Improving a Spelling Checker for Afrikaans

Menno van Zaanen and Gerhard van Huyssteen

University of Amsterdam, University of Tilburg
Potchefstroom University for Christian Higher Education

Abstract

In this paper we describe the development of an improved spelling checker for Afrikaans. We compare two currently available spelling checkers and discuss their shortcomings. The existing applications are restricted in their suggestion capabilities, as well as their precision and recall, mainly because they cannot treat morphologically complex words correctly. Here, we will mainly focus on improvements in precision and recall.

The general architecture of the existing spelling checker is discussed and several improvements are implemented. We describe an improved lookup phase and a newly added morphological analysis phase. The morphological analysis poses some problems which are also treated. Finally, some remaining problems are mentioned.

1 Introduction

Although Natural Language Processing and Computational Linguistics have been of extensive research interest for many years in the USA, Europe, and other parts of the world, these fields are by and large unexplored territory in South Africa. Except for some sporadic research efforts by individuals, no Human Language Technology (HLT) projects existed for many years.

Only recently an awareness of the importance of HLT for the South African community has started to bloom on both academic and governmental level, to such an extent that, in 2002, the minister of the Department of Arts and Culture convened an advisory committee to steer the development of HLT in South Africa. Since 2001, the Potchefstroom University prioritized these fields as strategically important research topics, and established a research focus area called "Language and Technology".

The university invested extensively in the enhancement of personnel and other resources to develop human language technologies for some of the South African languages, including the establishment of the first complete graduate study program in Computational Linguistics in South Africa, the setting up of a dedicated language technology laboratory, and the acquisition of text and speech corpora for Afrikaans, South African English, and Setswana.

The first major HLT project to be funded by the university was a project to develop a spelling checker for Afrikaans, based on morphological analysis. As the IT department at the Potchefstroom University had already developed a spelling checker for Afrikaans (the *PUK/Microsoft Spellchecker*, here called *PUKspell*), the development of an enhanced spelling checker was considered an "easy" project to get hands-on experience in the fields of Natural Language Processing and Computational Linguistics, and at the same time to develop reusable enabling technologies for Afrikaans. Over and above these educational and research purposes, the project was also deemed to be commercially viable, with the opportunity to support further research from the earnings of the spelling checker.

In this article we report on the progress of this project. We will first discuss the evaluation of the existing spelling checker, identifying the areas for improvement. Then we will give an overview of the architecture of the new spelling checker, indicating how it improves on the shortcomings of the existing spelling checker. To conclude, we discuss some remaining problems and possible solutions.

2 Comparison between Existing Spelling Checkers

In order to improve the existing *PUKspell*, we need to know what should be improved. We therefore compare two available spelling checkers for Afrikaans: *PUKspell* and another commercial spelling checker which we will call *Bspell*[1] (van Huyssteen 2002).

The evaluation of the spelling checkers is done according to a set of metrics. The next section will describe which metrics will be used and also discuss some of the problems that remain when evaluating spelling checkers. It concludes with a discussion of the results of the evaluation phase.

2.1 Evaluation Metrics

The evaluation of the spelling checkers that will be performed in this article can be divided into two parts, viz. user-friendliness (e.g. ease of installation, completeness and readability of the manual, ease of use, etc.), and performance (i.e. recall, precision, and suggestion adequacy).

An objective measurement of user-friendliness is difficult. What value should be assigned to, for example, the ease of installation? In the evaluation performed here, we found that both spelling checkers are comparable in user-friendliness. Both install without any problems, they have comparable manuals, and the user-interface is the same (it is the Microsoft Word interface). Based on this, no preference for one of the two spelling checkers could be made and no deficiency that should be improved could be found. The remaining part of the evaluation in this article will therefore focus on other metrics.

The metrics that are used to evaluate the performance of the spelling checkers (i.e. recall, precision, and suggestion adequacy), describe more directly measurable aspects. Table 1 gives an overview of the metrics that will be used here. Some of these terms and concepts might need some explaining

We will follow Paggio and Music (1998) in their definitions of the metrics. *Valid words* are words that are part of the language, or which are sanctioned by the language system, in contrast to *invalid words*, which are *not* part of the lexicon or language system. When the spelling checker claims a word is invalid, it is *flagging* that word, while *accepting* a word means treating it as valid. Accordingly, a *flag* is an indication that a word has been tagged as invalid (regardless if the word really

[1] Due to South African law, we are not allowed to use the name of the commercial spelling checker directly.

Table 1: Evaluation metrics

Metric	*Method of measurement*
Lexical recall	$\frac{\text{\# valid words accepted}}{\text{\# valid words}}$
Error recall	$\frac{\text{\# invalid words flagged}}{\text{\# invalid words}}$
Precision	$\frac{\text{\# correctly flagged invalid words}}{\text{\# words flagged}}$
Suggestion adequacy	$\frac{\text{\# correct suggestions for flagged, invalid words}}{\text{total \# of flagged, invalid words}}$

was invalid or not). *Suggestions* are alternative valid words that are offered to the user to replace a flagged word with.

The goal of a spelling checker is to flag all invalid words and to accept all valid words. If this is the case, all invalid words are correctly flagged.

With this information, it becomes intuitively clear what the metrics actually measure. Lexical recall indicates the percentage of valid words correctly accepted by the spelling checker, error recall gives the percentage of invalid words correctly flagged and precision gives the percentage of correct flags (correctly found invalid words) over all flags by the spelling checker.

Finally, we could measure how good, once an invalid word is found, the suggestions of the spelling checker are. Ideally, the spelling checker should only suggest the preferred correction. However, in practice it is sometimes unclear what the preferred correction really is. For example, what should be the correction of the invalid word: "learnd"? It could be "learn", "learned", or "learns", among possible others. Similarly, in Afrikaans: "mek" could be corrected to "melk" (milk), "met" (with), or "nek" (neck).

To calculate the suggestion adequacy, we have actually counted scores and divided them by the total number of flagged, invalid words. The score that we use in our evaluation for a single suggestion can be found in table 2. The score of an entire text is computed by adding the scores of all suggestions of correctly flagged, invalid words. The suggestion adequacy is the percentage of score over the flagged invalid words.

Of course, using scores instead of simple counts does not solve the problem of knowing what the actual preferred word is, but at least it gives an indication of how good spelling checkers are in suggesting corrections.

2.2 Problems of Evaluation

Evaluating spelling checkers is not as simple as it might seem, as several problems can occur and different variables should be taken into account. The most important

Table 2: Scoring of suggestions

Score	*Suggestion*
0.0	No suggestion or bad suggestions
0.5	Good suggestion anywhere in the list of suggestions
1.0	Good suggestion within the first three suggestions

variable seems to be the texts used to evaluate the effectiveness of the spelling checkers.

Test texts should of course contain valid and invalid words. Counts of found valid and invalid words are obtained by applying the spelling checkers to these texts. These counts are then used to determine the performance effectiveness of the spelling checkers, by using the formulas in table 1.

If the recall and precision results of the spelling checkers are to be compared, then the spelling checkers should of course be applied to the same texts. When comparing spelling checkers of different languages, this is not possible, and the only solution may be to develop some measure of difficulty of a text.

The composition of the test texts is of utmost importance. First of all, it should be known which words are valid and which are invalid (if, for example, one does not know which words are invalid, it is impossible to figure out if a spelling checker found all invalid words). To compose such a text, one would ideally make use of corpus material, including both valid and invalid words. However, to evaluate specific features of a spelling checker (e.g. its capability to handle productive compounding, proper names, abbreviations, or other morphologically complex words) one could also systematically compose a text to include data for this purpose. This information about the construction of the test texts should be taken into account when computing the recall and precision of the spelling checkers.

In this evaluation only one small, constructed text is used, since the focus is on finding possible improvements. The main problem with using a larger text is to label errors in the text (i.e. annotating) before applying the spelling checkers to it.

Another evaluation problem is to investigate whether spelling checkers can handle context-sensitive spelling checking. This can be relatively simple, for example, by investigating whether a spelling checker can take care of multi-word units, such as fixed expressions and idioms. It can also be more complex, by looking at the effectiveness of the spelling checker to valid words in context such as: "then" and "than" (in English), or homonyms such as "vlei" (flatter) and "vly" (lay down) in Afrikaans.

It should be noted that the evaluation described in this article is not meant to show how good the spelling checkers are or which one is better, but merely to investigate which improvements are wanted.

Table 3: Comparison of numerical results of the spelling checkers

Measure	*PUKspell*		*Bspell*	
Lexical Recall	95%	(1196/1255)	98%	(1225/1255)
Error Recall	84%	(69/82)	83%	(68/82)
Precision	54%	(69/128)	68%	(68/98)
Suggestion	60%	(41.5/69)	86%	(58.5/68)

2.3 Results

This section will describe some results of the two evaluated spelling checkers, *PUKspell* and *Bspell*. The results of applying the spell checkers to the test text described above, can be found in table 3. This table contains percentages as computed by the metrics of table 1 and, additionally, the plain counts are given between brackets.

In the current evaluation, we have applied the spelling checkers to a text containing 1337 different words, of which 1255 are valid and 82 invalid. For this evaluation task, the text has been constructed based on a small Afrikaans corpus of e-mail messages, and contains simple words, morphologically complex words, compounds, proper names, abbreviations, and foreign words that are valid in Afrikaans. Invalid words from the corpus have also been included. Evaluation of context-sensitive spelling checking is *not* performed, since both spelling checkers do not show any context-sensitivity.

The results show two interesting differences between the two spelling checkers. *PUKspell* performs overall worse than *Bspell* on this text. Most notably, it rejects too many valid words, hence the lower lexical recall and precision. This can possibly be explained by the fact that the lexicon of *Bspell* is 31% larger than that of *PUKspell* (respectively approximately 180,000 and 235,000 words). Secondly, when looking at the rejected words, we found that both spelling checkers have difficulty handling compounds. Thirdly, *PUKspell*'s suggestion adequacy is lower than that of *Bspell*.

When looking at the words that are rejected, we found that both spelling checkers have difficulty handling compounds. Since Afrikaans is an agglutinative language, compound formation is productive.

The results indicate three means of improving the existing spelling checker:

1. Increase the number of valid words the spelling checker will accept. This can be done by increasing the lexicon or adding morphological information.

2. Extend the spelling checker with compound analysis. Related to this improvement is the handling of morphologically complex (non-compound) words.

3. Improve the suggestion module. This includes generating better sugges-

tions, but also an improved ordering of the found suggestions.

To get a bit of an idea of how well these spelling checkers work, we mention results of the SCARRIE project, a spelling checker for Scandinavian languages. Paggio and Music (1998) mention a lexical recall of 96% (2734/2841), error recall of 20% (20/99) and a precision of 16% (20/127). Aduriz et al. (1997) mention a suggestion adequacy of 86%. These results are of course not directly comparable against the results in table 3, because other languages and other texts are used.

In this article, we will concentrate on the first two improvements. The improvements on the suggestion module are still under development and will be touched on briefly in section 4.3.

3 Comparing the Existing and Improved *PUKspell* Versions

To be able to understand how the improvements are incorporated in the existing spelling checker, we will first give an overview of the overall architecture of the existing spelling checker. We will then discuss the changes that are made in the architecture to implement the improvements.

3.1 General Architecture of Existing *PUKspell*

The currently available version of *PUKspell* has an architecture, as globally depicted in figure 1. Microsoft Word delivers an arbitrary piece of text to be checked. This piece of text is tokenized (a simple algorithm is used here, that segments the text using spaces as word boundaries). Each token is then given to the lexicon lookup module, which tries to find the token in the lexicon. This lexicon consists of roughly 180,000 words. If it is not found, the token is forwarded to the suggestion module, which searches for words in the lexicon that are relatively close to the token. These suggestions (if found) are given to the user.

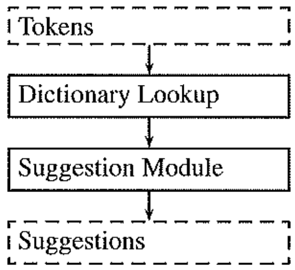

Figure 1: Architecture of the existing version of *PUKspell*

Two remarks need to be made. First, Microsoft Word fills an array with characters and then calls the spelling checker. The text in the array is not always a single word, a sentence, or entire paragraph. It can contain one or more words, possibly including sentence boundaries, but this can be different each time the spelling checker is called. The success of the spelling checker therefore depends on effective tokenization.

Second, the success of the spelling checker is directly related to the completeness of the lexicon. If not enough words are contained in the lexicon, too many valid words are flagged as invalid. This means that in theory all morphologically complex words *and* compounds should be added to the lexicon.

There has been a discussion on how large a spelling lexicon should be (Vosse 1994, p. 49), with two opposing views. One view claims that the lexicon size should be kept small. The reason for this is that increasing the lexicon size results in more misspelled words to be accepted. This happens in particular when infrequent valid words that differ only in one letter with another valid word are added to the lexicon. The other view concentrates on the opposite idea that the lexicon should be large, because this will reduce the number of valid words that are flagged as invalid.

3.2 Improvements

Now that the architecture of the existing spelling checker is known, we can describe the improvements that are implemented. We will follow a token, step-by-step, going through each phase in the spelling checker. After discussing all improvements, we will give an overview of the improved architecture.

3.2.1 Error Detection

The improved spelling checker contains two new modules that handle error detection. These modules are added to speed up the lookup process, such that the analysis of more complex words later on in the process does not have to be extremely efficient (because not all invalid words reach that part of the process).

The first error detection module is the *error lookup* module. It is a lookup module that searches a lexicon containing *invalid* words. Together with the invalid words, their best suggestions (one or more) are stored. When a word is found in this phase, the spelling checker can automatically give these suggestions to the user. An example word in this lexicon is: "inteligent" with "intelligent" as its suggestion.

A second way of filtering out obviously invalid words is done by using n-gram analysis. We have extracted n-grams (3, 4, and 5-grams of *graphemes*) from a corpus of plain text (of roughly 150,000 words). Tokens that contain n-grams that were not found in the corpus are considered invalid. This flags clearly invalid words such as "xyyz", but it also finds words, such as "maatxkappy" (society), where for example an *x* is (incorrectly) substituted for an *s*, because they are next to each other on the keyboard.

3.2.2 Extended Lexicon

The evaluation of the two spelling checkers showed us that too many valid words are not recognized. In the improved version of the spelling checker, we tackle this problem in two ways. In this section we describe a direct extension and in the next section a more indirect extension of the lexicon.

The easiest way to get the spelling checker to accept more valid words is simply to increase the size of the lexicon. In the next release of the spelling checker, the lexicon will be 39% larger; containing some 250,000 words. The lexicon size is then on par with that of *Bspell*.

However, increasing the lexicon does not entirely solve the problem. Even though more valid words are accepted, there is a structural difficulty with the generative lexical power of agglutinative languages. The next section will discuss a partial solution of this problem.

3.2.3 Morphological Analysis

Afrikaans has many morphological rules that can be used to create new words (Suid-Afrikaanse Akademie vir Wetenskap en Kuns 2002). Of course, one can add all words that can possibly be generated to the lexicon; the spelling checker will then accept all these words, but this will increase the size of the lexicon exponentially. However, if this is done, the structural information that is inherently incorporated in morphologically complex words (and that can be described by simple rules) is discarded.

The approach taken in the improved spelling checker is to incorporate morphological information in the form of morphological rules. This will keep the size of the actual lexicon small, but by applying the rules, morphologically complex words (that are not directly in the lexicon) are still considered valid. In other words, adding morphological rules has two advantages. The size of the lexicon is reduced, while on the other hand, the number of accepted words grows.

Figure 2 depicts the relationships between the size of the lexicon of the spelling checkers and the accepted words. The middle ellipse delimits the valid words accepted by the existing spelling checker. The smallest ellipse is the size of the original lexicon after removing all morphologically complex words. When morphological rules are added to the lexicon, all words in the largest ellipse are accepted.

Morphological rules normally describe how a morphologically complex word can be generated from a simpler word. This means that the spelling checker has to try to recognize the morphologically complex word by applying rules to words in the lexicon. This is of course infeasible. Instead, we chose to reverse the rules and reduce the morphologically complex tokens into simpler words using the rules. The heart of this phase is a Porter stemmer (Porter 1980, Kraaij and Pohlmann 1994) for Afrikaans. This stemmer has been developed at the Potchefstroom University (van Huyssteen and van Zaanen 2003).

Note that usually a stemmer is used in the context of information extraction

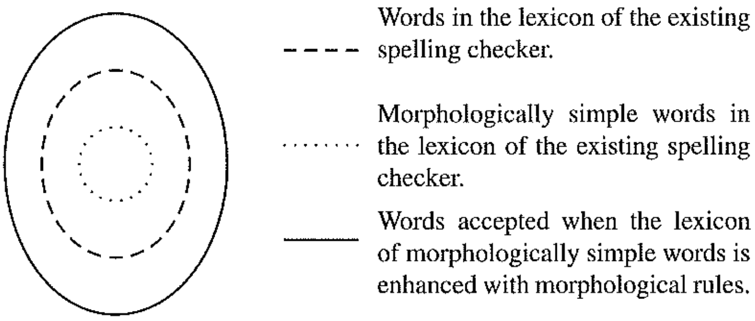
Words in the lexicon of the existing spelling checker.
Morphologically simple words in the lexicon of the existing spelling checker.
Words accepted when the lexicon of morphologically simple words is enhanced with morphological rules.

Figure 2: Morphological rules increase the number accepted valid words

and retrieval (IE/IR), whereas here it is used to enhance the lexicon. In the IE/IR context, words that are in the same semantic class should reduce to the same stem. This is not the case here, where valid words should simply remain valid (and invalid words must also remain invalid).

The main difficulty with using a stemmer is that it can easily over-generate. For example, Combrink (1990) identifies the *a-* as a possible prefix in Afrikaans, in words such as "asosiaal" (asocial) and "atonaal" (atonal). However, care should be taken not to remove this prefix in words like "Afrikaans" and "artistiek" (artistic), where the *a-* is clearly not a prefix.

The problem regarding over-generation is currently solved by using morphological rules only if they clearly do not introduce problems. This means that a prefix such as *a-* is not part of the rules, but *-lik* (as in "aansienlik" (considerable)) is. Note that the rules should *not* be able to transform an invalid word to a valid word. The other way around sometimes happens, but that does not matter. Applying the *-lik* rule to a valid word such as "aanblik" (sight) results in "aanb", which is not a valid word. At the moment, we have collected 232 morphological rules for the purpose of reduction of morphologically complex words.

Note that in the improved spelling checker, the lexicon containing morphologically simple words is expanded with respect to the lexicon of the existing spelling checker, which results in an even larger lexicon.

3.3 Architecture of Improved *PUKspell*

This section will discuss the architecture of the improved spelling checker. Figure 3 gives a graphical overview of the architecture. Each of the modules will be described in some detail next.

Similarly, as in the existing spelling checker, the improved spelling checker receives tokens to be validated. These tokens are send to the improved lookup module.

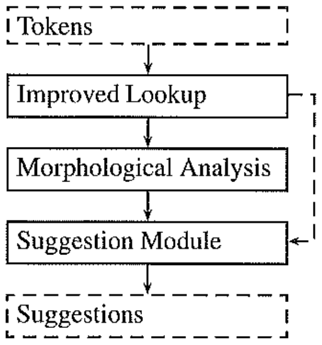

Figure 3: Architecture of the improved version of *PUKspell*

The lookup module consists of several steps. First, the error detection is done. It consists of two steps, lookup in the invalid word lexicon and n-gram analysis. If the token is found to be invalid, it is send to the suggestion module (shown as a dashed arrow in the figure).

The lookup module (invalid word lexicon and n-gram analysis) can only decide that a token is invalid. If the token is not recognized in this module, it is handed to the morphological analysis module. This module consists of several lookup steps (similar to the lookup module of the existing spelling checker).

Using the stemmer, the morphological analysis module transforms a token step-by-step into simpler words. Each time, the word is checked against the lexicon. If at one point a valid word is found, the token is considered valid.

If the token is still not found, it is given to the suggestion module. Currently, the suggestion module is a simple extension of that of the existing spelling checker. However, as mentioned earlier in this article, improving the suggestion module is future work. We discuss some of the problems in section 4.3. If suggestions are found then they are returned to the user.

4 Remaining Problems

The new version of the spelling checker for Afrikaans will have many improvements over the existing version. However, some problems still remain. This section will focus on these remaining problems. First, we will look at the evaluation, followed by handling compounds and, finally, something will be said about the suggestion module.

4.1 Evaluation

Even though the overall architecture of the spelling checker is described in this article, currently, the implementation of the spelling checker is only in the design phase. The improved spelling checker will be completely re-designed and re-implemented. Once the new implementation is finished, it will be evaluated.

The evaluation will be focusing on specific types of words, such as morphologically complex words, compounds, and foreign words that are acceptable in Afrikaans (and words that are not). This evaluation will show whether the improvements described in this article really lead to a better spelling checker. This will be described in a future publication.

4.2 Compounds

In section 3.2.3, we explained that adding morphological information to the spelling checker increases the number of valid words that can be handled. Morphological rules are implemented using a Porter stemmer. The stemmer can strip affixes, but is unable to handle compounds.

Finding the separate elements in compounds is a difficult problem. There are several reasons for this. For example:

- A compound can consist of several words, e.g. "strand-hand-doek-stof" (beach towel cloth), so trying to combine only two words from a lexicon is not enough.
- Words can have infixes, such as *s* in "bruilof-s-gas" (wedding guest), *e* in "student-e-nommer" (student number), *en* in "wa-en-huis" (garage). But also *ns*, *ens*, *tes*, and *er* are possible infixes in a compound.
- Sometimes, doubling of consonants occurs, for example in "den-n-e-boom" (pine tree), where the "n" is doubled.
- A compound can also be constructed using morphologically complex words, for example, "vergadering-s-prosedure" (meeting procedure) or "verantwoordelikheid-sin" (sense of responsibility).

At the moment, we use a longest prefix and suffix string matching to find elements in a compound (van Huyssteen and van Zaanen 2003). The idea is to find the longest valid word that is a pre- or suffix of the compound that can still be detected. This process must sometimes be repeated when compounds consist of more than two words. (This method can be improved when some more linguistic knowledge is added, as described in (Vandeghinste 2002).)

The approach sketched here is simple and seems to work often. It does not work well with morphologically complex words, since these are handled by a combination of the lexicon and the stemmer. In the end, a complex interplay between finding the word boundary and using the stemmer might solve this problem, when taking added letters (transition elements) and the doubling of consonants into account.

A more interesting approach would be to use machine learning. Using a lexicon of structured compounds (delimiters between the elements of compounds), it is possible to train a classifier. New words are then handed to the classifier, which will insert the word breaks.

There has already been some work on compound splitting (Koehn and Knight 2003), but most approaches make extensive use of structured training data. Unfortunately, this data is not (yet) available for Afrikaans, making these approaches less applicable. Even though, currently, we are applying the longest prefix and suffix string matching to compound splitting, when we have accumulated enough training material, we will venture into a machine learning approach.

Another problem, that is strictly related to handling compounds in a spelling checker, is that when a compound contains an error, it might not be so easy to find the compound boundaries. Previous work has concentrated on finding word boundaries within valid compounds. Spelling errors in a compound can result in incorrect word boundaries. Once incorrect word boundaries are found, it might prove difficult in the end to correct the error in the compound.

4.3 Suggestions

Generating suggestions for invalid words already has quite a long history (Wagner and Fischer 1974, Lu and Fu 1977, Kernighan et al. 1990). This research focuses on finding the most similar words with respect to the invalid word. The underlying idea is that the invalid word resulted from an edit operation (for example, insertion, deletion, substitution, or transposition) and thus applying the reverse of these operations to the invalid word should result, among others, in the meant valid word.

This approach can still be taken in the improved spelling checker. However, the morphological rules that have been added to the spelling checker introduce problems. Whereas the standard techniques search the lexicon, the improved spelling checker accepts words that can be in the lexicon *or* that can be analyzed by applying morphological rules on the words in the lexicon.

The idea to solve this problem is to try and stem the word using the morphological rules and then reverse this process to reach possible suggestions. However, this is by no means an easy problem. It is already known that the word is invalid (otherwise it would never reach the suggestion module). This means it is possible that an error occurred in one of the affixes the stemmer is trying to remove. For example, assume that the word "snelheid" (speed), with suffix *-heid* is misspelled as "snelhed" (deletion of the "i"). In this case, the suffix cannot be removed and the rest of the word cannot be found.

The suggestion module should take into account that spelling errors can occur in the affixes used by the lexicon. This increases the complexity of the algorithm.

To make things worse, this problem becomes even more difficult when compounds are handled. Firstly, finding word boundaries of the compound may prove difficult, as discussed in the previous section. Secondly, correcting invalid compounds, even when the correct word boundaries are known, is hard. The algorithm must correct multiple words instead of a single word. Thirdly, there are no clear or easily implementable rules on when to insert combining letters (*-s-*, *-en-*, etc.).

5 Conclusion

This article describes the development of an improved spelling checker for Afrikaans. Based on the evaluation of an existing spelling checker, various improvements are suggested. The main outcome of the comparison showed that precision and recall should be improved, as well as the suggestion module.

By adding morphological information, we try to enhance the generative power of the lexicon and subsequently recognize more words. To reduce the number of accepted invalid words, n-gram analysis and an error lexicon is added. This will ultimately lead to an increase in precision and recall.

Some problems remain, such as a good compound analysis phase and an improved suggestion module. Finally, once the improved spelling checker is implemented, it should be thoroughly evaluated.

On a more general note, we need to mention that the research is performed at the Potchefstroom University in South Africa and has three main goals. Firstly, the research performed on the spelling checker is used as a basis of a course in computational linguistics. It is used to illustrate several techniques and allows students to get a feeling of problems and their solutions in the field of computational linguistics. Secondly, since the field of computational linguistics is still developing, hardly any resources exist. This project results in a spelling checker, but also a stemmer and a lexicon. Thirdly, it tries to implement an improved spelling checker that can be commercially exploited.

Acknowledgments

We would like to thank the following members of the research team for their input in this research: Roald Eiselen, Christo Els, Petri Jooste, Christo Muller, Sulene Pilon, Martin Puttkammer, Werner Ravyse, and Daan Wissing. We would also like to express our gratitude towards Attie de Lange, Ulrike Janke, Boeta Pretorius, Bertus van Rooy, and Elsa van Tonder for technical, administrative, and legal support. The Potchefstroom University for Christian Higher Education also sponsors this project generously—our thanks to Frikkie van Niekerk for his support. Finally, we would like to thank the anonymous reviewers for their excellent comments and suggestions.

References

Aduriz, I., Alegria, I., Artola, X., Ezeiza, N., Sarasola, K. and Urkia, M. (1997), A spelling corrector for Basque based on morphology, *Literary & Linguistic Computing* **12** (1), 31–38.

Combrink, J. (1990), *Afrikaanse Morfologie*, Academia Press, Pretoria.

van Huyssteen, G. B. (2002), Desiderata of spellchecking/spell-checking/spell checking: towards an intelligent spellchecker for Afrikaans, Symposium on the Developing of Spelling Checkers for South African Languages, Potchefstroom University for Christian Higher Education, Potchefstroom.

van Huyssteen, G. B. and van Zaanen, M. M. (2003), A spellchecker for Afrikaans, based on morphological analysis, *in* G.-M. De Schrijver (ed.), *Proceedings of the 6th International Terminology in Advanced Management Applications Conference (TAMA2003)*, (SF)2 Press, Pretoria, pp. 189–194.

Kernighan, M. D., Church, K. W. and Gale, W. A. (1990), A spelling correction program based on a noisy channel model, *Proceedings of the 13th International Conference on Computational Linguistics (COLING)*, Association for Computational Linguistics (ACL), Helsinki, pp. 205–210.

Koehn, P. and Knight, K. (2003), Empirical methods for compound splitting, *Proceedings of the 11th Annual Meeting of the European Chapter of the Association for Computational Linguistics (EACL)*, European Chapter of the Association for Computational Linguistics (EACL), Dublin. to be published.

Kraaij, W. and Pohlmann, R. (1994), Porter's stemming algorithm for Dutch, *in* L. Noordman and W. de Vroomen (eds), *Informatiewetenschap 1994: Wetenschappelijke bijdragen aan de derde STINFON Conferentie*, Stinfon, Tilburg, pp. 167–180.

Lu, S.-Y. L. and Fu, K.-S. (1977), Stochastic error-correcting syntax analysis for recognition of noisy patterns, *IEEE Transactions on Computers* **C-26** (12), 1268–1276.

Paggio, P. and Music, B. (1998), Evaluation in the SCARRIE project, *Proceedings of the First International Conference on Language Resources and Evaluation*, pp. 277–282.

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

Suid-Afrikaanse Akademie vir Wetenskap en Kuns (ed.) (2002), *Afrikaanse woordelys en spelreëls*, 9th edn, Pharos, Capetown.

Vandeghinste, V. (2002), Automated compounding as a means for maximizing lexical coverage in speech recognition, *in* M. Theune, A. Nijholt and H. Hondorp (eds), *Computational Linguistics in the Netherlands 2001*, Rodopi, Amsterdam, pp. 190–203.

Vosse, T. (1994), *The Word Connection*, PhD thesis, University of Leiden, Leiden.

Wagner, R. A. and Fischer, M. J. (1974), The string-to-string correction problem, *Journal of the Association for Computing Machinery* **21** (1), 168–173.