Finite-State Spell-Checking with Weighted Language and Error Models—Building and Evaluating Spell-Checkers with Wikipedia as Corpus

Tommi A Pirinen, Krister Lindén

University of Helsinki—Department of Modern Languages
Unioninkatu 40 A—FI-00014 Helsingin yliopisto
{tommi.pirinen,krister.linden}@helsinki.fi

Abstract

In this paper we present simple methods for construction and evaluation of finite-state spell-checking tools using an existing finite-state lexical automaton, freely available finite-state tools and Internet corpora acquired from projects such as Wikipedia. As an example, we use a freely available open-source implementation of Finnish morphology, made with traditional finite-state morphology tools, and demonstrate rapid building of Northern Sámi and English spell checkers from tools and resources available from the Internet.

1. Introduction

[1] Spell-checking is perhaps one of the oldest most researched application in the field of language technology, starting from the mid 20th century (Damerau, 1964). The task of spell-checking can be divided into two categories: isolated non-word errors and context-based real-word errors (Kukich, 1992). This paper concentrates on checking and correcting the first form, but the methods introduced are extendible to context-aware spell-checking.

To check whether a word is spelled correctly, a *language model* is needed. For this article, we consider a language model to be a one-tape finite-state automaton recognising valid word forms of a language. In many languages, this can be as simple as a word list compiled into a suffix tree automaton. However, for languages with productive morphological processes in compounding and derivation that are capable of creating infinite dictionaries, such as Finnish, a cyclic automaton is required. In order to suggest corrections, the correction algorithm must allow search from an infinite space. A nearest match search from a finite-state automaton is typically required (Oflazer, 1996). The reason we stress this limitation caused by morphologically complex languages is that often even recent methods for optimizing speed or accuracy suggest that we can rely on finite dictionaries or acyclic finite automata as language models.

To generate correctly spelled words from a misspelled word form, an *error model* is needed. The most traditional and common error model is the Levenshtein edit distance, attributed to Levenshtein (1966). In the edit distance algorithm, the misspelling is assumed to be a finite number of operations applied to characters of a string: deletion, insertion, change, or transposition[2]. The field of *approximate string matching* has been extensively studied since the mid 20th century, yielding efficient algorithms for simple string-to-string correction. For a good survey, see Kukich (1992). Research on approximate string matching has also provided different fuzzy search algorithms for finding the nearest match in a finite-state representation of dictionaries.

For the purpose of the article, we consider the error model to be any two-tape finite state automaton mapping any string of the error model alphabet to at least one string of the language model alphabet. As an actual implementation of Finnish spell-checking, we use a finite-state implementation of a traditional edit distance algorithm. In the literature, the edit distance model has usually been found to cover over 80 % of the misspellings at distance one (Damerau, 1964). Furthermore, as Finnish has a more or less phonemically motivated orthography, the existence of homophonic misspellings are virtually non-existent. In other words, our base assumption is that the greatest source of errors for Finnish spell-checking is the slip-of-the-finger style of typo, for which the edit distance is a good error model.

The statistical foundation for the language model and the error model in this article is similar to the one described by Norvig (2010), which also gives a good overview of the statistical basis for the spelling error correction problem along with a simple and usable python implementation.

For practical applications, the spell-checker typically needs to provide a small selection of the best matches for the user to choose from in a relatively short time span, which means that when defining corrections, it is also necessary to specify their likelihood in order to rank the correction suggestions. In this article, we show how to use a standard weighted finite-state framework to include probability estimates for both the language model and the error model. For the language model, we use simple unigram training with a Wikipedia corpus with the more common word forms to be suggested before the less common word forms. In the error model, we design the weights in the edit distance automaton so that suggestions with a greater Levenshtein-Damerau edit distance are suggested after those with fewer errors.

To evaluate the spell-checker even in the simple case of correcting non-word errors in isolation, a corpus of spelling mistakes with expected corrections is needed. Constructing such a corpus typically requires some amount of manual labour. In this paper, we evaluate the test results both against a manually collected misspelling corpus and against automatically misspelled texts. For a description of the er-

[1]This is the author's draft; it may differ from the published version

[2]Transposition is often attributed to an extended Levenshtein-Damerau edit distance given in (Damerau, 1964)

ror generation techniques, see Bigert (2005).

2. Goal of the paper

In this article, we demonstrate how to build and evaluate a spell-checking and correction functionality from an existing lexical automaton. We present a simple way to use an arbitrary string-to-string relation transducer as a misspelling model for the correction suggestion algorithm, and test it by implementing a finite-state form of the Levenshtein-Damerau edit distance relation. We also present a unigram training method to automatically rank spelling corrections, and evaluate the improvement our method brings over a correction algorithm using only the edit distance. The paper describes a work-in-progress version of a finite state spell-checking method with instructions for building the speller for various languages and from various resources. The language model in the article is an existing free open-source implementation of Finnish morphology[3] (Pirinen, 2008) compiled with HFST (Lindén et al., 2009)—a free, featurewise fully compliant implementation of the traditional Xerox-style LexC and TwolC tools[4].

One aim of this paper is to demonstrate the use of Wikipedia as a freely available open-source corpus[5]. The Wikipedia data is used in this experiment for training the lexical automaton with word form frequencies, as well as collecting a corpus of spelling errors with actual corrections.

The field of spell-checking is already a widely researched topic, cf. the surveys by Kukich (1992) and Schultz and Mihov (2002). This article demonstrates a generic way to use freely available resources for building finite-state spell-checkers. The purpose of using a basic finite-state algebra to create spell-checkers in this article is two-fold. Firstly, the amount of commonly known implementations of morphological language models under different finite-state frameworks suggest that a finite-state morphology is feasible as a language model for morphologically complex languages. Secondly, by demonstrating the building of an application for spell-checking with a freely available open-source weighted finite-state library, we hope to outline a generally useful approach to building open-source spell-checkers.

To demonstrate the feasibility of building a spell-checker from freely available resources, we use basic composition and n-best-path search with weighted finite-state automata, which allows us to use multiple arbitrary language and error models as permitted by the finite-state algebra. To the best of our knowledge, no previous research has used or documented this approach.

To further evaluate plausibility of rapid conversion from morphological or lexical automata to spell checkers we also sought and picked up a free open implementation of the Northern Sámi morphological analyzer[6] as well as a word list of English from (Norvig, 2010), and briefly tested them with the same methods and similar error model as for Finnish. While the main focus of the article is on the creation and evaluation of a Finnish finite-state spell-checker, we also show examples of building and evaluating spell-checkers for other languages.

3. Methods

The framework for implementing the spell-checking functionality in this article is the finite-state library HFST (Lindén et al., 2009). This requires that the underlying morphological description for spell-checking is compiled into a finite-state automaton. For our Finnish and Northern Sámi examples, we use a traditional linguistic description based on the Xerox LexC/TwolC formalism (Beesley and Karttunen, 2003) to create a lexical transducer that works as a morphological analyzer. As the morphological analyses are not used for the probability weight estimation in this article, the analysis level is simply discarded to get a one-tape automaton serving as a language model. However, the word list of English is directly compiled into a one tape suffix tree automaton.

As mentioned, the original language model can be as simple as a list of words compiled into a suffix tree automaton or as elaborate as a full-fledged morphological description in a finite-state programming language, such as Xerox LexC and TwolC[7]. The words that are found in the transducer are considered correct. The rest are considered misspelled.

It has previously been demonstrated how to add weights to a cyclic finite-state morphology using information on base-form frequencies. The technique is further described by Lindén and Pirinen (2009). In the current article, the word form counts are based on data from the Wikipedia. The training is in principle a matter of collecting the corpus strings and their frequencies and composing them with the finite-state lexical data. Deviating from the article by Lindén and Pirinen (2009), we only count full word forms. No provisions for compounding of word forms based on the training data are made, i.e. the training data is composed with the lexical model. This gives us an acyclic lexicon with the frequency data for correctly spelled words.

The actual implementation goes as follows. Clean up the Wikipedia dump to extract the article content from XML and Wikipedia mark-up by removing the mark-up and the contents of mark-up that does not constitute running text, leaving only the article content untouched. The tokenization is done by splitting text at white space characters and separating word final punctuation. Next we use the spell-checking automaton to acquire the correctly spelled word forms from the corpora, and count their frequencies. The formula for converting the frequencies f of a token in the corpus to a weight in the finite-state lexical transducer is $W_t = -\log \frac{f_t}{CS}$, where CS is the corpus size in tokens. The resulting strings with weights can then be compiled into paths of a weighted automaton, i.e. into an acyclic tree automaton with log probability weights in the final states of the word forms. The original language model is then

[3]http://home.gna.org/omorfi

[4]http://hfst.sf.net

[5]Database dumps available at http://download.wikimedia.org

[6]http://divvun.no

[7]It is also possible to convert aspell and hunspell style descriptions into transducers. Preliminary scripts exist in http://hfst.sf.net/.

weighted by setting all word forms not found in the corpus to a weight greater than the word with frequency of one, e.g. $W_{max} = -\log \frac{1}{CS+1}$. The simplest way to achieve this is to compose the $\Sigma^{\star}$ automaton with final weight W_{max} with the unweighted cyclic language model. Finally, we take the union of the cyclic model and the acyclic model. The word forms seen in the corpus will now have two weights, but the lexicon can be pruned to retain only the most likely reading for each string.

For example in the Finnish Wikipedia there were 17,479,297 running tokens[8], and the most popular of these is 'ja' *and* with 577,081 tokens, so in this language model the $W_{ja} = -\log \frac{577081}{17479297} \approx 4.44$. The training material is summarized in the Table 1. The token count is the total number of tokens after preprocessing and tokenization. The unique strings is the number of unique tokens that belonged to the language model, i.e. the size of actual training data, after uniqification and discarding potential misspellings and other strings not recognized by the language model. For this reason the English training model is rather small, despite the relative size of the corpus, since the finite language model only covered a very small portion of the unique tokens.

Language	Finnish	Northern Sámi	English
Token count	17,479,297	258,366	2,110,728,338
Unique language strings	968,996	44,976	34,920
Download size	956 MiB	8.7 MiB	5.6 GiB
Version used	2009-11-17	2010-02-22	2010-01-30

Table 1: Token counts for wikipedia based training material

For finding corrections using the finite-state methodology, multiple approaches with specialized algorithms have been suggested, e.g. (Oflazer, 1996; Schulz and Mihov, 2002; Huldén, 2009). In this article, we use a regular weighted finite-state transducer to represent a mapping of misspellings to correct forms. This allows us to use any weighted finite-state library that implements composition.

One of the simplest forms of mapping misspellings to correct strings is the edit distance algorithm usually attributed to Levenshtein (1966) and furthermore in the case of spell-checking to Damerau (1964). A finite-state automaton representation is given by e.g. Schulz and Mihov (2002). A transducer that corrects strings can be any arbitrary string-to-string mapping automaton, and can be weighted. In this article, we build an edit distance mapping transducer allowing two edits.

Since the error model can also be weighted, we use the weight W_{max} as the edit weight, which is greater than any of the weights given by the language model. As a consequence, our weighted edit distance will function like the traditional edit distance algorithm when generating the corrections for a language model, i.e. any correct string with edit distance one is considered to be a better correction than a misspelling with edit distance two. For example assuming misspelling 'jq' for 'ja', the error model would find 'ja' at an edit distance of W_{max}, but also e.g. 'jo' *already* and so on. In this case the frequency data obtained from Wikipedia will give us the popularity order of 'ja' > 'jo'. A fraction of the weighted edit distance two transducer is given in Figure 1. The transducer in the figure displays a full edit distance two transducer for a language with two symbols in the alphabet; an edit distance transducer for a full alphabet is simply a union of such transducers for each pair of symbols in the language[9].

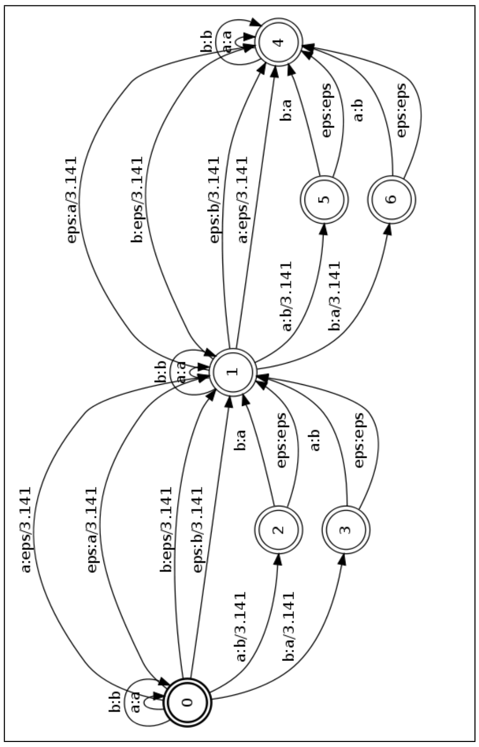

Figure 1: Edit distance transducer of alphabet a, b length two and weight π.

To get a ranked set of spelling correction suggestions, we simply compile the misspelled word into a path automaton T_{word}. The path automaton is composed with the correction relation T_E—in this case the weighted edit distance two transducer—to get an automaton that contains all the possible spelling corrections $T_{sug} = T_{word} \circ T_E$. We then compose the resulting automaton with the original weighted lexical data T_L to find the string corrections that are real words of the language model $T_f = T_{sug} \circ T_L$. The resulting transducer now contains a union of words with the combined weight of the frequency of the word form and the weight of the edit distance. From this transducer, a ranked list of spelling suggestions is extracted by a standard n-best-path algorithm listing unique suggestions.

4. Test Data Sets

For the Finnish test material, we use two types of samples extracted from Wikipedia. First, we use a hand-picked selection of 761 misspelled strings found by browsing the

[8]We used relatively naive preprocessing and tokenization, splitting at spaces and filtering html and Wikipedia markup

[9]For the source code of the Finnish edit distance transducer in the HFST framework, see `http://svn.gna.org/viewcvs/omorfi/trunk/src/suggestion/edit-distance-2.text`

strings that the speller rejected. These strings were manually corrected using a native reader's best judgement from reading the misspelled word in context to achieve a gold standard for evaluation.

Another larger set of approximately 10,000 evaluation strings was created by using the strings from the same Wikipedia corpus, and automatically introducing spelling errors similar to the approach described by Bigert et al. (2003), using isolated word Damerau-Levenshtein type errors with a probability of approximately 0.33 % per character. This error model could also be considered an error model applied in reverse compared to the error model used when correcting misspelled strings. As there is nothing limiting the number of errors generated per word except the word length, this error model may introduce words with an edit distance greater than two.

As the Northern Sámi gold standard, we used the test suite included in the svn distribution[10]. It seems to contain a set of common typos.

As the English gold standard for evaluation, we use the Birkbeck spelling error corpus referred to in (Norvig, 2010). The corpus has restricted free licensing, but the restrictions prohibit its use as training material in a free open source project.

5. Evaluation

To evaluate the correction algorithm, we use the two data sets introduced in the previous section. However, we use a slightly different error model to automatically correct misspellings than we use for generating them, i.e. some errors exceeding the edit distance of two are unfixable by the error model we use for correction.

The evaluation of the correction suggestion quality is given in Tables 2 and 3. The Table 2 contains precision values for the spelling errors from real texts, and Table 3 for the automatically introduced spelling errors. The precision is measured by ranked suggestions. In the tables, we give the results separately for ranks 1—4, and for the remaining lower ranks. The lower ranks ranged from 5—440 where the number of total suggestions ranged from 1—600. In the last column, we have the cases where a correctly written word could not be found with the proposed suggestion algorithm. The tables contain both the results for the weighted edit distance relation, and for a combination of the weighted edit distance relation and the word form frequency data from Wikipedia.[11]

As a first impression we note that mere Wikipedia training does improve the results in all cases; the number of suggestions in first position rises in all test sets and languages. This suggests that more mistakes are made in common words than in rare ones, since the low ranking word counts did not increase as a result of Wikipedia training.

In the Finnish tests, haplological cases like 'kokonaismalmivaroista' *from total ore resources* spelled as 'konaismalmivaroista' came in at the bottom of the list for both methods, because the correct word form is probably nonexistent in the training corpus, and the multi-part productive compound with an ambiguous segmentation produces lots of nearer matches at edit distance one. A more elaborate error model considering haplology as a misspelling with a weight equal or less than a single traditional edit distance would of course improve the suggestion quality in this case. The number of words getting no ranks is common to both methods. They indicate the spelling errors for which the correct form was neither among the ones covered by the error model for edit distance two nor in the language model. A substantial number are cases which were not considered in the error model, e.g. a missing space causing run-on words ('ensisijassa' instead of 'ensi sijassa' *in the first place*). A good number also comes from spoken or informal language forms for very common words, which tend to deviate more than edit distance two ('esmeks' instead of 'esimerkiksi' *for example*), with a few more due to missing forms in the language model. E.g. 'bakterisidin' is one edit from 'bakterisidina' *as bactericide*, but the correction is not made because the word does not exist in the language model. These error types are correctable by adding words to the lexicon, i.e. the language model, e.g. using special-purpose dictionaries, such as spoken language or medical dictionaries. Finally there is a handful of errors that seem legitimate spelling mistakes of more than two edits ('assosioitten' instead of 'assosiaatioiden'). For these cases, a different error model than the basic edit distance might be necessary.

Material	Rank 1	2	3	4	Lower	No rank	Total
Weighted edit distance 2							
Finnish	371	118	65	33	103	84	761
	49 %	16 %	9 %	4 %	14 %	11 %	100 %
Northern	2221	697	430	286	2743	2732	9115
Sámi	24 %	8 %	5 %	3 %	30 %	30 %	100 %
English	8739	2695	1504	940	3491	17738	35106
	25 %	8 %	4 %	3 %	10 %	51 %	100 %
Wikipedia word form frequencies and edit distance 2							
Finnish	451	105	50	22	62	84	761
	59 %	14 %	7 %	3 %	8 %	11 %	100 %
Northern	2421	745	427	266	2518	2732	9115
Sámi	27 %	8 %	5 %	3 %	28 %	30 %	100 %
English	9174	2946	1489	858	2902	17738	35106
	26 %	8 %	4 %	2 %	8 %	51 %	100 %

Table 2: Precision of suggestion algorithms with real spelling errors

Material	Rank 1	2	3	4	Lower	No rank	Total
Weighted edit distance 2							
Finnish	4321	1125	565	351	1781	1635	10076
	43 %	11 %	6 %	3 %	18 %	16 %	100 %
Northern	1269	257	136	80	528	7730	10000
Sámi	13 %	3 %	1 %	1 %	5 %	77 %	100 %
English	4425	938	337	290	1353	2657	10000
	44 %	10 %	3 %	3 %	14 %	27 %	100 %
Wikipedia word form frequencies and edit distance 2							
Finnish	4885	1128	488	305	1407	1635	10076
	49 %	11 %	5 %	3 %	14 %	16 %	100 %
Northern	1726	253	76	29	186	7730	10000
Sámi	17 %	3 %	1 %	1 %	2 %	77 %	100 %
English	5584	795	307	196	461	2657	10000
	56 %	8 %	3 %	2 %	5 %	27 %	100 %

Table 3: Precision of suggestion algorithms with generated spelling errors

[10] `https://victorio.uit.no/langtech/trunk/gt/sme/src/typos.txt`

[11] For full tables and test logs, see `http://home.gna.org/omorfi/testlogs`.

For the Northern Sámi spelling error corpus we note that a large amount of errors is not covered by the error model. This means that the error model is not sufficient for Northern Sámi spell checking as we can see a number of errors with edit distance greater than 2, e.g. 'sáddejun' instead of 'sáddejuvvon'.

Comparing our English test results with previous research using implementations of the same language and error model, we reiterate that a great number of words are out of reach by an error model of mere edit distance 2. Some of the test words are even word usage errors, such as 'gone' in stead of 'went', but unfortunately they were intermixed with the other spelling error material and we did not have time to remove them from the test corpus. The rest of the spelling errors beyond edit distance 2 are mostly caused by English orthography being relatively distant from pronunciation, such as 'negoshayshauns' in stead of 'negotiations', which usually are corrected with very different error models such as soundex and other phonetic keys as demonstrated by e.g. Mitton (2009). The results of the evaluation of the correction suggestions show a similar tendency as the one found in the original article by Norvig (2010).

The impact on performance when using non-optimized methods to check spelling and get suggestion lists was not thoroughly measured, but to give an impression of the general applicability of the methods, we note that for the Finnish material of generated misspellings, the speed of spell-checking was 3.18 seconds for 10,000 words or approx. 3,000 words per second, and the speed of generating suggestion lists, i.e. all possible corrections for 10,000 misspelled words took 10,493 seconds, i.e. on the average it took approx. 1 second to generate all the suggestions for each misspelled word, when measured with the GNU `time` utility and the `hfst-omor-evaluate` program from the HFST toolkit, which batch processes spell-checking tasks on tokenized input and evaluates precision and recall against a correction corpus. The space requirements for the Finnish spell checking automata are 9.2 MiB for the Finnish morphology and 378 KiB for the Finnish edit distance two automaton with an alphabet size of 72. As a comparison, the English language model obtained from the word list is only 3.2 MiB in size, and correspondingly the error model 273 KiB with an alphabet size of 54.

6. Discussion

The obvious and popular development is to extend the language model to support n-gram dictionaries with $n > 1$, which has been shown to be a successful technique for English e.g. by Mays (1991). The extension using the same framework is not altogether trivial for a language like Finnish, where the number of forms and unique strings are considerable giving most n-grams a very low-frequency.

Even if the speed and resource use for spell-checking and correction was found to be reasonable, it may still be interesting to optimize for speed as has been shown in the literature (Oflazer, 1996; Schulz and Mihov, 2002; Huldén, 2009). At least the last of these is readily available as an open-source finite-state implementation in foma[12], and is expandable for at least a non-homogeneous non-unit-weight edit distance with context restrictions. However, it does not yet cater to general weighted language models.

Other manual extensions to the spelling error model should also be tested. Our method ensures that arbitrary weighted relations can be used. Especially the use of a non-homogeneous non-unit-length edit distance can easily be achieved. Since it has been successfully used in e.g. hunspell[13], it should be further evaluated. Other obvious and common improvements to the edit distance model is to scale the weights of the edit distance by the physical distance of the keys on a QWERTY keyboard.

The acquisition of an error model or probabilities of errors in the current model is also possible (Brill and Moore, 2000), but this requires the availability of an error corpus containing a large (representative) set of spelling errors and their corrections, which usually are not available nor easy to create. One possible solution for this may of course be to implement an adaptive error model that modifies the probabilities of the errors for each correction made by user.

The methods were only evaluated on languages with substantial resources, but the use of freely available language resources and toolkits makes the proposed methods for creating spell-checkers interesting for less resourced languages as well, since most written languages already have text corpora, word lists and inflectional descriptions.

The next step is to improve the free open-source Voikko[14] spell-checking library with the HFST transducer-based spell-checking library. Voikko has been successfully used in open-source software such as OpenOffice.org, Mozilla Firefox, and the Gnome desktop (in the enchant spell-checking library).

7. Conclusions

In this article we have demonstrated an approach for creating spell-checkers from various language models—ranging from simple word lists to complex full-fledged implementations of morphology—built into a finite-state automata. We also demonstrated a simple approach to training the models using word frequency data extracted from Wikipedia. Further, we have presented a construction of a simple edit distance error model in the form of a weighted finite-state transducer, and proven usability of this basic finite state approach by evaluating the resulting spell-checkers against both manually collected smaller and automatically created larger error corpora. Given the amount of finite-state implementations of morphological language models it seems reasonable to expect that general finite-state methods and language models can support spell-checking for a large array of languages. The methods may be especially useful for less resourced languages, since most written languages already have text corpora, word lists and inflectional descriptions.

The fact that arbitrary weighted two-tape automata may be used for implementing error models suggests that it is relatively easy to implement different error models with available open-source finite-state toolkits. We also showed that

[12]`http://foma.sf.net`

[13]`http://hunspell.sf.net`

[14]`http://voikko.sf.net/`

combining the basic edit distance error model with a simple unigram frequency model already improves the quality of the error corrections. We also note that even using a basic finite-state transducer algebra from a freely available finite-state toolkit and no specialized algorithms, the speed and memory requirements of the spell-checking seems sufficient for typical interactive usage.

8. Acknowledgements

We thank our colleagues in the HFST research team and the anonymous reviewers for valuable suggestions regarding the article.

9. References

Kenneth R Beesley and Lauri Karttunen. 2003. *Finite State Morphology*. CSLI publications.

Johnny Bigert, Linus Ericson, and Antoine Solis. 2003. Autoeval and missplel: Two generic tools for automatic evaluation. In *Nodalida 2003*, Reykjavik, Iceland.

Johnny Bigert. 2005. *Automatic and unsupervised methods in natural language processing*. Ph.D. thesis, Royal institute of technolog (KTH).

Eric Brill and Robert C. Moore. 2000. An improved error model for noisy channel spelling correction. In *ACL '00: Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 286–293, Morristown, NJ, USA. Association for Computational Linguistics.

F J Damerau. 1964. A technique for computer detection and correction of spelling errors. *Commun. ACM*, (7).

Måns Huldén. 2009. Fast approximate string matching with finite automata. *Procesamiento del Lenguaje Natural*, 43:57–64.

Karen Kukich. 1992. Techniques for automatically correcting words in text. *ACM Comput. Surv.*, 24(4):377–439.

V. I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics—Doklady 10, 707710. Translated from Doklady Akademii Nauk SSSR*, pages 845–848.

Krister Lindén and Tommi Pirinen. 2009. Weighting finite-state morphological analyzers using hfst tools. In Bruce Watson, Derrick Courie, Loek Cleophas, and Pierre Rautenbach, editors, *FSMNLP 2009*, 13 July.

Krister Lindén, Miikka Silfverberg, and Tommi Pirinen. 2009. Hfst tools for morphology—an efficient open-source package for construction of morphological analyzers. In Cerstin Mahlow and Michael Piotrowski, editors, *sfcm 2009*, volume 41 of *Lecture Notes in Computer Science*, pages 28—47. Springer.

Eric Mays, Fred J. Damerau, and Robert L. Mercer. 1991. Context based spelling correction. *Inf. Process. Manage.*, 27(5):517–522.

Roger Mitton. 2009. Ordering the suggestions of a spellchecker without using context*. *Nat. Lang. Eng.*, 15(2):173–192.

Peter Norvig. 2010. How to write a spelling corrector. Web Page, Visited February 28th 2010, Available `http://norvig.com/spell-correct.html`.

Kemal Oflazer. 1996. Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. *Comput. Linguist.*, 22(1):73–89.

Tommi Pirinen. 2008. Suomen kielen äärellistilainen automaattinen morfologinen analyysi avoimen lähdekoodin menetelmin. Master's thesis, Helsingin yliopisto.

Klaus Schulz and Stoyan Mihov. 2002. Fast string correction with levenshtein-automata. *International Journal of Document Analysis and Recognition*, 5:67–85.