)	
CHEILA	, A LEAL assessoria Bradesco te convida p/ regularizacao da sua pendencia. Ligue: 0800055555/40222333 ou Whats 11 99999-8888. Cod	11226817
ALLAN	, A LEAL assessoria Bradesco te convida p/ regularizacao da sua pendencia. Ligue: 0800055555/40222333 ou Whats 11 99999-8888. Cod	20178297
GABRIEL	, A LEAL assessoria Bradesco te convida p/ regularizacao da sua pendencia. Ligue: 0800055555/40222333 ou Whats 11 99999-8888. Cod	20308489
ANTONIO	, A LEAL assessoria Bradesco te convida p/ regularizacao da sua pendencia. Ligue: 0800055555/40222333 ou Whats 11 99999-8888. Cod	20266208

Figure 6.1: An example of a trivial campaign, with a very strong pattern and only two variable parts.

A slightly more complicated example is shown in Figure 6.2. The pattern there is still strong, but more variable parts are detected. However, the RE describing this campaign still perfectly reflects the pattern of the campaign.

(.*?)	Seu (desconto|DESC0NT0)	foi atualizado\. Confira agora a nova proposta\. Ligue (08001111111|08009999999)	\. Valido ate 09\/10\.
CIRLENE	Seu DESC0NT0	foi atualizado. Confira agora a nova proposta. Ligue	08001111111	. Valido ate 09/10.
ANDRELINA	Seu DESC0NT0	foi atualizado. Confira agora a nova proposta. Ligue	08009999999	. Valido ate 09/10.
JEISYANE	Seu desconto	foi atualizado. Confira agora a nova proposta. Ligue	08001111111	. Valido ate 09/10.
JOSIVALDO	Seu desconto	foi atualizado. Confira agora a nova proposta. Ligue	08001111111	. Valido ate 09/10.
JOSEILDO	Seu DESC0NT0	foi atualizado. Confira agora a nova proposta. Ligue	08001111111	. Valido ate 09/10.
OZIANE	Seu desconto	foi atualizado. Confira agora a nova proposta. Ligue	08001111111	. Valido ate 09/10.
JANDIRA	Seu DESC0NT0	foi atualizado. Confira agora a nova proposta. Ligue	08001111111	. Valido ate 09/10.
JOSELMA	Seu DESC0NT0	foi atualizado. Confira agora a nova proposta. Ligue	08009999999	. Valido ate 09/10.

Figure 6.2: A campaign with a strong pattern, but multiple variable parts.

6.3.2 Pre-Sorting of the Campaign

The extraction algorithm allows processing campaign messages sorted by the number of tokens they consist of (in decreasing fashion). Through experimenting we discovered that in some cases pre-sorting of the input dataset improves results of the aligning process.

Once a token is assigned to a segment, it cannot be moved to another one, even though it would fit there better. It is not known where does a token fit the best, until all samples are processed. However, then it is too late to move them around. If a token appears first in the longest possible message (which contains this token), it is easier to align the same token when shorter messages are being processed.

	(.*?)	CONSULTE\/NEGOCIE SEU DEBITO
1	PREZADO(A),	CONSULTE/NEGOCIE SEU DEBITO
2	PREZADO(A),	CONSULTE/NEGOCIE SEU DEBITO
3	PREZADO(A),	CONSULTE/NEGOCIE SEU DEBITO
4	PREZADO(A),	CONSULTE/NEGOCIE SEU DEBITO
5	PREZADO(A),	CONSULTE/NEGOCIE SEU DEBITO
6	PREZADO(A),	CONSULTE/NEGOCIE SEU DEBITO
7	PREZADO(A),	CONSULTE/NEGOCIE SEU DEBITO
8	PREZADO(A),	CONSULTE/NEGOCIE SEU DEBITO
9	PREZADO(A),	CONSULTE/NEGOCIE SEU DEBITO
10	PEDRO,	CONSULTE/NEGOCIE SEU DEBITO
11	JOSE,	CONSULTE/NEGOCIE SEU DEBITO
12	PREZADO(A),	CONSULTE/NEGOCIE SEU DEBITO
13	MARCELO,	CONSULTE/NEGOCIE SEU DEBITO
14	LUIZA,	CONSULTE/NEGOCIE SEU DEBITO

(1)

	(.*?)	\,	CONSULTE\/NEGOCIE SEU DEBITO
1	PREZADO(A)	,	CONSULTE/NEGOCIE SEU DEBITO
2	PREZADO(A)	,	CONSULTE/NEGOCIE SEU DEBITO
3	PREZADO(A)	,	CONSULTE/NEGOCIE SEU DEBITO
4	PREZADO(A)	,	CONSULTE/NEGOCIE SEU DEBITO
5	PREZADO(A)	,	CONSULTE/NEGOCIE SEU DEBITO
6	PREZADO(A)	,	CONSULTE/NEGOCIE SEU DEBITO
7	PREZADO(A)	,	CONSULTE/NEGOCIE SEU DEBITO
8	PREZADO(A)	,	CONSULTE/NEGOCIE SEU DEBITO
9	PREZADO(A)	,	CONSULTE/NEGOCIE SEU DEBITO
10	PEDRO	,	CONSULTE/NEGOCIE SEU DEBITO
11	JOSE	,	CONSULTE/NEGOCIE SEU DEBITO
12	PREZADO(A)	,	CONSULTE/NEGOCIE SEU DEBITO
13	MARCELO	,	CONSULTE/NEGOCIE SEU DEBITO
14	LUIZA	,	CONSULTE/NEGOCIE SEU DEBITO

(2)

Figure 6.3: A difference between not pre-sorting the campaign (1) and doing so (2). The correct position is better identifiable, if longer (token-wise) messages are processed first.

An example of pre-sorting is shown in Figure 6.3. Part (1) shows that the comma character is not identified and gets covered by the wildcard. In (2), which had been pre-sorted, it becomes a static part. However, benefits of pre-sorting do not apply to every campaign. Therefore, it is left as an optional step. It is controlled by an argument of the program. As presented in Section 4.4.9, the main drawback of pre-sorting is the necessity of tokenizing every message of a campaign. Even though it would not be used to generalize the template.

6.3.3 An Outlier Ruining It All

Following experiment shows that it is not always desirable to use the whole campaign for template extraction, but rather its subset. Figure 6.4 shows a problem of one message significantly decreasing readability of the resulting RE. The message is similar enough to the common case, so it is not removed as an outlier. Its certain part is significantly different, though. In this case, the message might be deformed by an error during I/O operation, however its influence on the result is visible. This specific campaign contains 5 832 messages. Less than twenty of them is problematic in the described way. The generated RE matches 5 813 of messages (the others are correctly removed as outliers).

Regex: ^OLA(.*?) tudo bem(\;|\?)(sn|sou|seu)(a|Velit)(Gerente|asue)(Marta|Aline)e(.*?)(\)|\Ã|\@|\,)(.*?)(chama|liga)(no)?(Whatsapp)?([+-]?\d+)(\.)?\s*$

	OLA	(.*?)	tudo bem (\;|\?)	(sn|sou|seu)	(a|Velit)	(Gerente|asue)	(Marta|Aline)	e	(.*?)	(\)|\Ã|\@|\,)	(.*?)
1	OLA	J	tudo bem	?	sou	a	Gerente	Aline	e	preciso falar com voce sobre o seu DEBITO Itau	,	atualizamos seu DESC0NT0 me
8	OLA	A	tudo bem	?	sou	a	Gerente	Aline	e	preciso falar com voce sobre o seu DEBITO Itau	,	atualizamos seu DESC0NT0 me
9	OLA	S	tudo bem	?	sou	a	Gerente	Aline	e	preciso falar com voce sobre o seu debito Itau	,	atualizamos seu desconto me
10	OLA	H	tudo bem	?	sou	a	Gerente	Aline	e	preciso falar com voce sobre o seu DEBITO Itau	,	atualizamos seu desconto me
11	OLA	S	tudo bem	?	sou	a	Gerente	Aline	e	preciso falar com voce sobre o seu DEBITO Itau	,	atualizamos seu desconto me
12	OLA	A	tudo bem	?	sou	a	Gerente	Aline	e	preciso falar com voce sobre o seu debito Itau	,	atualizamos seu desconto me
29	OLA	R	tudo bem	?	sou	a	Gerente	Marta	e	preciso falar com voce sobre o seu debito do Itau	,	atualizamos seu desconto me
163	OLA	L	tudo bem	?	sou	a	Gerente	Aline	e	preciso falar com wD9g	Ã	□V,Ã□(P o seu DEBITO Itau,atualizamos seu desconto me
178	OLA	T	tudo bem	?	sou	a	Gerente	Aline	e	preciso falar com voce sobre o seu debito Itau	,	atualizamos seu desconto me
272	OLA	J	tudo bem	?	sou	a	Gerente	Marta	e	preciso falar com voce sobre o seu DEBITO do cartao Cetelem	,	atualizamos seu DESC0NT0 me
273	OLA	A	tudo bem	?	sou	a	Gerente	Aline	e	preciso falar com voce sobre o seu debito Itau	,	atualizamos seu desconto me

Figure 6.4: An example of a campaign in which a very small portion of a message (which is not detected as an outlier) influence the result in a very significant way.

On the other hand, Figure 6.5 shows the very same campaign, but only 100 samples were used to extract the template (and the number could go even lower). The result matches 5 743 messages of the campaign and it successfully gets rid of those problematic samples. Another, but less significant, advantage is a slightly reduced execution time.

	OLA	(.*?)	tudo bem\? sou a Gerente	(Aline|Marta)	e preciso falar com voce sobre o seu	(debito|DEBITO)	(do)?	Itau\,atualizamos seu	(DESC0NT0|desconto)	me
1	OLA	J	tudo bem? sou a Gerente	Aline	e preciso falar com voce sobre o seu	DEBITO		Itau,atualizamos seu	DESC0NT0	me
2	OLA	J	tudo bem? sou a Gerente	Aline	e preciso falar com voce sobre o seu	DEBITO		Itau,atualizamos seu	DESC0NT0	me
3	OLA	W	tudo bem? sou a Gerente	Aline	e preciso falar com voce sobre o seu	DEBITO		Itau,atualizamos seu	DESC0NT0	me
4	OLA	J	tudo bem? sou a Gerente	Aline	e preciso falar com voce sobre o seu	DEBITO		Itau,atualizamos seu	DESC0NT0	me
5	OLA	N	tudo bem? sou a Gerente	Aline	e preciso falar com voce sobre o seu	debito		Itau,atualizamos seu	desconto	me
6	OLA	M	tudo bem? sou a Gerente	Aline	e preciso falar com voce sobre o seu	DEBITO		Itau,atualizamos seu	DESC0NT0	me
7	OLA	F	tudo bem? sou a Gerente	Aline	e preciso falar com voce sobre o seu	debito		Itau,atualizamos seu	desconto	me
8	OLA	A	tudo bem? sou a Gerente	Aline	e preciso falar com voce sobre o seu	DEBITO		Itau,atualizamos seu	DESC0NT0	me
9	OLA	S	tudo bem? sou a Gerente	Aline	e preciso falar com voce sobre o seu	debito		Itau,atualizamos seu	desconto	me
10	OLA	H	tudo bem? sou a Gerente	Aline	e preciso falar com voce sobre o seu	DEBITO		Itau,atualizamos seu	desconto	me
11	OLA	S	tudo bem? sou a Gerente	Aline	e preciso falar com voce sobre o seu	DEBITO		Itau,atualizamos seu	desconto	me
12	OLA	A	tudo bem? sou a Gerente	Aline	e preciso falar com voce sobre o seu	debito		Itau,atualizamos seu	desconto	me
13	OLA	A	tudo bem? sou a Gerente	Aline	e preciso falar com voce sobre o seu	debito		Itau,atualizamos seu	desconto	me
14	OLA	R	tudo bem?sou a Gerente	Marta	e preciso falar com voce sobre o seu	debito	do	Itau,atualizamos seu	desconto	me

Figure 6.5: The same campaign, but only 100 samples were used for extraction. The template did not degrade, because problematic samples got filtered out.

6.3.4 Token Sequences Disconnected

It was already briefly mentioned in the summary of extraction algorithms' properties (Section 4.4.9). A token is a unit of alignment and a segment is an aggregation of tokens. During merging of an alignment, each token is put into a segment. By this step, information about sequences of tokens is lost. The result is shown in Figure 6.6 — part (1) shows the result with secondary RE optimization turned off, in part (2) it is turned on.

	(Sr)?	(\.)?	(\()?	(a)?	(\))?	(.*?)	\,	(Ainda|ainda)	nao recebemos
1	Sr	.	(	a	)	M	,	ainda	nao recebemos
2						W	,	Ainda	nao recebemos
3						O	,	Ainda	nao recebemos
4	Sr	.	(	a	)	E	,	Ainda	nao recebemos
5	Sr	.	(	a	)	R	,	ainda	nao recebemos
6	Sr	.	(	a	)	H	,	ainda	nao recebemos
7						D	,	Ainda	nao recebemos
8						M	,	Ainda	nao recebemos
9	Sr	.	(	a	)	M	,	ainda	nao recebemos
10	Sr	.	(	a	)	L	,	Ainda	nao recebemos
11	Sr		(	a	)	W	,	Ainda	nao recebemos
12	Sr		(	a	)	F	,	Ainda	nao recebemos

(1)

	(.*?)	\,	(Ainda|ainda)	nao recebemos
1	Sr.(a) M	,	ainda	nao recebemos
2	W	,	Ainda	nao recebemos
3	O	,	Ainda	nao recebemos
4	Sr.(a)E	,	Ainda	nao recebemos
5	Sr.(a) R	,	ainda	nao recebemos
6	Sr.(a) H	,	ainda	nao recebemos
7	D	,	Ainda	nao recebemos
8	M	,	Ainda	nao recebemos
9	Sr.(a) M	,	ainda	nao recebemos
10	Sr.(a)L	,	Ainda	nao recebemos
11	Sr(a) W	,	Ainda	nao recebemos
12	Sr(a) F	,	Ainda	nao recebemos

(2)

Figure 6.6: Token sequence disconnection caused by loss of information during alignment (1) with additional RE optimization turned of. Same campaign with additional RE optimization shown in (2). The optional segments get covered by the wildcard.

The described problem is mostly visible in part (1), but it affects both. The correct solution is, however, subjective. If all possibilities (of yellow sequence) were listed, it would easily exceed the given limit. Therefore, it would degrade into a wildcard group. It would end up as shown in part (2). The part "`(Sr)?(\.)?`" is not so harmful, considering that the '`.`' does not appear in every instance. However, the sequence "`(\)?(a)?(\)?`" always

occurs together. RE "`(\(a\))?`" would be more suitable. What is the correct solution here depends on users' requirements. The algorithm catches and visualizes the pattern nicely. Users can always modify the resulting RE manually to fit their needs.

However, such fine-grained approach allows detecting small deviations from the (current) pattern. It is shown in part (1) of Figure 6.6, where the missing dot character is detected, even though such pattern has not been observed yet. Sadly, the meta RE characters (parenthesis, question marks) make the result more difficult to read.

6.3.5 Typing Conventions

A very common trick to make spam detection more difficult is using different typing conventions. Most common cases are shown in Figure 6.7. Part (1) displays different conventions used for representing date and time.

Another very common case affects phone numbers (or long sequences of numbers in general). As shown in part (2), some of them are typed as whole, others are split into several parts. This is extremely difficult to catch correctly during tokenization. Therefore, a more suitable analysis is required after. However, in terms of visual presentation the tool makes the pattern nicely visible for users.

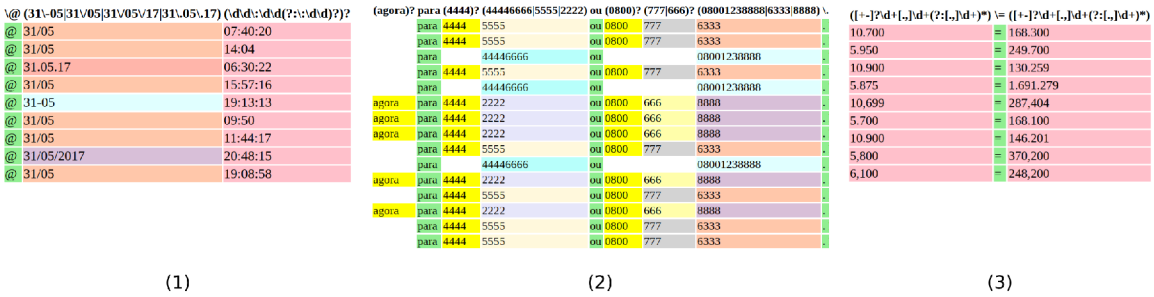

Figure 6.7: Examples of the most common cases where spammers use different notations — (1) date and time, (2) phone numbers, (3) numbers.

Last, but not least, case shown in part (3) of Figure 6.7 affects numbers in general. Multiple conventions can be used to write down a number. Mostly thanks to different decimal and thousands separators.

6.3.6 Campaign with Many Highly-Variable Parts

Many campaigns that we have analyzed follow this pattern — the structure is more or less fixed, but they contain multiple parts where highly variable data is placed. An example is shown in Figure 6.8. The ratio of fixed and variable parts is similar. The example contains five fixed parts, as well as five variable parts. We can see that variable parts can differ both lexicographically and in the number of tokens they consist of. However, we can conclude the algorithm succeeded in this case. It discovered the structure of the campaign and provided a readable RE on the output.

6.3.7 Campaign with Many Frequent Sub-Patterns

This example shows a campaign with many sub-patterns (Figure 6.9). Many of the constructions discussed it this section can be seen here — a highly variable part with names, different token types, optional groups, value-selection groups, rarely occurring values, . . . All

	(.*?)	\,	(.*?)	tenes \$	([+-]?\d+)	para c omprar en	(.*?)	presenta este cod de CY\!	([+-]?\d+)	FELIZ DIA MAMA\!
1	CA	,	LA CR	tenes $	13900	para c omprar en	BU HO	presenta este cod de CY!	11111	FELIZ DIA MAMA!
2	JA	,	GR NO	tenes $	21300	para c omprar en	LU HO	presenta este cod de CY!	22222	FELIZ DIA MAMA!
3	RO	,	RO EN	tenes $	4900	para c omprar en	CA AL	presenta este cod de CY!	33333	FELIZ DIA MAMA!
4	MO	,	RA ES	tenes $	10500	para c omprar en	LOS 5 HE	presenta este cod de CY!	44444	FELIZ DIA MAMA!
5	MA	,	AR MA	tenes $	7500	para c omprar en	CRI MA	presenta este cod de CY!	55555	FELIZ DIA MAMA!
6	OL	,	SE ED	tenes $	20400	para c omprar en	GA AU	presenta este cod de CY!	66666	FELIZ DIA MAMA!
7	SO	,	EL FI	tenes $	6700	para c omprar en	LA TA HOGAR	presenta este cod de CY!	77777	FELIZ DIA MAMA!

Figure 6.8: An example of a campaign where the ratio of fixed and highly variable parts is similar.

of them make extracting a pattern difficult for a human. Visualization can be, therefore, very useful.

It is difficult to automatically detect and remove outliers. In this case, certain parts of messages can differ significantly from each other. Having a visual representation of the template can be very useful in case a user decides to modify the RE manually.

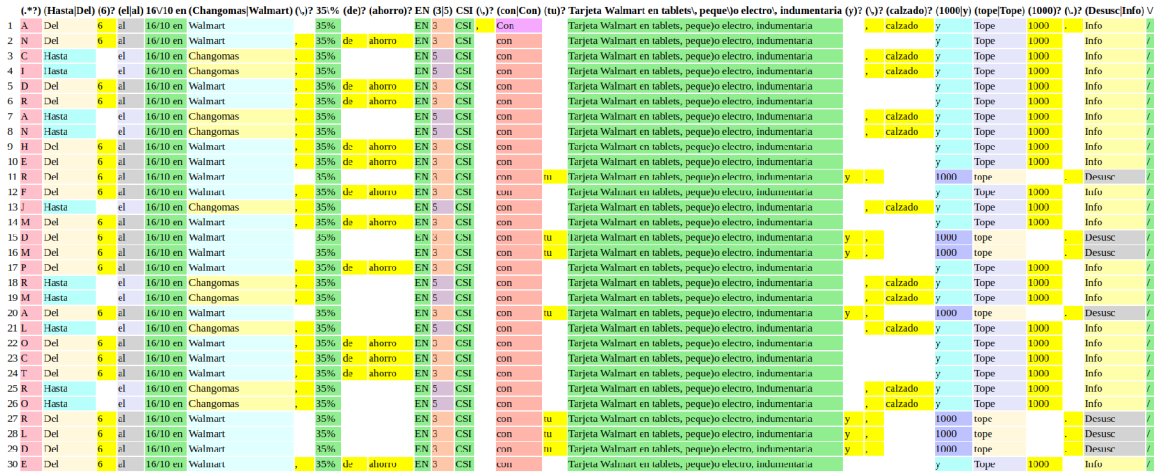

Figure 6.9: A campaign with a significant fixed part, but several frequent sub-patterns.

6.3.8 Template Degeneration

As much as the algorithm tries to extract a template, there is certain weakness in it. The threshold between transforming a segment into a choice group or a wildcard is a fixed number, which can be exceeded.

In figure 6.10 is shown an example which emphasizes this issue. Each segment simply gets more value than the threshold allows. This makes every segment to be transformed into a wildcard. The result after optimization is "`(.*?)`". Of course, the threshold can be raised, but it will still be a finite value. This significant drawback represents a trade-off (or a conflict) between readability and accuracy.

Even though this is an artificially constructed example, it demonstrates another spammers' approach to make SMS spam campaigns more diverse — *intentional typos*. Figure 6.11 shows a real-world example. Especially in SMS spam, typos can be quite common, because people do not care that much about them. This way a spammer can construct a more "trustworthy" message for advertisement, phishing, etc.

	(.*?)
1	a b c d e f
2	aa bb cc dd ee ff
3	aaa bbb ccc ddd eee fff
4	aaaa bbbb cccc dddd eeee ffff
5	aaaaa bbbbb ccccc ddddd eeeee fffff
6	aaaaaa bbbbbb cccccc dddddd eeeeee ffffff

Figure 6.10: A constructed example demonstrating problem with campaign degrading.

(.*?)	(para)?
WhatsApp Messenger	para
WhatsAppc Messenger	para
WhatsApp Messenger	para
WhatsApp heMessenger	para
WhatsApp wMessenger	para
WhatsApp Messengerp.	para
WhatsApp	
WhatsApp6 Messenger	para
WhatsApppp Messenger	

Figure 6.11: An example of a real campaign with intentional typos to make messages more diverse.

6.3.9 Patterns of Rotated Messages

Another artificial experiment constructed to test algorithm's capabilities in terms of pattern detection. Messages of this campaign contain the same sequence of tokens. However, in each message the start of the sequence is rotated by one step. The result is shown in Figure 6.12. We can see that the algorithm successfully detected the static sequence in each message and aligned corresponding parts.

	(l)?	(m)?	(n)?	(o)?	(p)?	a	b	c	d	e	f	g	h	i	j	k	(l)?	(m)?	(n)?	(o)?	(p)?
1						a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
2					p	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	
3				o	p	a	b	c	d	e	f	g	h	i	j	k	l	m	n		
4			n	o	p	a	b	c	d	e	f	g	h	i	j	k	l	m			
5		m	n	o	p	a	b	c	d	e	f	g	h	i	j	k	l				
6	l	m	n	o	p	a	b	c	d	e	f	g	h	i	j	k					

Figure 6.12: Campaign in which each message is a different rotation of the same sequence. The extraction algorithm correctly detects the shifted pattern.

6.3.10 Order of Messages

As discussed in Section 4.4.9, the incremental generalization of a template brings a significant drawback. The first message read decides what a genuine message is. If the first message happens to be an outlier, all the actually genuine messages will be discarded as outliers. An example is shown in Figure 6.13. In this case, all of the genuine messages are

skipped. Figure 6.14 shows different order of messages, where major pattern is properly reflected. A modification resolving this problem is proposed in Section 6.4. However, as results in Section 6.3.14 indicate, this issue occurred only in 2 (out of 300) cases.

	Bpk\/Ibu Yth\,Toyota (BX|BX4) akan jatuh tempo utk Service Berkala (30000km|10000km) \. Nikmati CC (C|CC) J di Auto2000\.S\&K* berlaku\. Utk Booking hub 061\-8888000						
1	Bpk/Ibu Yth,Toyota	BX	akan jatuh tempo utk Service Berkala	10000km	. Nikmati CC	CC	J di Auto2000.S&K* berlaku. Utk Booking hub 061-8888000
2	Bpk/Ibu Yth,trima kasih servis Toyota B1 tgl 14.05.2016 di Auto2000 G S. Jika ada keluhan, hub S P - Serv.Advisor di 061-8888000						
3	Bpk/Ibu Yth,trima kasih servis Toyota B2 tgl 14.05.2016 di Auto2000 B B. Jika ada keluhan, hub J P - Serv.Advisor di 061-8888000						
4	Bpk/Ibu Yth,trima kasih servis Toyota B3 tgl 14.05.2016 di Auto2000 R B. Jika ada keluhan, hub R A - Serv.Advisor di 061-8888000						
5	Bpk/Ibu Yth,Toyota	BX4	akan jatuh tempo utk Service Berkala	30000km	. Nikmati CC	C	J di Auto2000.S&K* berlaku. Utk Booking hub 061-8888000
6	Bpk/Ibu Yth,trima kasih servis Toyota B4 tgl 14.05.2016 di Auto2000 S. Jika ada keluhan, hub M S - Serv.Advisor di 061-8888000						
7	Bpk/Ibu Yth,trima kasih servis Toyota B1 tgl 14.05.2016 di Auto2000 G S. Jika ada keluhan, hub S P - Serv.Advisor di 061-8888000						
8	Bpk/Ibu Yth,trima kasih servis Toyota B2 tgl 14.05.2016 di Auto2000 B B. Jika ada keluhan, hub J P - Serv.Advisor di 061-8888000						

Figure 6.13: A demonstration of the first message defining what a genuine message is.

	Bpk\/Ibu Yth\,trima kasih servis Toyota (B4|B2|B3|B1) tgl 14\.05\.2016 di Auto2000 (G|R|B)? (B|S) \. Jika ada keluhan\, hub (M)? (J|R|S) (P|A)? \- Serv\.Advisor di 061\-8888000									
1	Bpk/Ibu Yth,trima kasih servis Toyota	B1	tgl 14.05.2016 di Auto2000	G	S	. Jika ada keluhan, hub		S	P	- Serv.Advisor di 061-8888000
2	Bpk/Ibu Yth,trima kasih servis Toyota	B2	tgl 14.05.2016 di Auto2000	B	B	. Jika ada keluhan, hub		J	P	- Serv.Advisor di 061-8888000
3	Bpk/Ibu Yth,trima kasih servis Toyota	B3	tgl 14.05.2016 di Auto2000	R	B	. Jika ada keluhan, hub		R	A	- Serv.Advisor di 061-8888000
4	Bpk/Ibu Yth,Toyota BX akan jatuh tempo utk Service Berkala 10000km. Nikmati CC CC J di Auto2000.S&K* berlaku. Utk Booking hub 061-8888000									
5	Bpk/Ibu Yth,Toyota BX4 akan jatuh tempo utk Service Berkala 30000km. Nikmati CC C J di Auto2000.S&K* berlaku. Utk Booking hub 061-8888000									
6	Bpk/Ibu Yth,trima kasih servis Toyota	B4	tgl 14.05.2016 di Auto2000		S	. Jika ada keluhan, hub	M	S		- Serv.Advisor di 061-8888000
7	Bpk/Ibu Yth,trima kasih servis Toyota	B1	tgl 14.05.2016 di Auto2000	G	S	. Jika ada keluhan, hub		S	P	- Serv.Advisor di 061-8888000
8	Bpk/Ibu Yth,trima kasih servis Toyota	B2	tgl 14.05.2016 di Auto2000	B	B	. Jika ada keluhan, hub		J	P	- Serv.Advisor di 061-8888000

Figure 6.14: A demonstration of the first message defining what a genuine message is (correct order reflecting what a majority looks like).

6.3.11 Campaigns with Weak Patterns

Unfortunately, there are also campaigns which cause significant troubles to the algorithm. They usually have very weak patterns or the pattern is masked by large amount of noise. Especially the noise brings too many distinct values into each segment and they eventually degenerate into a wildcard. Then, multiple wildcards in a row get optimized into a single one. The result is usually similar to what is shown in Figure 6.15. The final RE is not very campaign-specific, as it mostly consists of wildcards. Therefore, it does not describe patterns within the campaign.

	(.*?)	(\=|tgl|\:|\.)	(.*?)	(\=|\/|\:)	(.*?)	(\-|Awal)
1	S875-131-196, P'belian SS10	.	081267606679 Hrg	=	10700 SUKSES SN: 7053115453261752460 S (	Awal
2	S3091, P'belian SS20	.	082162734034 Hrg	=	20300 SUKSES SN: 7053116093961358020 S (	Awal
3	TRX AMAN OM,,#1091674 TSEL S10	.	085370628914 SUKSES. SN	/	Ref: 7053122562561305190. Saldo 300.250	-
4	sonasoni	:	A50.085216778887 Hrg	=	49600 SUKSES SN: 41001436297290 TRX: S50.(	Awal
6	PLN20	.	520051587836 Hrg	=	20550 SUKSES SN: 1013	-
7	paketdata 50&100 ok/XL 10	.	000 XL10.081993330890 SUKSES. SN	/	Ref: 17050411455001. Saldo 19,175	-
8	SATRIA TRONIK I10	.	085747854416 SUKSES. SN	/	Ref: 00881900001564219469. Saldo 256.294	-
9	TR10	.	089505202703 Hrg	=	10400 SUKSES SN: V10. Saldo Rp.(	Awal
10	S1055-8-274, P'belian SS10	.	085271060367 Hrg	=	10700 SUKSES SN: 7053116441261318090 S (	Awal
11	S10	.	081218041270 Hrg	=	10575 SUKSES S/N: 70531081904212175 80.SALDO Rp.(	Awal
12	S2453, P'belian SS5	.	082273475713 Hrg	=	5600 SUKSES SN: 7053116080261317690 S (	Awal
13	S5	.	085240347058 Hrg	=	5750 SUKSES VSN: 7053115230701525731. Saldo Rp.(	Awal
14	TG5	.	082319010345 Hrg	=	5600 SUKSES S/N: 7053109430921707090. Saldo Rp.(	Awal
16	\nS20	.	2.081248683269 Hrg	=	20500 SUKSES S/N: 7053116254321604310. Saldo Rp.(	Awal
17	MD, Trx X5	.	087850198883 Hrg	=	5500 SUKSES S/N: 17050411044216. Saldo (	Awal
18	S47-178, P'belian SS5	.	081275106087 Hrg	=	5750 SUKSES SN: 7053108020361482300 S (	Awal
19	S468, P'belian SS10	.	082165063006 Hrg	=	10600 SUKSES SN: 7053108232061065850 S (	Awal
20	Yth FEBRI ISAT I10 085851164132 SUKSES	.	SN	/	Ref: 00881300001563445553. Saldo 32.125	-

Figure 6.15: A very noisy campaign with a weak pattern. The extraction algorithm does not provide a sufficient RE on the output.

One of the possible solutions in such case is, again, to reduce the number of sample used for extraction. Noisy samples are usually dispersed over the campaign. This way they

can be filtered away or, at least, reduced. Figure 6.16 shows the same campaign, but the number of input samples was reduced to 100. Such RE matches around 72% of the whole campaign, while its readability is much better. Luckily, such campaigns are quite rare. In datasets used for development and testing, there are only few such examples among campaigns from Indonesia.

	(.*?)	\.	([+-]?\d+)	(\.)?	Hrg\=	([+-]?\d+)	SUKSES	(SN|S)?	(/)?	(SN|REF|VSN|TID|N)	\:	(.*?)	\(Awal
1	S875-131-196, P'belian SS10	.	081267606679		Hrg=	10700	SUKSES	S		N	:	7053115453261752460 S	(Awal
2	S3091, P'belian SS20	.	082162734034		Hrg=	20300	SUKSES	S		N	:	705311609396135 8020 S	(Awal
4	sonasoni : A50	.	085216778887		IIrg=	49600	SUKSES	S		N	:	41001436297290 TRX: S50.	(Awal
6	PLN20	.	520051587836		Hrg=	20550	SUKSES	S		N	:	1013-4932-1742-7977-2835/RUBINAH/B1/1300/19,0 Sal.	(Awal
9	TR10	.	089505202703		Hrg=	10400	SUKSES	S		N	:	V10. Saldo Rp.	(Awal
10	S1055-8-274, P'belian SS10	.	085271060367		Hrg=	10700	SUKSES	S		N	:	7053116441261318090 S	(Awal
11	S10	.	081218041270		Hrg=	10575	SUKSES	S	/	N	:	7053108190421217580.SALDO Rp.	(Awal
12	S2453, P'belian SS5	.	082273475713		Hrg=	5600	SUKSES	S		N	:	705311608026131 7690 S	(Awal
13	S5	.	085240347058		Hrg=	5750	SUKSES			VSN	:	7053115230701525731. Saldo Rp.	(Awal
14	TG5	.	082319010345		Hrg=	5600	SUKSES	S	/	N	:	7053109430921707090. Saldo Rp.	(Awal
16	\nS20.2	.	081248683269		Hrg=	20500	SUKSES	S	/	N	:	7053116254321604310. Saldo Rp.	(Awal
17	MD, Trx X5	.	087850198883		Hrg=	5500	SUKSES	S	/	N	:	17050411044216. Saldo	(Awal
18	S47-178, P'belian SS5	.	081275106087		Hrg=	5750	SUKSES	S		N	:	7053108020361482300 S	(Awal
19	S468, P'belian SS10	.	082165063006		Hrg=	10600	SUKSES	S		N	:	7053108232061065850 S	(Awal

Figure 6.16: A result of template extraction from only a few samples of a campaign with a very weak pattern.

6.3.12 Multi-line Messages

The algorithm is designed to work in line-oriented fashion, expecting each line to be a standalone message of a campaign. Therefore, a new line character is considered a separator of messages. However, data sets, which have been used to evaluate performance of the algorithm, contain several campaigns where messages are spread across multiple lines. An example is shown in Figure 6.17.

	(\\nEND|\\nREV|BEGIN)	\:	(V|20170531T222248Z|VCARD|20170531T142944Z|VCA)?
1	BEGIN	:	VCARD
2	\nVERSION:2.1		
3	\nN;ENCODING=QUOTED-PRINTABLE;CHARSET=UTF-8:;=		
4	\nFadly.;;;		
5	\nTEL;VOICE;CELL:082259699643		
6	\nEND	:	VCARD
7	\n		
8	BEGIN	:	VCARD
9	\nVERSION:2.1		
10	\nN;ENCODING=QUOTED-PRINTABLE;CHARSET=UTF-8:;=		
11	\nadek=20/amran;;;		
12	\nTEL;VOICE;CELL:+6285831337313		
13	\nEND	:	VCARD
14	\n		

Figure 6.17: An example of a campaign in which messages are spread across multiple lines. The algorithm cannot reconstruct the message from several lines and fails to provide a satisfying result.

We can see that it is the first line that is taken as a genuine message of the campaign. It cannot recognize other lines as parts of the same message and fails to provide a good result. This make the algorithm unfit for different types of communications, such as networking protocols, where such pattern is common. However, an extension of the algorithm is proposed in Section 6.4.

6.3.13 Long Sequences

The last experiment is focused on observing how does the algorithm perform on long sequences. It is important to, again, realize the length limitation of SMS messages. Long sequences for this experiment were constructed from existing campaigns. Messages of multiple campaigns were concatenated together to create longer sequences. Time necessary for extracting a template from 1 to 15 merged campaigns were observed. Campaigns in which length of messages is around 160 characters (the limit for SMS messages) were picked. Results are shown in Figure 6.18. There were 100 messages used for template extraction and the required times are an average from five execution on a single machine.

There were 4 long-message campaigns constructed. One was constructed by concatenating a trivial campaign repeatedly (blue curve). Its messages contained only one variable part, the rest was static. The second experiment (red curve) is also a concatenation of the same, still relatively simple, campaign with 3 variable parts. The third one (yellow curve) is constructed in the same way, but the pattern of its structure is more complex — up to seven highly variable parts. The last one (green curve) is a concatenation of randomly selected campaigns, which provided good results during experimenting.

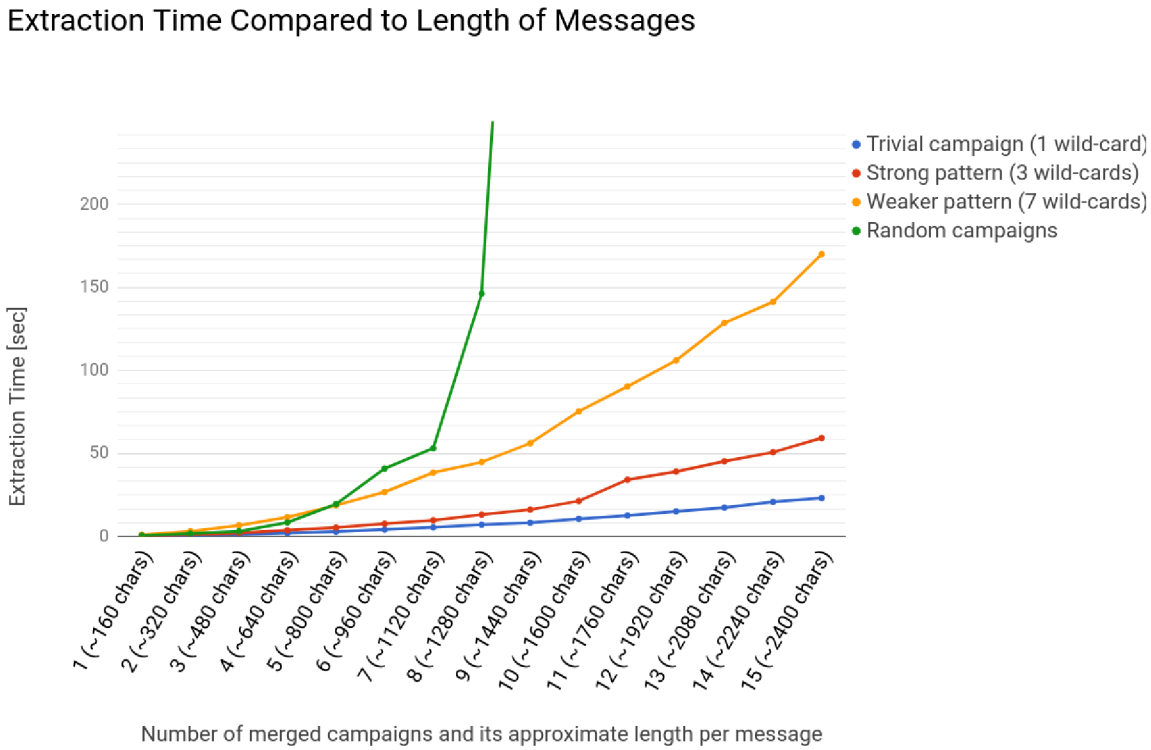

Figure 6.18: Chart shows relation between length of messages in a campaign and time necessary for extraction. Blue, red and orange curves shows results when a same campaign is repeated to construct longer messages. The green line shows concatenation of random messages.

In the cases where the same campaign was just repeated, to construct a longer message, the time necessary for template extraction reflects complexity of the campaign. From the basic constructed campaign, where messages are up to 2400 characters long, a template is extracted under 30 seconds. In a slightly more complex case it is still under one minute

for 2400 characters per message. In the most complex repetitive campaign it takes slightly over two minutes.

However, when random campaigns are concatenated to create a single long one, the results get bad around 7 merged instances. We did not observe any extra high complexity in campaigns 8 and 9, to influence the result so much. Nevertheless, it is a good demonstration that for complex campaigns, there is a limit around 1000 characters, before the extraction starts taking too long. After that it depends on complexity of the campaign.

6.3.14 Overall Result Quality

Here we provide a chart summarizing quality of extracted REs from each campaign of the dataset. Evaluation of a RE's quality is subjective, because it is impossible to make it standardized for all possible campaigns. Especially when readability is its part. In total, 300 hundred campaign were evaluated. The chart with results is shown in Figure 6.19. Following trends were observed:

- *Ok* — Template reflects the pattern in a clear understandable way. It does not degenerate.
- *Minor Outlier* — A certain part of a single message is very different from the general pattern, while its majority is the same. If this sample is not identified as an outlier, it decreases readability of the result. This situation can be avoided by using less samples for template extraction.
- *Sequence Disconnection* — A phenomenon described in Section 6.3.4.
- *Partially degrades* — A small part of the RE degrades to a single wildcard RE, while the rest reflects the pattern appropriately.
- *Significantly degrades* — Majority of the RE degrades into a wildcard, due to high variety of messages with weak pattern. This issue can be usually resolved by using less samples to extract the template, while maintaining high match ratio.
- *Multi-line* — A campaign contains messages spread across multiple lines.
- *Order of Messages* – If an outlier is the first message, genuine messages are then rejected due to a design flaw.

We can see that the extraction is most efficient on campaigns from Brazil. On the other hand, campaigns from Indonesia are the most problematic. It reflects the reality that they have the highest ratio of campaigns with weak patterns and a lot of noise. However, in all three cases, satisfying output makes majority of results.

Categories *Minor Outlier, Sequence Disconnection, Degrades Partially/Significantly* provide slightly worsen results. However, they can be usually significantly improved by using less samples for extraction, or simple modification by the user.

6.4 Future Work Based on Experiments

Doing experiments with many real-world campaigns showed how does the algorithm (and the whole tool) perform. Thanks to the highly variable character of the data, we were able to discover several drawbacks which were not so obvious at the beginning. Majority of

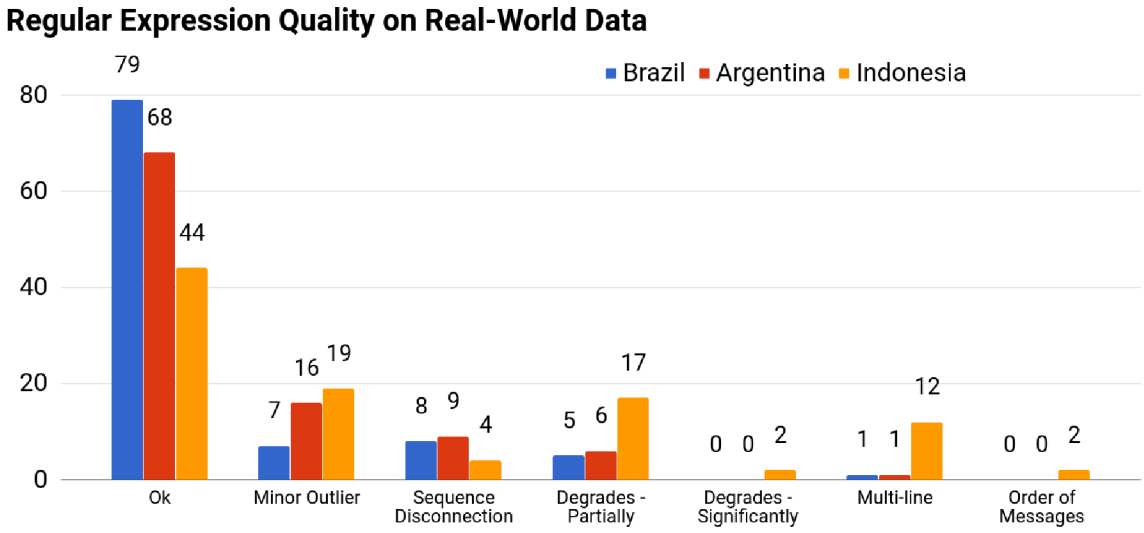

Figure 6.19: Evaluation of regular expressions extracted from real-world data.

these observations were already implemented and resolved. Some of them are left for future extensions. Those are discussed in this section.

Probably one of the main drawback is the order-dependent extraction process. It is one of the very few cases in which the extraction can fail completely. However, the incremental generalization of a template brings one of the biggest advantages — good performance. Therefore, there has to be a good trade-off found. One of the solution can be extraction multiple templates from a single campaign. If a message does not match the current template, a new template is created. Based on alignment with each template, messages would be assigned to an existing template, or they would create a new one. This process would be a burden for performance, but as experiments have shown, these cases are rather rare. The good new is that this issue occurred only twice, out of 300 campaigns.

Second extension, which could improve provided results, is implementing a mechanism for re-merging tokens back into original sequences. Currently, the information about exact token sequences gets lost during alignment merging. This is sometimes beneficial, because it allows detecting slight changes in otherwise fixed patterns. On the other hand in some cases it decreases readability of the resulting RE.

Another extension, based on over-all evaluation of experiments, is adding support for multi-line messages. User could manually define what the separator is — a new line, a sequence of characters, ... This way more types of data could be processed, not just short messages on a single line.

Performing intra-segment analysis for constructing more accurate RE for each segment, to make a RE more specific to each campaign. However, benefits of this feature would have to be discussed in comparison with the extra computation necessary.

Considering the interaction part of the tool, a more advanced graphical interface with more options could be introduced. This way users would not be forced to use the tool mainly in the command line. Lastly, providing users with a list of all concrete values matched in each RE group, could be very useful for further analysis of a campaign.

Chapter 7

Conclusion

SMS is a basic service present on almost every mobile device. Hundreds of thousands of messages are sent every minute. Even though the usage of SMS service is decreasing in some parts of the world, they are still very popular and widely used in others. Therefore, SMS platform represents huge potential for variety of fraudulent activities, such as spamming. Unfortunately, not enough attention has been paid to SMS spam and methods to fight it, as it has been with e-mail spam.

First, we a made brief summary of spam, with focus on SMS communication. We presented its properties and basic methods of fighting it. We further specified the relationship between a campaign and a template and how a template can be used to identify its source campaign.

Even though the purpose of this work is to help fighting spam in SMS, the task can be generalized — learning a formal description of a language from a finite set of positive examples. Therefore, we showed several methods (such as automata induction, genetic programming, etc.) to accomplish this task, their advantages and possible drawbacks.

We designed an algorithm capable of reversely extracting a template from given messages of a campaign. It is based on aligning two sequences based on similarity of their certain parts. This idea was adopted (and extended) from bioinformatics, where this approach is used to find regions of similarity in protein sequences. It performs incremental generalization of an initial template. The algorithm is design in a way that time-consuming algorithms are executed only when a message does not match the current template. This way even very large campaigns can be processed fast.

At the end of execution, a template in form of a regular expression is provided. Making a conclusion on whether the resulting template is correct, or not, is very subjective, as readability is one of the criteria. Performance of the algorithm was validated against more than three hundred real campaigns from all around the world.

In the future, a few drawbacks of the algorithm design should be eliminated. For example, as of now the result of extraction is dependent on the order of messages. Also, detecting situations when a token-oriented (fine-grained) approach is not optimal, and using coarse-grained. Implementing more advanced graphical interface supporting all of the features, would also be beneficial for users.

Bibliography

[1] PLY (Python Lex-Yacc). http://www.dabeaz.com/ply/. accessed: 2018-03-13.

[2] Regex Generator++. http://machinelearning.inginf.units.it/data-and-tools. accessed: 2017-12-22.

[3] Riverbank Computing Limited. https://www.riverbankcomputing.com/software/pyqt/intro. accessed: 2018-04-15.

[4] Bartoli, A.; Lorenzo, A. D.; Medvet, E.; et al.: Inference of Regular Expressions for Text Extraction from Examples. *IEEE Transactions on Knowledge and Data Engineering*. vol. 28, no. 5. May 2016: pp. 1217–1230. ISSN 1041-4347. doi:10.1109/TKDE.2016.2515587.

[5] Delany, S. J.; Buckley, M.; Greene, D.: SMS spam filtering: Methods and data. *Expert Systems with Applications*. vol. 39, no. 10. 2012: pp. 9899 – 9908. ISSN 0957-4174. doi:https://doi.org/10.1016/j.eswa.2012.02.053.
Retrieved from:
http://www.sciencedirect.com/science/article/pii/S0957417412002977

[6] Denis, F.; Lemay, A.; Terlutte, A.: Learning regular languages using RFSAs. *Theoretical Computer Science*. vol. 313, no. 2. 2004: pp. 267 – 294. ISSN 0304-3975. doi:https://doi.org/10.1016/j.tcs.2003.11.008. algorithmic Learning Theory.
Retrieved from:
http://www.sciencedirect.com/science/article/pii/S0304397503006121

[7] Fernau, H.: Algorithms for learning regular expressions from positive data. *Information and Computation*. vol. 207, no. 4. 2009: pp. 521 – 541. ISSN 0890-5401. doi:https://doi.org/10.1016/j.ic.2008.12.008.
Retrieved from:
http://www.sciencedirect.com/science/article/pii/S0890540109000169

[8] García, P.; Ruiz, J.; Cano, A.; et al.: *Inference Improvement by Enlarging the Training Set While Learning DFAs*. Berlin, Heidelberg: Springer Berlin Heidelberg. 2005. ISBN 978-3-540-32242-9. pp. 59–70. doi:10.1007/11578079_7.
Retrieved from: https://doi.org/10.1007/11578079_7

[9] García, P.; de Parga, M. V.; Álvarez, G. I.; et al.: Universal automata and NFA learning. *Theoretical Computer Science*. vol. 407, no. 1. 2008: pp. 192 – 202. ISSN 0304-3975. doi:https://doi.org/10.1016/j.tcs.2008.05.017.
Retrieved from:
http://www.sciencedirect.com/science/article/pii/S0304397508003976

[10] Gilleland, M.: Levenshtein Distance, in Three Flavors.

[11] Gold, E. M.: Language identification in the limit. *Information and Control*. vol. 10, no. 5. 1967: pp. 447 – 474. ISSN 0019-9958.
doi:https://doi.org/10.1016/S0019-9958(67)91165-5.
Retrieved from:
http://www.sciencedirect.com/science/article/pii/S0019995867911655

[12] Gruber, H. J.; Holzer, M.: From Finite Automata to Regular Expressions and Back - A Summary on Descriptional Complexity. In *Int. J. Found. Comput. Sci.*. 2015.

[13] Iedemska, J.; Stringhini, G.; Kemmerer, R.; et al.: The Tricks of the Trade: What Makes Spam Campaigns Successful? In *2014 IEEE Security and Privacy Workshops*. May 2014. pp. 77–83. doi:10.1109/SPW.2014.21.

[14] ITU World Telecommunication: ICT Facts & Figures 2010.
http://www.itu.int/ITU-D/ict/material/FactsFigures2010.pdf. accessed: 2017-12-10.

[15] ITU World Telecommunication: ICT Facts & Figures 2015. https://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2015.pdf. accessed: 2017-12-10.

[16] Junczys-Dowmunt, M.: A Genetic Programming Experiment in Natural Language Grammar Engineering. In *15th International Conference on Text, Speech and Dialogue (TSD)*, *Lecture Notes in Computer Science*, vol. 7499, edited by P. Sojka; A. Horák; I. Kopecek; K. Pala. Brno, Czech Republic: Springer. 2012. pp. 336–344.
Retrieved from: http://emjotde.github.io/publications/pdf/mjd2012tsd2.pdf

[17] Murynets, I.; Piqueras Jover, R.: Crime Scene Investigation: SMS Spam Data Analysis. In *Proceedings of the 2012 Internet Measurement Conference*. IMC '12. New York, NY, USA: ACM. 2012. ISBN 978-1-4503-1705-4. pp. 441–452.
doi:10.1145/2398776.2398822.
Retrieved from: http://doi.acm.org/10.1145/2398776.2398822

[18] Needleman, S. B.; Wunsch, C. D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*. vol. 48, no. 3. 1970: pp. 443 – 453. ISSN 0022-2836.
doi:https://doi.org/10.1016/0022-2836(70)90057-4.
Retrieved from:
http://www.sciencedirect.com/science/article/pii/0022283670900574

[19] NumPy developers: NumPy. http://www.numpy.org/. accessed: 2018-03-05.

[20] OECD: Background Paper for the OECD Workshop on Spam.
doi:http://dx.doi.org/10.1787/232784860063.
Retrieved from: /content/workingpaper/232784860063

[21] Oncina, J.; Garcia, P.: Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, *Series in Machine Perception and Artificial Intelligence*, vol. 1, edited by N. P. de la Blanca; A. Sanfeliu; E. Vidal. World Scientific, Singapore. 1992. pp. 49–61.

[22] Rafique, M. Z.; Alrayes, N.; Khan, M. K.: Application of Evolutionary Algorithms in Detecting SMS Spam at Access Layer. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. GECCO '11. New York, NY, USA: ACM. 2011. ISBN 978-1-4503-0557-0. pp. 1787–1794. doi:10.1145/2001576.2001816. Retrieved from: `http://doi.acm.org/10.1145/2001576.2001816`

[23] Wyard, P.: Context free grammar induction using genetic algorithms. In *IEE Colloquium on Grammatical Inference: Theory, Applications and Alternatives*. Apr 1993. pp. P11/1–P11/5.

Appendix A

Program execution

```
tempex.py  [-h] [--imethod {s,r,sb,rb}] [--isamples ISAMPLES] [-o OUTPUT]]
           [--omethod {html}] [--vmethod {s,r}] [--vsamples VSAMPLES]
           [--gui]
           [input]
```

Positional arguments::

`input`	Input data set for template extraction. Mandatory unless '`--gui`' is set.

Optional arguments:

`-h, --help`	Show this help message and exit.
`--imethod {s,r,sb,rb}`	Input extraction method `[buffered][sequential|random]`. Default value is `'sb'`.
`--isamples ISAMPLES`	How many input samples extract. None for sequential means the whole file.
`-o OUTPUT` `--output OUTPUT`	Name of the output file. Stdout used if not set.
`--omethod {html}`	If not set, simple a template and validation result are printed. Value `html` produces HTML validation result on the output.
`--vmethod {s,r}`	Validation extraction method [sequential|random])
`--vsamples VSAMPLES`	How many samples should be used to validate the result. Reads the whole file in default.
`--gui`	Opens a simple graphical viewer.
`--presort`	Samples are ordered by number of tokens in decreasing fashion.
`--extraoptim`	If set, performs extra optimization of resulting RE.
`--verbose`	If set, more detailed output is provided.

Appendix B

CD Contents

```
/
├── data/                 ... Campaigns for demonstration
├── docs/                 ... Project documentation
│   ├── latex/            ... LaTeX source files
│   └── thesis.pdf        ... Thesis in PDF format
├── LICENCES/             ... License files for used libraries
└── src/                  ... Source files of the extraction tool
    ├── README.md
    ├── requirements.txt  ... List of dependencies
    ├── tempex.py         ... Main execution file
    └── tempex/           ... Modules and packages
```