A SUFFIX SUBSUMPTION-BASED APPROACH TO BUILDING STEMMERS AND LEMMATIZERS FOR HIGHLY INFLECTIONAL LANGUAGES WITH SPARSE RESOURCES

Vlado Kešelj,
Dalhousie University

Danko Šipka,
Arizona State University

Abstract: We present a general suffix-based method for construction of stemmers and lemmatizers for highly inflectional languages with only sparse resources. The process is directly implementable with described efficient design and it is evaluated on a construction of a stemmer for the Serbian language. The evaluation on real data has shown an accuracy of 79%.

1 Introduction

Two important tasks at the low level of Natural Language Processing (NLP) are stemming and lemmatization. Stemming is well-known in the NLP, IR (Information Retrieval), and Text Mining research areas as an essential preprocessing step for some tasks, such as text and document retrieval, document clustering, classification, information extraction, and other content-related applications. Descriptively speaking, stemming is a word transformation in which a word may be stripped of some suffixes without loosing its core semantic content. Very frequent words are usually removed as stop-words in an IR system, and they are not subject to stemming. We could think of stemming as a process of normalization in which several morphological variants of a word are mapped into the same form. An elaborate discussion about stemming and its application to IR is given in [7]. Stemming brings two important benefits to an IR system: (1) a better IR recall can be achieved since query words are matched with their variants in the documents, and (2) stemming decreases the size of the overall term vocabulary, which leads to significant efficiency benefits in speed and memory requirements, due to decreased size of the term index and dimensionality of term vectors. Namely, in the vector-space model of IR, the documents are represented as vectors of weights, where each weight corresponds to a term in a vocabulary. Removing stop-words and very rare words from the vocabulary is the first step in dimensionality reduction, and on top of this, further reduction by stemming is estimated to be about one third [6]. Significant benefits in retrieval performance are sometimes disputed ([4] §3.4), at least for English, but for highly inflectional languages stemming or some equivalent preprocessing is essential [5]. Stemming can also be used as a preprocessing step in information extraction, and various other tasks.

Is Suffix stripping sufficient? Beside suffix removal, one could be tempted to use prefix removal as well, but prefixes usually change word meaning radically and it is preferred that they are left intact [7]. Stemming has been a mainly suffixbased transformation since the publication of the Porter's stemmer [6], and it has been successfully applied to several other languages of Indo-European family; e.g., stemmers for 16 languages are implemented in the Snowball framework [7]. However, one should not generalize this suffix-oriented methodology to all languages; for example Arabic relies on prefixes, suffixes, and infixes in morphological transformations, such as using prefixes to indicate person feature in verb. The languages in the Bantu group use prefixes to form plurals. For some languages such as Chinese, this question is of no relevance at all. Irregular inflections are not well-handled by suffix-based transformations

and they should be handled as exception word lists, one of which is the stop-word list.

The concepts of a word *stem* and a word *root* are related but distinct: A stem is a product of the stemming process, which conflates all semantically close words, while a root is an "inner" word from which the initial word derives; i.e., it has an etymological meaning [7]. Finding a root frequently requires removal of prefixes as well, while they are not removed in stemming. Another computational problem related to stemming is morphological analysis, which aims at breaking words into smallest parts that maintain a unit meaning related to the meaning of the initial word [3].

Lemmatization. Similarly to stemming, lemmatization is a morphological transformation that changes a word into a normalized form. However, while the purpose of stemming is to conflate related morphological variations into one unifying form, and separate unrelated forms, a lemmatizer returns the corresponding lemma, which is the normalized word form as it would appear in the dictionary.

In the rest of the paper we will first discuss related work in section 2, then we will formally introduce our approach and methodology in section 3. In section 4 we describe the resource that we used as a the starting point. In section 5 we describe experiments and discuss the results, and in section 6 we conclude with a summary of the results and the main contributions, and propose tasks for future work.

The resources and program used in the paper are made publicly available and can be found at http://www.cs.dal.ca/~vlado/nlp/2007-sr.

2 Related Work

Likely the best-known and most widely used stemmer is the Porter stemmer for English [6]. The Lovins's stemmer was another known stemmer, created about the same time (a bit earlier) than the Porter stemmer. The original Porter stemmer was implemented in BCPL, the programming language that was a predecessor of C and not used so much these days. The stemmer has been re-implemented in many different languages, but the reader should be aware that many of them do not implement stemmer exactly as it was specified.[1] Both, Porter and Lovins's stemmers, are examples of algorithmic stemmers. There are two general approaches to stemming: dictionary-based and algorithmic. We discuss them in more details in the next section.

Since the appearance of the Porter stemmer a number of stemmers were implemented. For example, the Snowball framework [7] at the moment includes stemmers for 16 languages. Russian is the only Balto-Slavonic language currently implemented in the Snowball framework. There are some other implementations being publicly available. A notable site is CPAN[2], which hosts several stemmers, including a wrapper module for Snowball. For majority of languages there are no publicly available stemmers, especially for languages with sparse electronic linguistic resources. To paraphrase [7], while there are large amounts of publications discussing stemming, there are only a few descriptions that can be readily implemented in the popular efficient programming languages, such as C, Perl, Java, or similar; and there are a relatively small number of publications giving quantitative analysis and evaluations of stemmer performance.

The theoretical basis of our methodology is related to the finite state methodology described in [1] and [8].

Related to Serbian language, our search for a wider set of stemmers for any of the Slavonic languages of former Yugoslavia produced only a few results. Two stemmers could be found: A stemmer for Slovene is described in [5] and it is evaluated on an IR task, but we could not locate any available implementation. The "three new stemmers for Slovene" were mentioned at the web site of the INCO-Copernicus project[3]. There was a discussion at the Snowball list about including a Slovene stemmer into the framework[4]. There

is a publicly available Perl code for a Croatian stemmer[5]. It includes very limited documentation (several code revision comments), and only the author's user id 'dpavlin'. It seems to be a short and well-written stemmer, but it is not clear what is its coverage. It could be a toy stemmer designed only for 143 words included in the test data. The other related projects on morphological analysis that seem to have implemented lemmatizers, but not stemmers are [10] in Serbian and [9] in Croatian.

Contributions. The three main contributions of this paper are: (1) developing and making publicly available implemented stemmer for Serbian, and associated resources, (2) providing quantitative analysis of the stemmer and various steps in the process of its development, and (3) proposing and testing a general approach to building stemmers and lemmatizers for highly-inflectional languages with sparse resources. We find that the method that we used provide some interesting insights into the algorithms and data structures needed for efficient implementation of such stemmers.

3 Background

Algorithmic and Dictionary stemmers. There are two approaches to building stemmers:

1. dictionary-based approach and
2. algorithmic approach.

In the *dictionary approach*, we rely on the extensive linguistic knowledge collected in a machine-readable dictionary, while in the *algorithmic approach* we use a relative small set of rules. The algorithmic approach is generally more efficient and more compact in the sense of program size, i.e., Kolmogorov complexity. According to the Occam's razor this should lead to more generality and robustness when previously unseen words are encountered. On the other hand, the dictionary approach is more straightforward in handling exceptions and may be easier to modify and maintain. The boundary between approaches is not clear: a dictionary approach usually needs at least some rules. For example, in many highly-inflectional languages, such as Serbian, proper names are inflected and one cannot expect to have all proper names included in a dictionary. Similarly, an algorithmic stemmer will usually have lists of exceptions, which are small dictionaries. The approach that we explore here is algorithmic. On top of the known advantages of the algorithmic approach, an algorithmic approach is even more advantageous in the context of having an initial lexical resource of limited coverage with significant number of errors, i.e., noise. Overfitting the model with the resource, which would come with the dictionary-based approach, would lead to a decreased stemmer performance not only on the unseen words, but also on the training lexicon.

Stemming and Lemmatization. Under a more general term *lemmatization* we distinguish three different levels, each of which provides more sophisticated analysis

of a word:

1. **stemming**, which has been described,
2. **direct lemmatization**, or translation of a word form to a lemma, and
3. **annotated lemmatization**, or translation of a word form to a lemma annotated with the features associated with the word form.

In **direct lemmatization**, for any given word from a text, the lemmatizer returns a lemma, i.e., a base form of the word that could be found in a dictionary. The advantageous of direct lemmatization over stemming include better distinguishing between word variations, and this would lead to better applications in the IR domain. The approach could also be used in on-line dictionaries where users frequently enter variations of a word that are not directly represented in a dictionary, but their base form is. A disadvantage of direct lemmatization is that there could be a number of inherent ambiguities, since some word forms may correspond to different lemmas. Without knowing the word context, a lemmatizer can only return all of them and let the user or calling

application resolve ambiguities. In the stemming process, these ambiguities are resolved by merging different lemmas and their word forms into the same class, which has a single representative stem.

Annotated lemmatization maps, similarly to direct lemmatization, a word form into one or more lemmas, with an addition of providing a set of morphological features that are associated with this word form, such as gender, case, and number. The set of features should be such that an inverted process of morphological generation could produce the exact word form based on the provided lemma and the set of features. Annotated lemmatization can be regarded as an extended direct lemmatization, since it incrementally provides more information. Annotated lemmatization may face a higher degree of ambiguity than direct lemmatization, if there is more than one set of features that generate the same word form from the same lemma.

As an example, *stemming* translates all words in the set {boxer, boxers, boxing, boxed, . . . } to the word 'box'; *direct lemmatization* makes translations 'boxers' → 'boxer', and 'boxing' → 'box'; and *annotated lemmatization* produces translation 'boxers' → 'boxer.noun.plural'.

4 Methodology

Based on the published literature, an exclusive algorithmic approach to stemming has been suffix stripping, or, more precisely, suffix substitution. The main representative is the Porter algorithm: The algorithm groups the rules into five steps applied in succession and at most one rule can be triggered in a group. Each rule consists of a condition and a substitution of the form $s_1 \rightarrow s_2$, with the interpretation that if the condition is satisfied for a word and the word has suffix s_1, the suffix s_1 is replaced with suffix s_2. The conditions used in Porter stemmer are either such that they can be represented as suffix requirements as well, they involve minimal length of the stem in number of syllables, or it is a requirement that the stem contains a vowel. If several rules in one group are applicable, then the longest suffix match is applied. The total number of rules is 63, but if we want to represent them in a "plain-suffix" format, e.g., instead of matching a "double consonant" we actually repeat the rule with each consonant, then the number of rules is about 120. These kind of rules seem to be applicable to stemmers for other languages in the Indo-European family as well. This is the motivation behind the development of the special-purpose programming language Snowball [7].

It has been noted that the conditions on the stem length do not seem to be very important for Russian and Slovene.[6] Based on this observation, we assume that the plain suffix substitution rules should be sufficient in building our stemmer. We use only some trivial conditions on stem length, and exploring further these conditions is part of the future work. Compared to more complex rules, the plainsuffix rules are sufficient since the complex suffix rules can be expressed as a larger set of plain-suffix rules. Since the Porter stemmer is roughly equivalent to about 120 plain-suffix rules in English, which is a low-inflectional language, we expect that the number of plain-suffix rules for a highly inflectional language such as Serbian could be an order of thousands.

Lexical Morphological Resource. Our base lexical resource is a list of mappings of words w into their lemmas l. This "mapping" is not a functional relation since a word could be mapped to several lemmas. It is a general word relation: $w \xrightarrow{l} l$.

A Simple Dictionary-based Direct Lemmatizer (SDDL) could be created by using this resource. For any given word w the lemma $l(w)$ is determined by the resource relation $w \xrightarrow{l} l(w)$. Two issues are: (1) ambiguity, since one word could be associated with more than one lemma, and (2) coverage, documents regularly include words not seen before in a dictionary (*hapax legomena*).

Stemmer Derivation. The process of *deriving a stemmer* is divided into the following steps:

4.1 creation of stem-classes,
4.2 generation of stems and suffixes,
4.3 sorting suffixes by frequency, and
4.4 generation of suffix-rules.

4.1. Creation of stem-classes. If two words w_1 and w_2 have the same stem, we say that they conflate [7], and we write $w_1 \sim w_2$. The conflation relation is an equivalence relation and it partitions the set of words into the classes of equivalence. We call these classes stem-classes. We create the stem-classes from our resource by defining the conflation relation to be reflexive, symmetric, and transitive closure of the relation $\xrightarrow{l}$. Namely, for any three words w_1, w_2 and w_3:

$w_1 \sim w_1$,
$w_1 \xrightarrow{l} w_2 \Rightarrow w_1 \sim w_2 \wedge w_2 \sim w_1$, and
$w_1 \sim w_2 \wedge w_2 \sim w_3 \Rightarrow w_1 \sim w_3$.

Transitive closure is frequently implemented using matrix, but it would likely be prohibitively expensive in this case due to matrix size. An efficient way is to use the UNION-FIND data structure [2]. The result of this phase are stemclasses, i.e., groups of words that should be conflated by the stemmer. All words derived from the same lemma, according to the relation $\xrightarrow{l}$, will be conflated, but since one word may be associated with several lemmas, these lemmas will be merged into the same class as well. The quality of stem-classes needs to be verified experimentally.

4.2. Generation of stems and suffixes. In this step we need to identify what are correct stems for each word and good suffixes. An unsupervised machine learning method is applied due to sparse resources that are available. For each stem-class, we find the longest common prefix of all words in the class and define this to be the stem of each word in the class. After this, for each word in the class, the part that remains after the stem is collected as a valid suffix. We keep the count of suffixes, i.e., frequency, with an expectation that high-frequency suffixes will be good candidates for suffix-removal rules.

4.3. Sorting suffixes by frequency. Generated valid suffixes are sorted by frequency for selection of significant suffixes. While highly frequent suffixes will likely be useful, suffixes of low frequency, e.g., one, should be discarded not only to reduce the number of rules, but to produce more general rules that do not overfit coincidental word overlaps.

4.4. Generation of suffix-rules. We consider several way of generating suffix rules and experimentally evaluate each of them. Simple suffix-removal rules are considered, i.e., the rules are of the form $s \rightarrow \varepsilon$, where ε is an empty string.

4.4a Frequency-based Subsumption Stemmer. In the first approach, called frequency-based subsumption stemmer, we first select suffixes that occur with frequency higher than a given threshold. These frequent suffixes are called *valid suffixes*, and they are candidates for the suffix removal algorithm. The set of all valid suffixes is denoted by S_v. If a valid suffix s_1 is a suffix of another valid suffix s_2, than any word ending with suffix s_2, also ends with suffix s_1, so we say that suffix s_1 *subsumes* suffix s_2, and write $s_1 \supseteq s_2$, or we say that s_2 is more specific than s_1. If two valid suffixes can be removed from a word, then one subsumes the other one, and the more specific one is removed. Otherwise a more specific affix would never be applied. Additionally, this is a principle used in all Porter-style stemmers.

4.4b Greedy Subsumption Stemmer: The rules for suffix removal are selected according to suffix frequency in descending order, similar to 4.4a. The additional condition is applied by measuring stemming accuracy of the newly formed group after each rule. If the accuracy is not improved by a certain threshold, the rule is not selected.

4.4c Optimal Suffix Stemmer: Presence or absence of suffixes can be used in more complex ways that in simple suffix removal rules. For example, some rules of the Porter stemmer a rule are stated as "if a word has suffix s_1 and not s_2, then suffix s_3 is removed." The goal of the optimal suffix stemmer is to explore whether a better performance could be achieved by creating such, more complex rules, while still using only the

suffixes generated from step 2. Such optimization problem is not obviously tractable to compute, but we show that it is tractable, and implement an efficient algorithm to solve it. We say that two words w_1 and w_2 are indistinguishable by the set of valid suffixes S_v, and write $w_1 \equiv s_v w_2$, if for each suffix $s \in S_v$, s is or is not suffix of w_1 and w_2 in the same time. If two words are indistinguishable, then they are either changed or unchanged by the same suffix-removal rule in the stemming process. Additionally, the relation $\equiv s_v$ is an equivalence relation and it partitions the set of words into $|S_v| + 1$ equivalence classes (or $|S_v|$ if $\varepsilon \in S_v$). These equivalence classes are important in the context of complex suffix rules since two words in the same class cannot be separated by matching them with valid suffixes; and if two words belong to different classes then it is possible to create a boolean expression over valid-suffix matching conditions to separate the words. Hence, to find the optimal achievable accuracy with a set of suffixes S_v, we need to locally optimize each equivalence class by finding the most optimal suffix to be removed from each word in the class. This can be efficiently performed.

lemmas	47,489
word forms	675,140
word form → lemma pairs	696,263

Table 1: Lexical Resource Statistics

5 Evaluation

5.1 Lexical Morphological Resource

Our processing started from a basic lexical resource for Serbian language, which was manually created and enriched by applying derivational rules. We went through a long process of cleaning, and the resource still includes some errors. To make processing easier, the diacritic Latin letters in Serbian are transcribed into the so-called 'dual1' encoding (e.g., č=cx, ć=cy). The resource consists of word → lemma pairs, and the basic statistics is shown in Table 1. Distinct part-of-speech tags are not counted in this statistics. For example, in English, one could count work/NN (noun) and work/VB (verb) as two different lemmas, but we count them as one. This kind of ambiguity is not very frequent in Serbian.

We can also note that the number of (word form, lemma) pairs is larger than the number of word forms, but not much larger (≈ 3%). This means the Simple Dictionary-based Direct Lemmatizer (SDDL), described in the previous section could be quite accurate, since about 97% wordforms map uniquely to one lemma. It can be observed that there are about 14 different word forms per one lemma on average.

5.2 Simple Dictionary-based Direct Lemmatizer

The performance of SDDL depends on the ambiguity level of the dictionary, i.e., the resource. We define ambiguity level of a word w as ambiguity(w) = $|\{l : w \xrightarrow{l} l\}|$, i.e., the number of lemmas associated with the word. For unambiguous words, i.e., words with ambiguity level 1, the lemmatizer would give a correct answer, at least according to the resource. The distribution of ambiguity levels is given in Table 2.

Ambiguity level	Number of word forms	Percentage
6	1	0.00015 %
5	18	0.0027 %
4	156	0.023 %
3	1566	0.23 %
2	17446	2.58 %
1	655953	97.16 %

Table 2: Ambiguity level distribution of the word forms in the resource

This implies that, assuming a uniform distribution of words, we could expect an accuracy of at least 97% of the SDDL. The most ambiguous word with 6 corresponding lemmas in the resource is 'žute' (engl. *yellow*) and its lemmas are: žut, žuta, žuteti, žutiti, žutjeti.

Corpus-based Evaluation. In the above estimate we assume uniform distribution of words in text, which is not realistic. The words are typical-

ly distributed according to the Zipf's law [4]—a power distribution law, very different from uniform distribution. To make a more realistic evaluation we use a text corpus. As a representative corpus of the common contemporary language, we have chosen a collection of articles from the news magazine "Vreme" (engl. "Time") from the period of five years 2001–5. The corpus size is 44MB and it consists of 6.6 million words.

The first use of the corpus is to evaluate the coverage of our resource, i.e., the percentage of corpus words that are included in the resource. After the first run we found that only 56% of the words in the corpus were found in the resource with case-sensitive matching. Besides names, the words are capitalized at the beginning of a sentence and in titles so we found that case-insensitive coverage is 61%. An examination of unrecognized words reveals that about 35% of them are proper names. Another significant unrecognized group are conjunctions and prepositions, which are very frequent and happened not be included in the resource. The proper names are a group that is hard to predict so we cannot assume that they would be covered by a better resource. However, the names follow similar morphological patterns as common nouns, which is an additional evidence that an algorithmic approach would be advantageous, and that it would generalize better. Within these 61%, 50% words in the corpus (49.79% more precisely) are unambiguous in the resource (ambiguity(w)=1). This is about 50/61 ≈ 82% of recognized words, which is a less optimistic evaluation than the one obtained for uniform distribution. This implies that SDDL would have accuracy of at least 50%, and likely not much higher than 61%, assuming that some simple strategy for unknown words is used.

1	457	(1,1%)	8	3946	(9,5%)
4	1436	(3,4%)	9	1494	(3,6%)
5	1703	(4,1%)	12	3962	(9,6%)
6	1320	(3,2%)	13	2433	(5,8%)
7	11942	(28,7%)	29	547	(1,3%)
31	2633	(6,3%)			
32	1481	(3,6%)			
33	2872	(6,9%)			
34	446	(1,1%)			
37	632	(1,5%)			

Table 3: Distribution of Stem Class Sizes, higher than 1%

Before proceeding with evaluation of our stemmer-generating method, the lexical resource is improved in the following way. The ten most frequent word that are not covered by the resource are: 'i', 'u', 'na', 'za', 'su', 'a', 'ne', 'od', 'sa', and 'o', which are very frequent functional words and are omitted from the resource simply because those part-of-speech tags were not included (conjunctions: 'i', and 'a'; prepositions: 'u', 'na', 'za', 'od', 'sa', and 'o'; auxiliary verb: 'su', and adverb 'ne'). After manually adding 200 more word-lemma pairs, the coverage increased to 85% with the 73% unambiguous words from the resource. This is a usable accuracy, but the limitations are that it requires almost 700,000 wordlemma pairs, has no generalization capability, and likely contains some errors evident in the resource.

5.3 Stemmer Evaluation

Step 4.1: Creation of stem-classes. After transitive closure, 677,868 unique words from the resource are distributed into 41,681 classes, giving on average 16.3 words per class. The number of words per class varies between 1 and 307 words per class. The classes with more than 80 words are very sparse. For example, two largest stem classes have 307 and 283 words. After examining them, we see that they are created by incorrectly merging two or more proper stem classes, likely due to some erroneous word-lemma pairs. The most frequent stem-class size is 7, which is 29% of the classes. The distribution of class sizes with more than 1% of all stem classes is shown in Table 3.

Step 4.2: Generation of stems and suffixes. After producing stems and suffixes in this step, any empty stems obtained are indicators of in-

correct stemclasses. In a number of cases it was caused by the prefix ‘naj-’, which is used in superlative inflections of adjectives and adverbs. As we noted before, the prefixbased derivations should not be treated in stemming. As an illustration, if we are searching a document collection for the highest mountain peak in the world, we are likely interested in ‘highest’ precisely and not ‘high’ peaks or comparison ‘higher peak’, even though these are conflated in English. Removal of prefix ‘naj-’ would cause additional errors since it appears as a prefix in non-superlative words, such as ‘najamnik’ and ‘najahati’. The issue could be resolved by removing the prefix ‘naj-’ only when matched with the corresponding superlative suffixes ‘-ija’, ‘-iji’, and similar. We address this problem by separating superlative and nonsuperlative stem-classes.

Another source of empty stems are irregular inflections, such as the plural noun ‘ljudi’ of ‘čovek’ or ‘čovjek’ and auxiliary verb form ‘ćeš’ of ‘biti’. Both of these could be handled by an exception list, but we decide to separate them in different stem-classes. We assume that an IR user would not expect a search term to be expanded in this way (e.g., for ‘ljudi’), or auxiliary verbs would be removed as stop-words anyway.

The stems of length 1 are suspects of incorrect classes, but they were not systematically removed. One example is the word ‘beže’, which is a present tense form of verb ‘bežati’ (engl. to escape), and the vocative case of the noun ‘beg’ (engl. bey), which leads to an erroneous merge of two, otherwise correct, stem classes. In the first run 650 words produced an empty stem. For all of them, we manually fixed the original resource, which caused break-up of corresponding stem-classes and production of non-empty stems. Short stems (e.g., length 1) are also frequently created by incorrectly merged stem-classes, but we hypothesized that it may not be necessary to fix them in this experiment, since the later methods use the most frequent suffixes, which should have high reliability. The maximal common prefix method used to generate stems created some additional overlap among stem-classes, effectively merging them: 39,289 stems are created, 1,823 (4.6%) of those were ambiguous in the sense that they were associated with more than one stem-class. Only 253 had ambiguity level of three or more, with the stem ambiguity level decreasing quickly when sorted in descending order. The most ambiguous stems are given in the list below.

43 ist	18 post	16 samo	14 ekst
26 rast	18 sat	15 ust	13 zast

12 ost	7 pos	7 nast
12 konst	7 podst	7 nas

These highly ambiguous stems do not maintain meaning of the word, and an improvement method for this step is a part of our future work.

Step 4.3: Sorting suffixes by frequency. In this step 18,274 suffixes were generated, and the top of the sorted list of generated suffixes with frequency is given in the table below.

24833 -e	6495 -oj
22874 -u	6475 -omu
22389 -i	6121 -oga
22184 -a	6118 -og
19475 -om	5929 -ti
17756 -o	5775 -t
16190 ‘’ (empty)	4412 -h
	4399 -m
8996 -im	4303 -cyecx
8281 -ama	4289 -cyu
8101 -ih	4273 -le
7573 -te	4272 -la
7472 -ima	4268 -li
6821 -mo	4252 -cye

All of these suffixes have linguistic interpretation.

Step 4.4: Generation of Suffix Rules. A direct implementation of evaluation of different suffix-rule generation approaches lead to a very slow evaluation. An efficient implementation with a compact trie (historically also known as

a Patricia trie) with reversed strings significantly reduced running time, from 5-6 hours for initial experiments to about 5-10 minutes.

(4.4a) Frequency-based Subsumption Stemmer. For the frequency-based subsumption stemmer, we started with an empty set of valid suffixes and incrementally added one rule at the time in order determined by rule frequency. After each stem the stemming accuracy according to our generated stems is measured. It started with 2.4% with $S_V = \varnothing$ and gradually increased to 56.3% with 98 suffix rules, and then gradually decreased to 14.2% when all 17,839 suffixes were included.

(4.4b) Greedy Subsumption Stemmer. In the greedy approach, we add rules in the same order as 4.4a, but before and after adding each suffix we measure accuracies A_1 and A_2 in the number of correct stems. The rule is accepted if $A_2 - A_1 > \Theta$, where Θ is a given parameter, i.e., the suffix is accepted only if it improves accuracy for more than a given threshold. For example, if $\Theta = 0$, then a suffix is accepted only if it does not decrease the overall accuracy; if $\Theta = 1$, then the number of correct stems must increase by 1 at least, and so on. The higher Θ parameter is, we expect the better generalization by choosing less rules that have higher quality, but they may decrease performance. The results are shown in the following table.

Θ	Valid Suffixes	Accuracy
0	9849	74,15
1	8633	74,16
2	3367	73,38
3	1901	72,95
4	1557	72,83
5	1262	72,66
6	1124	72,56
7	1002	72,46
8	933	72,39
9	878	72,32
10	831	72,26
15	673	71,99
20	592	71,78
25	497	71,48
30	453	71,30
35	423	71,16
40	410	71,09
45	380	70,90
50	360	70,76
60	347	70.65
70	319	70.39
80	310	70.29
90	298	70.14
100	294	70.23
150	273	69.87
200	230	68.80
250	218	68.43
300	202	67.77
350	188	67.08
400	180	66.65
450	179	66.59
500	175	66.31
600	131	62.74
700	121	61.82
800	114	61.03
900	87	57.76
1000	85	57.48

Two interesting observations that can be made are that the accuracy is much higher than with the previous approach, and the accuracy drops very initially slowly while the number of rules drops quickly, which is another very encouraging observation. At the $\Theta = 7$ we obtain 1002 suffix rules with accuracy only about 1.7% less than the best one. This fits well with our prediction that a stemmer for Serbian language would need about 1000 suffix rules.

(4.4c) Optimal Suffix Stemmer. The accuracy of the optimal suffix stemmer is 81.83%. This is the upper bound of what can be achieved with the obtained set of valid suffixes and the corresponding suffix removal rules, when evaluated on the produced set of stems. We can see that the greedy approach is not that much lower, especially considering the argument that our goal should not be to match the optimal accuracy since

we would overfit the initial flaws of the lexical resources and some incorrect stems produced in previously described process.

5.4 Unbiased Evaluation

To evaluate the stemmers in an unbiased way we use the news corpus, run the stemmers on a sample set of words of the corpus and manually judge produced stems. We choose to evaluate two stemmers: 4.4c (Optimal Suffix Stemmer) and the greedy stemmer (4.4b) with the parameter $\Theta = 7$ and 1000 generated rules. An interactive program reads the words from the corpus in sequence and runs both stemmers on them. Since we are more interested in words not included in the resource, the words that exist in the resource and for which the stemmers produce the same stem are ignored. Otherwise, the stems are produced for manual evaluation with four decisions: only greedy correct, only optimal correct, both correct, both incorrect, and ignore. The option 'ignore' is used to exclude some functional words which are obvious stop-words and some English words appearing in the corpus. A stem is judged to be correct if the original meaning can be clearly predicted from the stem (no over-stemming), and it seems that the stem covers all morphological variations of the lemma (no under-stemming). After evaluating 1000 non-ignored words from the corpus (with possible repetitions) the result was: 127 words with greedy correct only, 90 optimal correct only, 663 both correct, and 120 none correct. These results confirm two of our hypotheses: (1) The stemmers produced in the process seem to be usable in IR (greedy accuracy 79% and Optimal accuracy 75%); and (2) the greedy approach produces not only as good results as the optimal stemmer, but generalizes even better (better accuracy) with only 1000 rules.

6 Conclusion and Future Work

In summary, we described and evaluated a largely automatic general approach to generating stemmers for highly-inflectional languages with only a few resources. Some limitations in the process are discovered as well as opportunities for further improvement. The final evaluation has shown 79% accuracy on real data for the Greedy stemmer, which is even a bit higher than the accuracy obtained on the training data, showing a very good generalization capability. Some directions for future work are: (1) evaluation on more data, (2) inclusion of suffix substitution rules instead of just suffix removal rules, and (3) inclusion of stem length parameter. With suffix substitution rules, the method can be directly applied to the lemmatizer generation.

[1]The current official web site for the Porter's stemmer is http://tartarus.org/~martin/PorterStemmer/, and it is the authoritative source for the implementations of the original stemmer. A quick test to check authenticity of a Porter stemmer implementation is the word 'agreement'—it is not changed in the original Porter stemmer, while some incorrect implementations change it.

[2]CPAN—Comprehensive Perl Archive Network, http://cpan.org/, is an open-source repository for Perl packages.

[3]http://www.mf.uni-lj.si/ds/new-stemmers.html

[4]http://snowball.tartarus.org/archives/snowball-discuss/0722.html

[5]http://svn.rot13.org/index.cgi/stem-hr

[6]Source: Snowball mailing list.

References

[1] K. Beesley and L. Karttunent. Finite State Morphology. CSLI, 2003.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction fo Algorithms. The MIT Press, 2nd edition, 2002.

[3] Chris Jordan, John Healy, and Vlado Kešelj. Swordfish: An unsupervised ngram based approach to morphological analysis. In SIGIR'06: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 657–658, Seattle, Washington, USA, August 2006. ACM Press.

[4] Daniel Jurafsky and James H. Martin. Speech and Language Processing. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2000.

[5] Mirko Popović and Peter Willett. The effectiveness of stemming for natural language access to Slovene textual data. Journal of the American Society for Information Science, 43(5):384–390, 1992.

[6] Martin F. Porter. An algorithm for suffix stripping. Program, 14(3):130–137, July 1980.

[7] Martin F. Porter. Snowball: A language for stemming algorithms. Published on WWW, October 2001. Last access in April 2007.

[8] S. Sheremetyeva, W. Jin, and S. Nirenburg. Rapid deployment morphology. Machine Translation, 13(4):239–268, 1998.

[9] Marko Tadić. Hrvatski lematizacijski poslužitelj. Published on WWW, 2005. Last access in April 2007.

[10] Duško Vitas and Cvetana Krstev. Derivational morphology in an e-dictionary of Serbian. In Proceedings of 2nd Language & Technology Conference, pages 139–143, Poznan, Poland, April 21–23 2005.