Automata-Driven Stemming

Raphael Matori da Rocha[*] Wladmir Cardoso Brandão[†]

[*]Department of Software Engineering
Pontifical Catholic University of Minas Gerais
Belo Horizonte, MG, Brazil
raphael.matori@sga.pucminas.br

[†]Department of Computer Science
Pontifical Catholic University of Minas Gerais
Belo Horizonte, MG, Brazil
wladmir@pucminas.br

Abstract

Stemming is the process of reducing inflected words to their root form, the stem. Search engines have been using it to improve the user search experience. Stemming algorithms, or simply stemmers, usually adopt suffix stripping strategies to decide which reductions should be performed over words. They process a list of rules, particularly built from the constraints of each language, checking which word suffixes should be removed or replaced.

In this article, we introduce our automata-driven suffix stripping strategy, in which the flow of the stemming is driven by deterministic finite automata. In particular, we propose automata-driven stemmers for English, Portuguese and Spanish. To investigate the behavior of our stemmers in a real world scenario, we assess their efficiency by contrasting them with the state-of-the-art stemmers for each language.

Experiments on three datasets attest the efficiency of our stemmers, with reduction of up to 71.62% in average stemming time. Additionally, the gains is greater (73.10%) when no reduction is performed in a word. In some languages, such as English, many words have few morphological variations and should not be reduced. Moreover, suffix stripping approaches for other languages can be easily tuned to use deterministic finite automata, therefore increasing their performance.

1 Introduction

Search engines fulfill user information needs by providing an access point to relevant information in the Web. More than 3.3 billion user queries were processed every day, and over 30 trillion uniquely addressable documents were tracked by the leading commercial search engine (Cutts, 2012). The Web massive-scale demands time and retrieval performance to provide an outstanding search experience to users.

Stemming is an effective approach used by search engines to improve the user search experience by reducing query processing time and increasing recall. A stemming algorithm, or simply stemmer, reduces inflected words to their root form, the stem, i.e., the portion of the original word that remains after affixes removal (Baeza-Yates and Ribeiro-Neto, 2011). For instance, the word "engineer" is the stem of the words "engineering", "engineered", and "engineers". Particularly, stemming a document reduces its vocabulary size and, therefore, the time to process queries. Moreover, query reformulation by stemming increases the match between the query and relevant documents, improving recall.

In this article, we introduce our automata-driven stemmers for English, Portuguese and Spanish languages. In particular, they use deterministic finite automata (DFA) to reduce inflected words to their stem with a minimum number of character comparisons. The use of automata for stemming has already been proposed in literature (Kazem et al., 2005; Ram and Devi, 2010). Different from the previously proposed approaches, which use a single deterministic finite automata to perform word reductions, we tune well-known state-of-the-art suffix stripping approaches reported in literature by replacing the word reduction task driven by lists of rules by a word reduction task driven by multiple DFAs.

State-of-the-art stemmers reported in literature perform stemming by sequentially comparing suffixes in a preconceived list of rules and the word suffix to decide the reductions that should be performed in the word. Complexity analysis shows that the asymptotic notation for these stemmers is $\Theta(nm)$, where n is the number of words to reduce, and m is the number of suffixes in the list of rules. Additionally, the asymptotic notation for our stemmers is also $\Theta(nm)$, in the worst case, but $\Theta(n \log m)$ in the average case.

Complexity analysis do not describe the real behavior of our stemmers at the running time. Therefore, we assess their efficiency in a real world scenario by contrasting them with the state-of-the-art stemmers reported in literature (Jivani, 2011; Orengo and Huyck, 2001; Porter, 1980). Experiments using three datasets, one for each language, attest the efficiency of our automata-driven stemmers. Experimental results show that our stemmers outperformed state-of-the-art with reduction of up to 71.62% in average stemming time. In addition, the results show that the gains are even greater when no reduction is performed in a word, a frequent case for "non-romance languages", such as English, where words have few variations. Moreover, we can easily increase the performance of suffix stripping stemmers for other languages besides English, Portuguese and Spanish, by tuning them do use our automata-driven strategy. Particularly, the key contributions of this article are:

- We propose automata-driven stemmers for English, Portuguese and Spanish languages. The automata-driven strategy used in our stemmers is simple and can be adopted by other suffix stripping stemmers to improve their performance.

- We provide complexity analysis for our automata-driven stemmers and the state-of-the-art stemmers reported in literature (Jivani, 2011; Orengo and Huyck, 2001; Porter, 1980), highlighting the differences between them.
- We thoroughly evaluate our automata-driven stemmers in a real world scenario, contrasting them with the state-of-the-art baselines.

The remainder of this article is organized as follows: Section 2 reviews the related literature on automata and stemming. Section 3 presents our automata-driven stemmers, including an analysis of their computational complexity. Section 4 describes the setup, datasets and evaluation procedures used in the experimental evaluation of our stemmers. Section 5 shows experimental results, attesting the efficiency of our stemmers when compared with the state-of-the-art baselines reported in literature. Finally, Section 6 provides a summary of the contributions and the conclusions made throughout the other sections, presenting directions for future research.

2 Related Work

In this section, we review the related literature on automata and stemming. In particular, we present an overview on automata theory, focusing on the use of automata in information retrieval. Additionally, we present an overview on stemming strategies and approaches, highlighting the state-of-the-art suffix stripping approaches reported in literature.

2.1 Automata

Automata are abstract computing machines designed to automatically follow a preconceived sequence of operations (Hopcroft et al., 2000). The are several uses of automata theory to address computer science problems, such as software design, behavior of digital circuits, lexical analyses of compilers, text processing, and network security. Particularly in information retrieval, automata can be used on text processing to improve the performance on text representation, storage and retrieval (Kazem et al., 2005; Sipser, 2013). Automata-based approaches can also improve the efficiency on text processing by matching strings in constant time (Jiang et al., 2009).

A particular well-known type of automata is deterministic finite automata (DFA), finite-state machines composed by states and transitions. In DFAs, when the automaton get a symbol as input in its initial state, it makes a transition to another state according to a transition function, and the transitions continue until a final state is achieved. DFAs accept or reject a finite number of symbols and has a unique deterministic computation for each input string (Sipser, 2013). They are mathematically represented by the 5-tuple (Q, Σ, δ, q_0, F), where Q is a finite set of states, Σ is a finite set of input symbols (alphabet),

δ is the transition function, where δ: $Q \times \Sigma \Rightarrow Q$, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final (accept) states.

In the following, we provide an example on how a DFA decides to accept or reject a given sequence of symbols. Consider $\{a_1, a_2, \cdots, a_n\}$ a sequence of input symbols and q_0 the start state. First, we check the transition function δ, for example $\delta(q_0, a_1) = q_1$, to find the state that the DFA enters after processing the first input symbol a_1. Second, we process the next input symbol, a_2, by evaluating $\delta(q_1, a_2) = q_2$. We continue by finding states q_3, q_4, $\cdots$, q_n such that $\delta(q_i - 1, a_i) = q_i$ for each i. If $q_n \subseteq F$, then the sequence of input symbols is accepted, else it is rejected.

Recently, DFA was used for multi-string matching in high performance network intrusion detection system (Jiang et al., 2009), and for stemming (Kazem et al., 2005). The stemming approach represents the DFA by using a two-dimensional array with rows representing states and columns representing input letters. Every state is an accept state and the DFA decides which suffix group should be compared by using the state that the DFA halted. Therefore, DFA avoid comparing a long list of strings, providing gains of up to 18% in average stemming time.

2.2 Stemming

From the late 1970s, the demand of efficient algorithms to store and process an increasing volume of data promoted an increase in the research interest in the information retrieval field (van Rijsbergen, 1979). Information retrieval comprises the representation, storage, organization of, and access to information items (Baeza-Yates and Ribeiro-Neto, 2011). Information retrieval systems are mechanisms to index, search and classify documents, which must efficiently store indexed documents and retrieve relevant results to users (Singh and Saini, 2014).

The huge amount of text in the Web that must be entirely indexed by the search engines demands an effective document preprocessing procedure to improve user experience. Document preprocessing comprises different text operations, such as lexical analysis, stop-words removal, stemming, and selection of index terms (Baeza-Yates and Ribeiro-Neto, 2011). Particularly, stemming has been used in several applications, such as natural language processing and information retrieval (Sharifloo and Shamsfard, 2008). Stemmers reduce a word to its stem by detecting different inflections and derivations of morphological variants of words (Frakes and Baeza-Yates, 1992). A challenging problem in stemming is that different strategies potentially produce over and under stemming. Over-stemming is a false-positive case when a word is reduced beyond the necessary. In this case, two words with different stems can be reduced to the same stem. Under-stemming is a false-negative case when a word is reduced less than necessary. In this case, two words with the same stem can be reduced to different stems (Jivani, 2011).

There are several stemmers, for different languages, reported on literature (Bijal and Sanket, 2014; Honrado et al., 2000; Kazem et al., 2005; Orengo and Huyck, 2001; Sharifloo and Shamsfard, 2008). Each one of them present different strategies to reduce words, being more or less efficient in time and effective in avoiding over and under stemming.

In particular, we classify these algorithms according with the stemming strategy they use: table lookup, successor variety, n-grams, and affix removal (Baeza-Yates and Ribeiro-Neto, 2011).

2.2.1 Stemming Strategies

The most simple stemming strategy is table lookup. In this strategy, we look for a word in a preconceived table with all possible words and their respective stems. Table 1 shows an example of a table with words and stems.

Word	Stem
engineering	engineer
engineered	engineer
engineers	engineer

Table 1: Example of a table used by the table lookup stemming strategy.

From Table 1, we observe that the stem of the words “engineering”, “engineered”, and “engineers” is “engineer”, while the word “engineer” should not be reduced, since it does not appear in the lookup table. Although simple, this strategy assumes that we know all the possible words that should be reduced. Besides that, we have to create and store the lookup table in memory (Frakes and Baeza-Yates, 1992).

Another stemming strategy is successor variety, which uses heuristic to analyze prefix frequency and length among the variants of a word (Stein and Potthast, 2007). Figure 1 present a suffix tree used by the successor variety strategy.

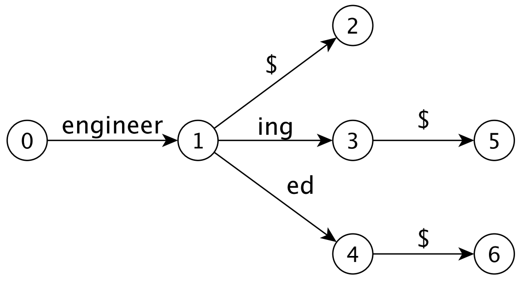

Figure 1: A suffix tree for the words ”engineer”, ”engineered”, ”engineering”. \$-sign denotes the end of string.

From Figure 1, we observe that, for the words “engineering”, “engineered”, and “engineer”, the longest common prefix is “engineer”, so it is the stem for the three words. This strategy is language independent and demands a large vocabulary, since it is based in statistical rules.

Another language independent stemming strategy is n-grams. Different from successor variety, it uses statistical analysis to identify frequent n-grams from words (Jivani, 2011). The intuition is that similar words share a higher proportion of n-grams. Table 2 shows the frequencies of different n-grams of the word "engineering". The frequencies were extracted from the Corpus of Contemporary American English (COCA)[1].

n-gram	frequency	n-gram	frequency
engine$	54,460	engineeri$	21,329
enginee$	32,737	engineerin$	21,239
engineer$	32,737	engineering$	21,238

Table 2: n-grams frequencies of the word "engineering" from the Corpus of Contemporary American English (COCA).

From Table 2 we observe that the 9-gram "engineeri$" has a relative high frequency across the corpus. All the n-grams, with $n < 9$, are highly frequent, and with $n > 9$ tend to exhibit equal or gradually lower frequencies, ending in the total frequency of the word. Thus, the 9-gram "engineeri$" is an approximation of the stem for the word "engineering". The n-grams strategy identifies word's inflection by analyzing the frequency of word's n-grams (Jivani, 2011). It provides high accuracy, but demands the preprocessing of a large corpus of documents to perform stemming.

The affix removal strategy removes or replaces prefixes and suffixes of a word (Bijal and Sanket, 2014). In particular, it provides a set of prefixes and suffixes along with context-sensitive rules that must be checked and applied to reduce a word to its stem (Singh and Gupta, 2016). For example, applying the rule[2] *if(word.endsIn(al, ance, ic, able, ou, ous, ize).removeSuffix* over the words "generic", "general", and "generous", we get the same stem "gener". In this case, three words with distinct stems are reduced to the same stem, a typical case of over-stemming that produces aggressive conflations. The affix removal strategy is language dependent and potentially produces over and under stemming (Jivani, 2011). However, it is simple and provides a significantly increased accuracy compared with the other strategies, requiring no preprocessing of large corpus or large vocabularies.

2.2.2 State-of-the-art Stemming Approaches

There are several stemming approaches for different languages (Bijal and Sanket, 2014). The effective ones perform suffix stripping on words of simple morphology languages, such as West European languages (Jivani, 2011). For English, one of the first stemmer reported in literature uses a context-sensitive longest-match approach, which processes a list of the most frequent suffixes of the language, removing from the word the longest ones (Lovins,

[1] `http://corpus.byu.edu/coca`

[2] This rule is a sample of the rule in the step *4* of the Porter algorithm for English stemming (Porter, 1980).

1968). Porter proposed a suffix stripping approach for English by observing that many English suffixes comprise a combination of smaller and simpler suffixes (Porter, 1980). The Porter approach divides the stemming task in five reduction steps, where different rules are checked in each step to decide if reductions should be performed. Additionally, Porter proposed the Snowball framework for stemming, allowing researchers to propose their own stemming approaches for other languages (Jivani, 2011). Nowadays, the Porter approach is the most popular and the state-of-the-art stemmer for English.

For Portuguese stemming, the steps and rules differ significantly from English. For example, "ão" is commonly used in Portuguese to denote augmentative. However, not all the words ending in "ão" are in augmentative form. Additionally, there are others challenges in Portuguese stemming, such as homographs, irregular verbs, and variant morphological root. The RSLP stemming approach (Orengo and Huyck, 2001) was proposed to overcome limitations of the previously proposed Portuguese stemming approaches (Alvares et al., 2005; Porter, 1980). Particularly, it divides the stemming task in seven reduction steps that must be performed in a specific order. Figure 2 presents the stemming flow of RSLP.

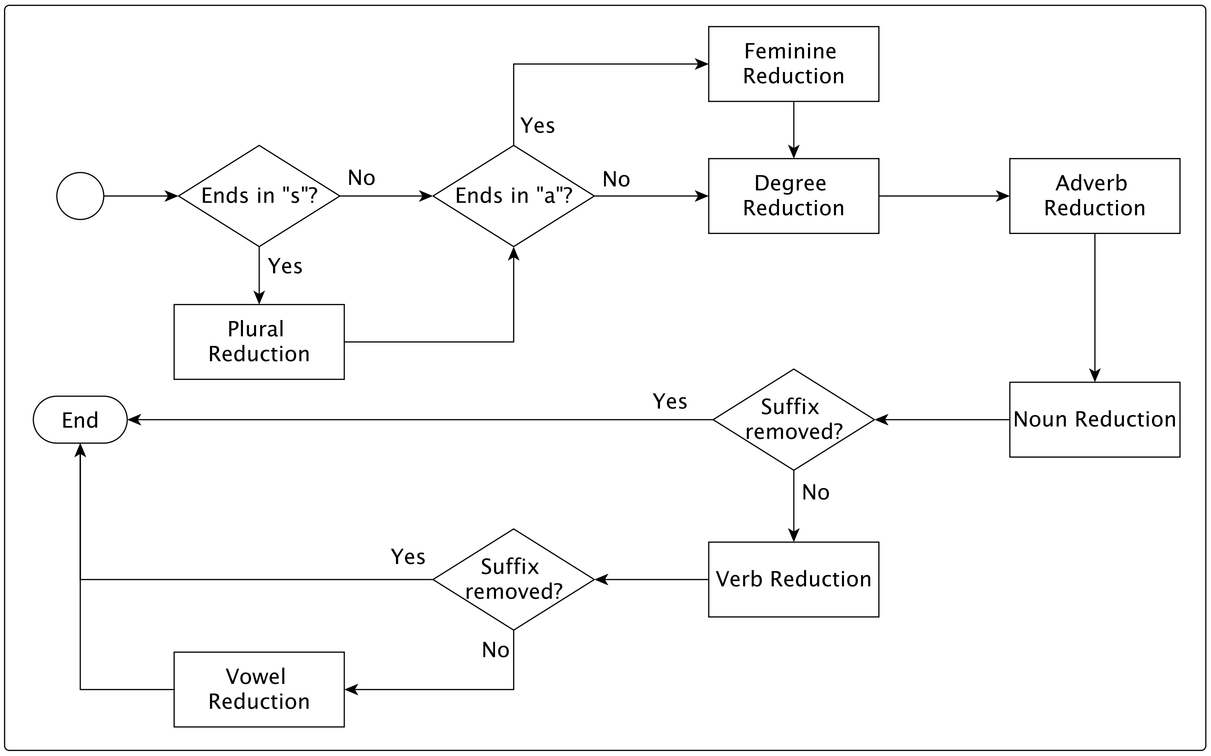

Figure 2: The RSLP stemmer workflow.

The RSLP stemmer is 25% more effective than the Snowball stemming approach for Portuguese. Therefore, RSLP is the state-of-the-art stemmer for Portuguese. Similarly to Portuguese, the Spanish language is grammatically complex with a lot of injections that change the root of words, not only in their ending. The state-of-the-art Spanish stemming approach reported in literature, here referenced by Spanish Snowball, combines dictionary lookups and hundreds of rules for suffix stripping (Honrado et al., 2000).

Both Porter, RSLP and the Spanish Snowball stemmer are suffix stripping approaches based on rules checking. They compare word suffixes with suffixes in a list of rules to decide the reductions that should be performed. Their computational cost is proportional to the number of words to process (n), the number of suffixes in the list of rules (m), and the size, in characters, of the longest suffix in the list (k). Thus, the processing time for these approaches is proportional to

$$n \sum_{i=1}^{m} k \tag{1}$$

From the Equation 1, we observe that, these stemmers run as fast as the smaller the number and size of suffixes in the list of rules. Considering the size of the longest suffix in the list a constant, the asymptotic notation for these aproaches is $\Theta(nm)$.

3 Automata-Driven Stemmer

Automata-driven stemmers are suffix stripping approaches that use deterministic finite automata (DFA) to drive the flow of the stemming process, deciding the reductions that should be performed in each step. Different from list-driven approaches, such as Porter, RSLP and Spanish Snowball stemmers, which must sequentially process several rules in each step to decide whether remove suffixes, the automata-driven stemmer uses DFA to eliminate the unnecessary rules checking, thus reducing the number of character comparisons to the minimum. Figure 3 shows the DFA for the step 3 of the Porter stemmer, which removes the “icate”, “ative”, “alize”, “ical”, “ful”, “iciti”, and “ness” suffixes from a word.

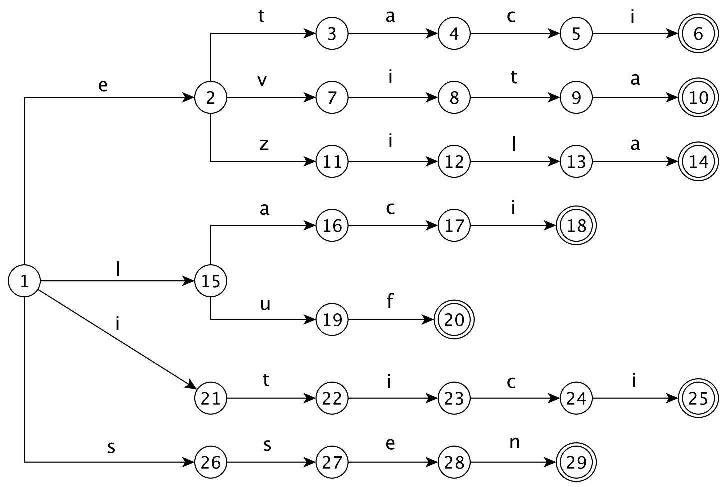

Figure 3: Deterministic finite automata for the step 3 of processing of the Porter stemmer.

From Figure 3, we observe that the automata-driven stemmer works backwards, hence, we must follow the DFA starting by one of the accept states (double border circles) and walk back until the first state (circle 1). For example, to achieve the reduction "ness", we must start in the final state 29 and walk back until the first state.

Different from list-driven stemmers, which check first the longest suffix to avoid wrong reductions, automata-driven stemmers check first the shorter possible reduction, and enter in an accept state before continue. In addition, by working backwards, it creates a cluster using a common character ending for each interaction, i.e., for each iteration (character), automata-driven stemmers divide the problem to n subproblems of different word endings. From Figure 3, we observe that state 1 reduces the problem to 4 subproblems, state 2 reduces to 3 subproblems, and state 15 reduces to 2 subproblems.

The computational cost of the automata-driven stemmers is proportional to the number of words to process (n), the number of DFAs (p), and the length of the longest path in the DFAs, which is, in the worst case, the size, in characters, of the longest suffix to process (k). Thus, the processing time for these approaches is proportional to

$$n \sum_{i=1}^{p} k \tag{2}$$

From the Equation 2, we observe that these stemmers run as fast as the smaller the number of DFAs and the size of the suffixes. Remember that, automata-driven stemmers divide the problem of size m to $p \ll m$ subproblems. In particular, in the worst case, $p = m$, and in the average case, $p \approx \log m$. Thus, considering the size of the longest suffix a constant, the asymptotic notation for automata-driven stemmers in the worst case is $\Theta(nm)$, and in the average case is $\Theta(n \log m)$.

The asymptotic notation does not describes the real behavior of an algorithm at the running time. In a real world scenario, the upper bound can be overestimate. Therefore, to investigate the behavior of automata-driven stemmers in a real world scenario, we propose three automata-driven stemmers, for English, Portuguese and Spanish, and contrast them with state-of-the-art list-driven baselines reported in literature (Jivani, 2011; Orengo and Huyck, 2001; Porter, 1980). In the next two sections, we describe experimental setup and present experimental results.

4 Experimental Setup

In this article, we introduce three different stemming approaches, the automata-driven stemmer for English (EnADS), the automata-driven stemmer for Portuguese (PtADS), and the automata-driven stemmer for Spanish (SpADS)[3]. We validate our proposed stemmers by contrasting them with state-of-the-art baselines on different datasets. Particularly, in this section we answer the following research questions:

[3]The code of our automata-driven stemmers are available at `http://www.wladmirbrandao.com/en/stemming`.

- Are our proposed automata-driven stemmers more efficient than the state-of-the-art baselines?
- How our proposed stemmers perform in each stemming step?

Note that, as mentioned in Section 3, our stemmers and the baselines performs the same word reductions, presenting the same accuracy. Our stemmers differ from baselines in the way they compare suffixes. Therefore, we focus our analysis on efficiency.

4.1 Baselines

We contrast our proposed automata-driven stemmers with state-of-the-art baselines reported in literature. For English, the Porter stemmer (Porter, 1980) is the baseline (EnBsl). For Portuguese, the RSLP stemmer (Orengo and Huyck, 2001) is the baseline (PtBsl), and for Spanish, the Spanish Snowball stemmer (Jivani, 2011) is the baseline (SpBsl).

Each baseline performs stemming by sequentially processing a fixed number of reduction steps. In each step, a word is transformed in another one, if necessary. After the last step, the last word reduction provides the stem. For the EnBsl, we have eight reduction steps, while for the PtBsl we have seven reduction steps, and for SpBsl we have five reduction steps. Table 3 show, for each baseline, examples of steps and suffixes.

Baseline	Step	Suffixes
EnBsl	1a	s ss ies sses
EnBsl	1b	e ed at ing
EnBsl	2	tion ation ence ational
PtBsl	Plural	s ns ais res
PtBsl	Degree	ão aço inho zinho
SpBsl	0	la lo les nos
SpBsl	2a	ya ye yo yendo

Table 3: Reduction steps and suffixes for the baselines.

From Table 3, we observe that the number of suffixes that must be checked is different in each step and for each baseline. In particular, there are steps with a single suffix, such as the step *1c* of EnBsl, and steps with a long list of suffixes, such as the step *Verb* of PtBsl, with 89 suffixes.

4.2 Datasets

To assess the efficiency of our stemmers and the baselines, we built the EnWiki, PtWiki and SpWiki datasets. They are collections of words extracted from the English, Portuguese and Spanish Wikipedia[4], respectively. In particular, each dataset is composed of words

[4]We used the Wikipedia dump files from 09-20-2016.

extracted from the Wikipedia articles, excluding those ones with three or less characters, XML marks, and punctuation. Table 4 shows the size of each dataset. In addition, it shows the number of words that should (Reduction) and should not (No Reduction) be reduced by the stemmers in each dataset.

Dataset	Size (GB)	# words			
		Reduction		No Reduction	
EnWiki	8.42	550,285,786	(47.82%)	600,562,670	(52.18%)
PtWiki	1.14	123,789,555	(85.39%)	21,187,284	(14.61%)
SpWiki	2.42	252,973,534	(82.17%)	54,885,204	(17.83%)

Table 4: The size of the datasets and the number of words to reduce (or not).

From Table 4, we observe that in EnWiki, approximately 47.82% of the 1,150,848,456 words in the dataset should be reduced, while in PtWiki the percentage is approximately 85.39% of the 144,976,839 words, and in SpWiki the percentage is approximately 82.17% of the 307,858,738 words. The English language presents fewer words to be reduced than the "romance languages", such as Portuguese and Spanish, which have several variations for the same word. Note that, in both three languages, one word can be reduced several times to produce its stem.

4.3 Evaluation Procedures

We use the EnWiki dataset to validate the efficiency of EnADS and EnBsl. Similarly, we use the PtWiki dataset to validate the efficiency of PtADS and PtBsl, and the SpWiki dataset to validate the efficiency of SpADS and SpBsl. We report efficiency in terms of average stemming time (AST), i.e., the average elapsed time in microseconds (μs) to obtain stems for words. We ignore the time to load suffix rules in memory to ensure a fair assessment of our stemmers and the baselines. The results are averages on 10 trials and significance is verified with a two-tailed paired t-test (Jain, 1991).

The experiments were carried out on a computer with a 64-bit Intel i5 quad-core 1.6 GHz processor, 8 GB of RAM memory and one SSD disk of 120 GB.

5 Experimental Results

In this section, we describe the experiments we have carried out to evaluate our proposed automata-driven stemmers. In particular, we address the two research questions stated in Section 4, by contrasting the efficiency of our stemmers with state-of-the-art baselines, described in Section 4.1. As mentioned in Section 4.3, significance is verified with a two-tailed paired t-test (Jain, 1991), with the symbol $\blacktriangle$ ($\blacktriangledown$) denoting a significant increase (decrease) in average stemming time at the $p < 0.01$ level, and the symbol $\bullet$ denoting no significant difference.

5.1 Stemming Efficiency

In this section, we address our first research question, by assessing the overall efficiency of our automata-driven stemmers. To this end, Table 5 shows the performance of our automata-driven stemmers compared with the baselines in three scenarios: i) Reduction, using all the words in the dataset, in which at least one reduction is performed; ii) No Reduction, using all the words in the dataset, in which no reduction is performed; iii) All, using all the words in the dataset. For all automata-driven stemmers, percentage improvement figures compared with the baselines are also shown.

Stemmer	AST (μs)					
	Reduction		No Reduction		All	
EnBsl EnADS	126.056 103.984	(-17.50%) ▼	530,779 406.013	(-23.50%) ▼	656.835 509.997	(-22.35%) ▼
PtBsl PtADS	147.554 46.862	(-68.24%) ▼	43.057 23.020	(-46.53%) ▼	190.611 69.882	(-63.33%) ▼
SpBsl SpADS	185.591 61.428	(-66.90%) ▼	590.448 158.788	(-73.10%) ▼	776.039 220.216	(-71.62%) ▼

Table 5: Overall efficiency of the automata-driven stemmers and the baselines.

From Table 5, we observe that our automata-driven stemmers significantly improves on all baselines and across all reduction scenarios. In particular, the gains in average stemming time are 71.62% for the Spanish language, 63.33% for the Portuguese language, and 22.35% for the English language. Additionally, we observe that, for the English and Spanish languages, the gains are greater in the scenario in which no reduction is performed, 23.50% and 73.10%, respectively. For these languages, the baselines perform all the stemming steps, and the worst case occurs when no reduction is performed. For Portuguese, from Figure 2, we observe that not all steps are performed, and the worst case does not occurs when no reduction is performed. Recalling our first research question, these observations attest the efficiency of our automata-driven stemmers.

5.2 Stemming Efficiency by Steps

In this section, we address our second research question, by assessing the efficiency of our automata-driven stemmers and baselines in each stemming step.

5.2.1 English Stemming

There are eight steps for English stemming. Table 6 shows, for each stemming step, the number and percentages of word reductions, and the performance of our automata-driven stemmer and the baseline.

Step	# word reductions		AST (μs)		
			EnBsl	EnADS	
1a	163,166,672	(24.62%)	19,910	22,293	(+11.97%) ▲
1b	120,863,700	(18.24%)	28,514	23,628	(-17.13%) ▼
1c	85,439,398	(12.90%)	16,480	13,627	(-17.31%) ▼
2	34,265,461	(5.17%)	3,519	2,428	(-31.00%) ▼
3	11,664,232	(1.76%)	1,450	490	(-66.20%) ▼
4	117,885,231	(17.79%)	30,403	20,506	(-32.55%) ▼
5a	123,646,949	(18.66%)	24,207	19,924	(-17.69%) ▼
5b	5,722,816	(0.86%)	1,570	1,085	(-30.89%) ▼
Total	662,654,459		126,056	103,984	(-17.50%) ▼

Table 6: Efficiency of our English automata-driven stemmer and the baseline in each stemming step

From Table 6, we observe that the total number of word reductions is greater than the number of words in the dataset reported in Table 4. Remember that, one word can be reduced several times in different stemming steps. In addition, we observe that our automata-driven stemmer outperforms the baseline in almost all stemming steps, with reduction in average stemming time from 17.13% (step *1b*) to 66.20% (step *3*). The exception is step *1a*, with an increase of 11.97%. Table 7 shows the top-k most frequent suffixes that must be checked in each reduction step and their frequency percentage. The steps *1c*, *5a*, and *5b* were omitted, since only one suffix is checked.

k	Step									
	1a		1b		2		3		4	
1	s	(90.11%)	e	(48.77%)	ation	(29.77%)	ical	(44.12%)	er	(19.08%)
2	ies	(5.01%)	ed	(27.30%)	tion	(21.54%)	icate	(16.80%)	al	(19.03%)
3	ss	(4.41%)	ing	(15.79%)	alli	(10.15%)	ative	(15.20%)	ate	(17.39%)
4	sses	(0.47%)	l	(4.14%)	ence	(8.36%)	ness	(10.76%)	ic	(10.13%)

Table 7: Top-4 most frequent suffixes that must be checked in each reduction step by the Porter stemmer.

From Table 7 we observe that more than 90% of the word reductions performed in step *1a* are very simple, demanding a single character comparison. In this case, performing word reduction using list of rules is more efficient than performing word reduction using DFA. One important constraint of the baseline is that longest suffixes must be processed first to assure that they will be removed, if necessary. However, the prior processing of less frequent suffixes has a negative impact on efficiency. For example, from Table 7 we observe that for step *1a*, "s" is the most frequent suffix, occuring 90.11% of times and, for efficiency, should be processed first. However, EnBsl must first processes "sses", "ies", and "ss" in this restrict order to assure that the longest suffixes will be removed. Different from the baseline, EnADS checks suffixes incrementally. For example, in step *1a*, after checking "s", the process only continues if the next character is "e" (ies) or "s" (sess). Thus, this incremental processing improves efficiency.

In seven of eight stemming steps our automata-driven stemmer outperforms the baseline. Recalling our second research question, these observations attest the efficiency of our automata-driven stemmer in different steps of English stemming.

5.2.2 Portuguese Stemming

There are seven steps for Portuguese stemming. Table 8 shows, for each stemming step, the number and percentages of word reductions, and the performance of our automata-driven stemmer and the baseline.

Step	# word reductions		AST (μs)		
			PtBsl	PtADS	
Plural	27,183,868	(16.51%)	9,036	5,521	(-38.89%) ▼
Feminine	9,271,025	(5.63%)	4,397	3,128	(-28.86%) ▼
Degree	7,263,874	(4.41%)	20,476	13,151	(-35.77%) ▼
Adverb	1,442,800	(0.88%)	3,072	2,183	(-28.93%) ▼
Noun	30,517,849	(18.53%)	45,134	12,996	(-71.20%) ▼
Verb	24,175,254	(14.68%)	62,763	7,733	(-87.67%) ▼
Vowel	64,802,473	(39.36%)	2,676	2,150	(-19.65%) ▼
Total	164,657,143		147,554	46,862	(-68.24%) ▼

Table 8: Efficiency of our Portuguese automata-driven stemmer and the baseline in each stemming step

From Table 8, we observe that our automata-driven stemmer outperforms the baseline in all stemming steps, with reduction in average stemming time from 19.65% (*Vowel*) to 87.67% (*Verb*). Table 9 shows the top-k most frequent suffixes that must be checked in each reduction step and their frequency percentage. The steps *Adverb* and *Vowel* were omitted, since only one suffix is checked.

From Table 9 we observe that the frequency of suffixes is well distributed in the steps *Feminine*, *Noun* and *Verb*, and concentrated in the steps *Plural* and *Degree*. From Table 7 we also observe a concentration in the step *1a* and a well-distribution of frequencies in the other steps. The frequency distribution of suffixes impacts on the performance of the stemmers. We observe that automata-driven stemmers outperform list-driven stemmers more intensely in steps with a well-distribution of frequencies. In this case, while automata-driven stemmers demand a very small number of character comparisons to reduce a word, list-driven stemmers must compare the word suffix with a large number of suffixes in the list of rules.

k	Step									
	Plural		Feminine		Degree		Noun		Verb	
1	s	(78.89%)	ada	(23.42%)	ão	(93.64%)	ado	(16.23%)	ia	(15.65%)
2	ões	(5.39%)	na	(19.97%)	inho	(2.94%)	al	(8.24%)	er	(12.37%)
3	res	(4.36%)	ica	(17.51%)	aço	(1.25%)	ico	(7.84%)	i	(10.77%)
4	ais	(4.30%)	eira	(8.43%)	arra	(0.61%)	aç	(7.26%)	ar	(10.37%)

Table 9: Top-4 most frequent suffixes that must be checked in each reduction step by the RSLP stemmer.

In all stemming steps our automata-driven stemmer outperforms the baseline. Recalling our second research question, these observations attest the efficiency of our automata-driven stemmer in different steps of Portuguese stemming.

5.2.3 Spanish Stemming

There are five steps for Spanish stemming. Table 10 shows, for each stemming step, the number and percentages of word reductions, and the performance of our automata-driven stemmer and the baseline.

Step	# word reductions		AST (μs)		
			SpBsl	SpADS	
0	23,229,283	(8.70%)	1,836	570	(-68.95%) ▼
1	23,341,764	(8.74%)	22,537	4,370	(-80.60%) ▼
2a	998,108	(0.37%)	305	36	(-88.19%) ▼
2b	76,393,797	(28.60%)	118,418	12,321	(-89.59%) ▼
3	143,116,654	(53.59%)	42,495	44,131	(+3.85%) ●
Total	267,079,606		185,591	61,428	(-66.90%) ▼

Table 10: Efficiency of our Spanish automata-driven stemmer and the baseline in each stemming step

From Table 10, we observe that our automata-driven stemmer outperforms the baseline in four of five stemming steps, with reduction in average stemming time from 68.95% (step *0*) to 89.59% (step *2b*). In the step *3*, there is no significant difference between our automata-driven stemmer and the baseline. Table 11 shows the top-k most frequent suffixes that must be checked in each reduction step and their frequency percentage.

From Table 11 we observe that, different from English and Portuguese, no step for Spanish stemming presents a frequency concentration in few suffixes. Additionally, in step *3* there are very simple suffixes with short length. In this case, automata-driven and list-driven stemmers present similar computational cost, since the number of character comparisons are too small.

k	Step									
	0		1		2a		2b		3	
1	la	(16.30%)	ación	(13.85%)	ya	(21.16%)	es	(19.71%)	a	(32.83%)
2	les	(15.34%)	ica	(8.43%)	ye	(18.06%)	as	(16.05%)	o	(32.73%)
3	lo	(14.39%)	idad	(7.74%)	yo	(17.89%)	ar	(6.77%)	e	(19.41%)
4	nos	(11.22%)	mente	(6.33%)	yó	(12.46%)	ado	(6.73%)	os	(10.37%)

Table 11: Top-4 most frequent suffixes that must be checked in each reduction step by the Spanish Snowball stemmer.

In four of five stemming steps our automata-driven stemmer outperforms the baseline. Additionally, in the remaining stemming step there is no significant difference between our automata-driven stemmer and the baseline. Recalling our second research question, these observations attest the efficiency of our automata-driven stemmer in different steps of Spanish stemming.

6 Conclusion

We introduced novel automata-driven stemmers for English, Portuguese and Spanish. Our stemmers use deterministic finite automata to drive the flow of the stemming process. In contrast with state-of-the-art suffix stripping stemmers, not only do our stemmers eliminated the processing of a long list of suffix rules, but they also reduce the number of character comparisons to the minimum.

The state-of-the-art suffix stripping stemmers perform stemming by sequentially comparing suffixes in a preconceived list of rules and the word suffix to decide the reductions that should be performed in the word. Complexity analysis shows that the asymptotic notation for these stemmers is $\Theta(nm)$, where n is the number of words to reduce, and m is the number of suffixes in the list of rules. Additionally, the asymptotic notation for our stemmers is also $\Theta(nm)$, in the worst case, but $\Theta(n \log m)$ in the average case.

We thoroughly evaluated our proposed stemmers in a real world scenario by contrasting them with the state-of-the-art baselines reported in literature using three datasets. The results of this evaluation attest the efficiency of our stemmers, with reduction of 71.62% in average stemming time over the Spanish baseline, 63.33% over the Portuguese baseline, and 22.35% over the English baseline. In addition, by breaking down our analysis into different processing steps, we demonstrated the robustness of our stemmers, which outperformed the baselines in 18 of the 20 stemming steps. Moreover, we demonstrated that the incremental character processing of our stemmers prevents the prior processing of less frequent suffixes, improving their efficiency over the baselines.

For future work, we plan to improve our stemmers to achieve $O(n \log m)$ computational cost, by using a bit level suffix comparison instead of a character level comparison to balance the paths in the deterministic finite automata.

7 Acknowledgments

The authors are thankful for the partial support given by the Pontifical Catholic University of Minas Gerais (Grant PUCMINAS-FIP 2016/11086-S2), MCTI/CNPq (Grant 444156/2014-3) and the authors' individual grants and scholarships from CNPq.

References

Alvares, R. V., Garcia, A. C. B., and Ferraz, I. (2005). STEMBR: A stemming algorithm for the brazilian portuguese language. In *Proceedings of the 12th Portuguese Conference on Artificial Intelligence*, EPIA'05, pages 693–701.

Baeza-Yates, R. and Ribeiro-Neto, B. (2011). *Modern information retrieval: the concepts and technology behind search*. Pearson Education, 2nd edition.

Bijal, D. and Sanket, S. (2014). Overview of stemming algorithms for indian and non-indian languages. *International Journal of Computer Science and Information Technologies*, 5(2):1144–1146.

Cutts, M. (2012). Spotlight keynote. In *Proceedings of Search Engines Strategies*, San Francisco, CA, USA.

Frakes, W. B. and Baeza-Yates, R. (1992). *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, 1st edition.

Honrado, A., Leon, R., O'Donnel, R., and Sinclair, D. (2000). A word stemming algorithm for the spanish language. In *Proceedings of the 17th International Symposium on String Processing and Information Retrieval*, SPIRE'10, pages 139–145.

Hopcroft, J. E., Motwani, R., Rotwani, and Ullman, J. D. (2000). *Introduction to Automata Theory, Languages and Computability*. Addison-Wesley, 2nd edition.

Jain, R. (1991). *The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling*. Wiley.

Jiang, J., Tang, Y., Liu, B., Wang, X., and Xu, Y. (2009). SPC-FA: Synergic parallel compact finite automaton to accelerate multi-string matching with low memory. In *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS'09, pages 163–164.

Jivani, A. G. (2011). A comparative study of stemming algorithms. *International Journal of Computer Technology and Applications*, 2(6):1930–1938.

Kazem, T., Russell, B., and Mohammad, S. (2005). A stemming algorithm for the farsi language. In *Proceedings of the International Conference on Information Technology: Coding and Computing*, ITCC'05, pages 158–162.

Lovins, J. B. (1968). Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11(1,2):22–31.

Orengo, V. M. and Huyck, C. (2001). A stemming algorithm for the portuguese language. In *Proceedings of the 8th International Symposium on String Processing and Information Retrieval*, SPIRE'01, pages 186–193.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, 14(3):130–137.

Ram, V. S. and Devi, S. L. (2010). Malayalam stemmer. In *Proceedings of the Knowledge Sharing on Morphological Analysers and Generators*, pages 105–113.

Sharifloo, A. A. and Shamsfard, M. (2008). A bottom up approach to persian stemming. In *Proceedings of the 3th International Joint Conference on Natural Language Processing*, ACL'08, pages 583–588.

Singh, J. and Gupta, V. (2016). Text stemming: Approaches, applications, and challenges. *ACM Computing Surveys*, 49(3):45:1–45:46.

Singh, V. and Saini, B. (2014). An effective pre-processing algorithm for information retrieval systems. *International Journal of Database Management Systems*, 6(6):13–24.

Sipser, M. (2013). *Introduction to the Theory of Computation.* Cenage Learning, 3th edition.

Stein, B. and Potthast, M. (2007). Putting successor variety stemming to work. In *Proceedings of the 30th Annual Conference of the German Classification Society*, GFKL'07, pages 367–374.

van Rijsbergen, C. J. (1979). *Information Retrieval.* Butterworth-Heinemann, 2nd edition.