WNSpell: a WordNet-Based Spell Corrector

Bill Huang
Princeton University
yh3@princeton.edu

Abstract

This paper presents a standalone spell corrector, WNSpell, based on and written for WordNet. It is aimed at generating the best possible suggestion for a mistyped query but can also serve as an all-purpose spell corrector. The spell corrector consists of a standard initial correction system, which evaluates word entries using a multifaceted approach to achieve the best results, and a semantic recognition system, wherein given a related word input, the system will adjust the spelling suggestions accordingly. Both feature significant performance improvements over current context-free spell correctors.

1 Introduction

WordNet is a lexical database of English words and serves as the premier tool for word sense disambiguation. It stores around 160,000 word forms, or lemmas, and 120,000 word senses, or synsets, in a large graph of semantic relations. The goal of this paper is to introduce a spell corrector for the WordNet interface, directed at correcting queries and aiming to take advantage of WordNet's structure.

1.1 Previous Work

Work on spell checkers, suggesters, and correctors began in the late 1950s and has developed into a multifaceted field. First aimed at simply detecting spelling errors, the task of spelling correction has grown exponentially in complexity.

The first attempts at spelling correction utilized edit distance, such as the Levenshtein distance, where the word with minimal distance would be chosen as the correct candidate.

Soon, probabilistic techniques using noisy channel models and Bayesian properties were invented. These models were more sophisticated, as they also considered the statistical likeliness of certain errors and the frequency of the candidate word in literature.

Two other major techniques were also being developed. One was similarity keys, which used properties such as the word's phonetic sound or first few letters to vastly decrease the size of the dictionary to be considered. The other was the rule-based approach, which implements a set of human-generated common misspelling rules to efficiently generate a set of plausible corrections and then matching these candidates with a dictionary.

With the advent of the Internet and the subsequent increase in data availability, spell correction has been further improved. N-grams can be used to integrate grammatical and contextual validity into the spell correction process, which standalone spell correction is not able to achieve. Machine learning techniques, such as neural nets, using massive online crowdsourcing or gigantic corpora, are being harnessed to refine spell correction more than could be done manually.

Nevertheless, spell correction still faces significant challenges, though most lie in understanding context. Spell correction in other languages is also incomplete, as despite significant work in English lexicography, relatively little has been done in other languages.

1.2 This Project

Spell correctors are used everywhere from simple spell checking in a word document to query completion/correction in Google to context-based in-passage corrections. This spell corrector, as it is for the WordNet interface, will focus on spell correction on a single word query with the additional possibility of a user-inputted semantically-related word from which to base corrections off of.

2 Correction System

The first part of the spell corrector is a standard context-free spell corrector. It takes in a query such as $speling$ and will return an ordered list of three possible candidates; in this case, it returns the set $\{spelling, spoiling, sapling\}$.

The spell corrector operates similarly to the Aspell and Hunspell spell correctors (the latter which serves as the spell checker for many applications varying from Chrome and Firefox to OpenOffice and LibreOffice). The spell corrector we introduce here, though not as versatile in terms of support for different platforms, achieves far better performance.

To tune the spell corrector to WordNet queries, stress is placed on bad misspellings over small errors. We will mainly use the Aspell data set (547 errors), kindly made public by the GNU Aspell project, to test the performance of the spell corrector. Though the mechanisms of the spell corrector are inspired by logic and research, they are included and adjusted mainly based on empirical tests on the above data set.

2.1 Generating the Search Space

To improve performance, the spell corrector needs to implement a fine-tuned scoring system for each candidate word. Clearly, scoring each word in WordNet's dictionary of 150,000 words is not practical in terms of runtime, so the first step to an accurate spell corrector is always to reduce the search space of correction candidates.

The search space should contain all possible reasonable sources of the the spelling error. These errors in spelling arise from three separate stages (Deorowicz and Ciura, 2005):

1. Idea $\rightarrow$ thought word

   i.e. $distrucally \rightarrow destructfully$

2. Thought word $\rightarrow$ spelled word

   i.e. $egsistance \rightarrow existence$

3. Spelled word $\rightarrow$ typed word

   i.e. $autocorrecy \rightarrow autocorrect$

The main challenges regarding search space generation are:

1. Containment of all, or nearly all, possible reasonable corrections
2. Reasonable size
3. Reasonable runtime

There have been several approaches to this search space problem, but all have significant drawbacks in one of the criteria of search space generation:

- The simplest approach is the lexicographic approach, which simply generates a search space of words within a certain edit distance away from the query. Though simple, this minimum edit distance technique, introduced by Damerau in 1964 and Levenshtein in 1966, only accounts for type 3 (and possibly type 2) misspellings. The approach is reasonable for misspellings of up to edit distance 2, as Norvig's implementation of this runs in $\sim$0.1 seconds, but time complexity increases exponentially and for misspellings such as $funetik \rightarrow phonetic$ that are a significant edit distance away, this approach will not be able to contain the correction without sacrificing both the size of the search space and the runtime.

- Another approach is using phonetics, as misspelled words will most likely still have similar phonetic sounds. This accounts for type 2 misspellings, though not necessarily type 1 or type 3 misspellings. Implementations of this approach, such as using the SOUNDEX code (Odell and Russell, 1918), are able to efficiently capture misspellings such as $funetik \rightarrow phonetic$, but not misspellings like $rypo \rightarrow typo$. Again, this approach is not sufficient in containing all plausible corrections.

- A similarity key can also be used. The similarity key approach stores each word under a key, along with other similar words. One implementation of this is the SPEED-COP spell corrector (Pollock and Zamora, 1984), which takes advantage of the usual alphabetic proximity of misspellings to the correct word. This approach accounts for many errors, but there are always a large number of exceptions, as the misspellings do not always have similar keys (such as the misspelling $zlphabet \rightarrow alphabet$).

- Finally, the rule-based approach uses a set of common misspelling patterns, such as $im \rightarrow in$ or $y \rightarrow t$, to generate possible sources of the typing error. The most complicated approach, these spell correctors are able to contain the plausible corrections for most spelling errors quite well, but will miss many of the bad misspellings. The implementation by Deoroicz and Ciura using this approach is quite effective, though it can be improved.

Our approach with this spell corrector is to use a combination of these approaches to achieve the best results. Each approach has its strengths and weaknesses, but cannot achieve a good coverage of the plausible corrections without sacrificing size and runtime. Instead, we take the best of each approach to much better contain the plausible corrections of the query.

To do this, we partition the set of plausible corrections into groups (not necessarily disjoint, but with a very complete union) and consider each separately:

- Close mistypings/misspellings:

  This group includes typos of edit distance 1 ($typo \rightarrow rypo$) and misspellings of edit distance 1 ($consonent \rightarrow consonant$), as well as repetition of letters ($mispel \rightarrow misspell$). These are easy to generate, running in $O(n \log n\alpha)$ time, where n is the length of the entry and α is the size of the alphabet, to generate and check each word (though increasing the maximum distance to 2 would result an significantly slower time of $O(n^2 \log n\alpha^2)$.

- Words with similar phonetic key:

  We implement a precalculated phonetic key for each word in WordNet, which uses a numerical representation of the first five consonant sounds of the word:

  0: (ignored) a, e, i, o, u, h, w, [gh](t)
  1: b, p
  2: k, c, g, j, q, x
  3: s, z, c(i/e/y), [ps], t(i o), (x)
  4: d, t
  5: m, n, [pn], [kn]
  6: l
  7: r
  8: f, v, (r/n/t o u)[gh], [ph]

  Each word in WordNet is then stored in an array with indices ranging from [00000] (no consonants) to [88888] and can be looked up quickly.

  This group includes words with a phonetic key that differs by an edit distance at most 1 from the phonetic key of the entry ($funetik \rightarrow phonetic$), and also does a very good job of including typos/misspellings of edit distance greater than 1 (it actually includes the first group completely, but for pruning purposes, the first group is considered separately) in very little time $O(Cn)$ where $C \sim 5^2 \times 9$.

- Exceptions:

  This group includes words that are not covered by either of the first two groups but are still plausible corrections, such as $lignuitic \rightarrow linguistic$. We observe that most of these exceptions either still have similar beginning and endings to the original word and are close edit distance-wise or are simply too far-removed from the entry to be plausible. Searching through words with similar beginnings that also have similar endings (through an alphabetically-sorted list) proves to be very effective in including the exception, while taking very little time.

As many generated words, especially from the later groups, are clearly not plausible corrections, candidate words of each type are then pruned with different constraints depending on which group they are from. Words in later groups are subject to tougher pruning, and the finding of a close match results in overall tougher pruning.

For instance, many words in the second group are quite far removed from the entry and completely implausible as corrections (e.g. $zjpn \rightarrow$ [00325] $\rightarrow$ [03235] $\rightarrow suggestion$), while those that are simply caused by repetition of letters (e.g. $lllooollll \rightarrow loll$) are almost always plausible, so the former group should be more strictly pruned.

Finally, since the generated search space after group pruning can be quite large (up to 200), depending on the size of the search space, the search space may be pruned, repetitively, until the size of the search space is of an acceptable size.

Some factors considered during pruning include:

- Length of word
- Letters contained in word
- Phonetic key of word

- First and last letter agreement
- Number of syllables
- Frequency of word in text (COCA corpus)
- Edit distance

This process successfully generates a search space that rarely misses the desired correction, while keeping both a small size in number of words and a fast runtime.

2.2 Evaluating Possibilities

The next step is to assign a similarity score to all of the candidates in the search space. It must be accurate enough to discern that $disurn \longrightarrow discern$ but $disurn \not\longrightarrow disown$ and versatile enough to figure out that $funetik \longrightarrow phonetic$.

Our approach is a modified version of Church and Gale's probabilistic scoring of spelling errors. In this approach, each candidate correction c is scored following the Bayesian combination rule:

$$P(c) = p(c) \max\Big(\prod_i p(t_i \mid c_i)\Big)$$
$$C(c) = c(c) + \min\Big(\sum_i c(t_i \mid c_i)\Big)$$

Where $P(c)$ is the frequency of the candidate correction, $P(t_i \mid c_i)$ the cost of each edit distance operation in a sequence of edit operations that generate the correction. The cost is then scored logarithmically based on the probability, where $c(t_i \mid c_i) \propto -\log\big(p(t_i \mid c_i)\big)$. The correction candidates are then sorted, with lower cost meaning higher likelihood.

We use bigram error counts generated from a corpora (Jones and Mewhort, 2004) to determine the values of $c(t \mid p)$. Two sets of counts were used:

- Error counts:
  - Deletion of letter β after letter α
  - Addition of letter β after letter α
  - Substitution of letter β for letter α
  - Adjacent transposition of the bigram $\alpha\beta$
- Bigram/monogram counts (log scale):
  - Monograms α
  - Bigrams $\alpha\beta$

First, we smooth all the counts using add-k smoothing (where we set $k = \frac{1}{2}$), as there are numerous counts of 0. Since the bigram/monogram counts were retrieved in log format, for sake of simplicity of data manipulation, we only smooth the counts of 0, changing their values to -0.69 (originally undefined). We then calculate $c(t_i \mid c_i)$ as:

$$c(t_i \mid c_i) = k_1 \log\left(\frac{1}{p(\alpha \to \beta)}\right) + k_2$$

Where $p(\alpha \to \beta)$ is the probability of the edit operation and k_1, k_2 factors that adjust the cost depending on the uncertainty of small counts and the increased likelihood of errors if errors are already present.

For the different edit operations, $p(x \to y)$ is:

$$p(x \to y) = \begin{cases} \text{deletion :} & \frac{\text{del}'(xy)}{N'(xy)} \\ \text{addition :} & \frac{\text{add}'(xy) \cdot N}{N'(x) \cdot N'(y)} \\ \text{substitution :} & \frac{\text{sub}'(xy) \cdot N}{N'(x) \cdot N'(y)} \\ \text{reversal :} & \frac{\text{rev}'(xy)}{N'(xy)} \end{cases}$$

And for deletion and addition of letters at the beginning of a word:

$$p(x \to y) = \begin{cases} \text{deletion :} & \frac{\text{del}'(.y)}{N'(.y)} \\ \text{addition :} & \frac{(\text{add}'(.y)) \cdot N \cdot w}{N'(y)} \end{cases}$$

To evaluate the minimum cost $\min\big(\sum_i c(t_i \mid c_i)\big)$ of a correction, we use a modified Wagner-Fischer algorithm, finds the minimum in $O(mn)$ time, where m, n are the lengths of the entry and correction candidate, respectively. This is done over for candidate corrections in the search space generated in (3.1).

Now, the probabilistic scoring by itself is not always accurate, especially in cases such as $funetik \longrightarrow phonetic$. Thus, we modify the scoring of each candidate correction to significantly improve the accuracy of the suggestions:

- Instead of setting $c(c) = -\log(p(c)$, we find that using $c(c)$ as multiplicative constant as a function $f(c)^\gamma$, where $f(c)$ is the frequency of the word in the corpus and γ an empirically-determined constant, yields significantly more accurate predictions.
- We add empirically-determined multiplicative factors λ_i pertaining to the following factors regarding the entry and the candidate correction:

  - Same phonetic key (not restricted to first 5 consonant sounds)
  - Same aside from repetition of letters
  - Same consonants (ordered)
  - Same vowels (ordered)
  - Same set of letters
  - Similar set of letters
  - Same number of syllables
  - Same after removal of es

  (Note that other factors were considered but the factors pertaining to them were insignificant)

The candidate corrections are then ordered by their modified costs $C'(c) = C(c)\prod_i \lambda_i$ and the top three results, in order, are returned to the user.

3 Semantic Input:

The second part of the spell corrector adds a semantic aspect into the correction of the search query. When users have trouble entering the query and cannot immediately choose a suggested correction, they are given the option to enter a semantically related word. WNSpell then takes this word into account when generating suggestions, harnessing WordNet's vast semantic network to further optimize results.

This added dimension in spell correction is very helpful for the more severe errors, which usually arise from the "idea $\rightarrow$ thought word" process in spelling. These are much harder to deal with than conventional mistypings or misspellings, and are exactly the type of error WNSpell needs to be able to handle (as mistyped or even misspelled queries can be fixed without too much trouble by the user). The semantic anchor the related word provides helps WNSpell establish the idea" behind the desired word and thus refine the suggestions for the desired word.

To incorporate the related word into the suggestion generation, we add some modifications to the original context-free spell corrector.

3.1 Adjusting the Search Space:

One of the issues in search space generation in the original is that a small fraction of plausible corrections are still missed, especially in more severe errors. To improve the coverage of the search space, we modify the search space to also include a nucleus of plausible corrections generated semantically, not just lexicographically. Since the missed corrections are lexicographically difficult to generate, using a semantic approach would be more effective in increasing coverage.

The additional group of word forms is generated as follows:

1. For each synset of the related word, we consider all synsets related to it by some semantic pointer in WordNet.

2. All lemmas (word forms) of these synsets are evaluated.

3. Lemmas that share the same first letter or the same last letter and are not too far away in length are added to the group.

The inclusion of the additional group is indeed very effective in capturing the missed corrections and remains relatively small in size.

Some examples of missed words captured in this group from the training set are (entry, correct, related):

- *autoamlly, automatically, mechanically*

- *conibation, contribution, donation*

3.2 Adjusting the Evaluation:

We also modify the scoring process of each candidate correction to take into account semantic distance. First, each candidate correction is assigned a semantic distance d (higher means more similar) based on Lesk distance:

$$d = \max_i \max_j s(r_i, c_j)$$

Which takes the maximum similarity over all pairs of definitions of the related word r and candidate c where similarity s is measured by:

$$s(r_i, c_j) = \sum_{w \in R_i \cap C_j, w \notin S} k - \ln(n_w + 1)$$

Which considers words w in the intersection of the definitions that are not stopwords and weights them by the smoothed frequency n_w of w in the COCA corpus (as rarity is related to information content) and some appropriate constant k.

Additionally, if r or c is found in the other definition, we also add to the similarity s of two definitions $a\big(k - \ln(n_{r/c} + 1)\big)$ for some appropriate constant $a > 1$. This resolves many issues that come up with hypernyms/hyponyms (among others) where two similar words are assigned a low score since the only words in common in their definitions may be the words themselves.

We also consider the number n of shared subsequences of length 3 between r and c, which is very helpful in ruling out semantically similar but lexicographically unlikely words.

We then adjust the cost function C' by:

$$C'' = \frac{C'}{(d+1)^{\alpha}(n+1)^{\beta}}$$

For some empirically-determined constants α and β. The new costs are then sorted and the top three results returned to the user.

4 Results

We used the Aspell data set to train the system. The test set consists of 547 hard-to-correct words. This is ideal for our purposes, as we are focusing on correcting bad misspellings as well as the easy ones. Most of the empirically-derived constants from (3.2) were determined based off of results from this data set.

4.1 Without Semantic Input

We compare the results of WNSpell to a few popular spellcheckers: Aspell, Hunspell, Ispell, and Word; as well as with the proposition of Deorowicz and Ciura, which seems to have the best results on the Aspell test set so far (other approaches are based off of unavailable/uncompatible data sets).

Ideally, for comparison, it would be ideal to run each spell checker on the same lexicon and on the same computer for consistent results. However, due to technical constraints, it is rather infeasible to do so. Instead, we will use the results posted by the authors of the spell checkers, which, despite some uncertainty, will still yield consistent and comparable results.

First, we compare our generated search space with the lists returned by Aspell, Hunspell, Ispell, and Word (Atkinson). We use a subset of the Aspell test set containing all entries whose corrections are in all five dictionaries. The results are shown in Table 1.

Search Space Results

Method	% found	Size (0/50/100%)		
WNSpell	97.4	1	10	66
Aspell (0.60.6n)	90.1	2	12	100
Hunspell (1.1.12)	83.2	1	4	15
Ispell (3.1.20)	54.8	0	1	29
Word 97	75.4	0	2	20

Table 1

Compared to these three spell correctors, WNSpell clearly does a significantly better job containing the desired correction than Aspell, Hunspell, Ispell, or Word within a set of words of acceptable size.

We now compare the results of the top three words returned on the list with those returned by Aspell, Hunspell, Ispell, Word. We also include data from Deorowicz and Ciura, which also uses the Aspell test set. Since the dictionaries used were different, we also include Aspell results using their subset of the Aspell test set. The results are shown in Table 2, and a graphical comparison is shown in Figure 1.

Once again, WNSpell significantly outperforms the other five spell correctors.

Aspell Test Set Results (% Identified)

Method	Top 1	Top 2	Top 3	Top 10
WNSpell	77.5	88.5	91.2	96.1
Aspell (0.60.6n)	54.3	63.0	72.9	87.1
Hunspell (1.1.12)	58.2	71.5	76.6	82.3
Ispell (3.1.20)	40.1	47.9	50.4	54.1
Word 97	62.6	69.4	72.7	75.4
Aspell (n)	56.9	66.9	74.7	87.9
DC	66.3	75.5	79.6	85.5

Table 2

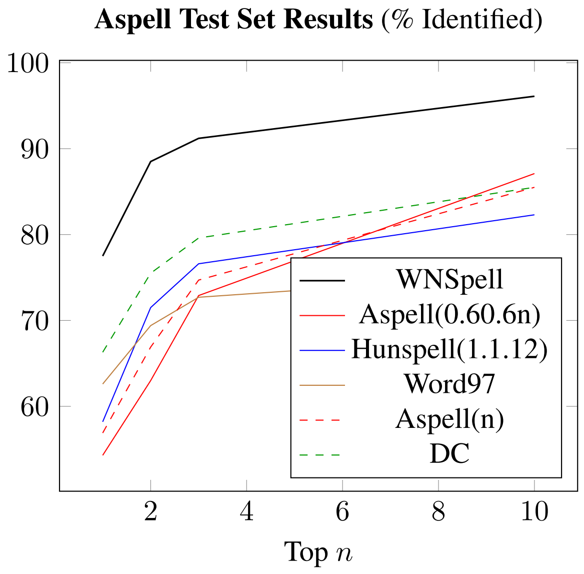

Figure 1

We also test WNSpell on the Aspell common misspellings test set, a list of 4206 common misspellings and their corrections. Since the word corrector was not trained on this set, it is a blind comparison. Once again, we use a subset of the Aspell test set containing all entries whose corrections are in all five dictionaries. The results are shown in tables 3 and 4, and a graphical comparison is shown in Figure 2.

Blind Search Space Results

Method	% found	Size (0/50/100%)		
WNSpell	98.4	1	4	50
Aspell (0.60.6n)	97.7	1	9	100
Hunspell (1.1.12)	97.3	1	5	15
Ispell (3.1.20)	85.2	0	1	26

Table 3

Blind Test Set Results

Method	Top 1	Top 2	Top 3	Top 10
WNSpell	91.4	96.3	97.6	98.3
Aspell (0.60.6n)	73.6	81.2	92.0	97.0
Hunspell (1.1.12)	80.8	92.0	95.0	97.3
Ispell (3.1.20)	77.4	82.7	84.3	85.2

Table 4

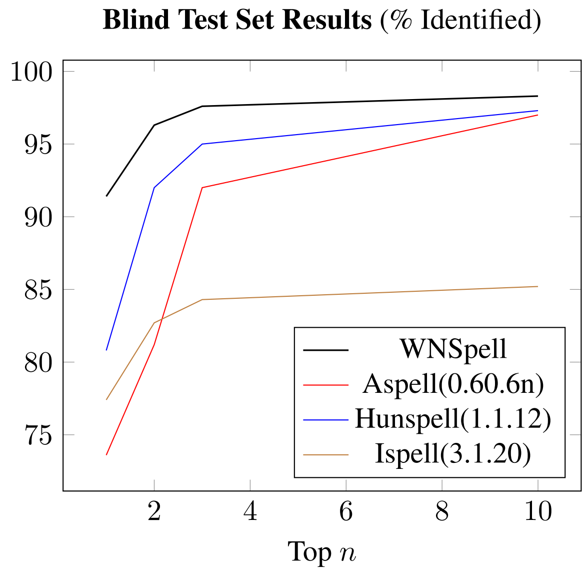

Figure 2

Additionally, WNSpell runs in decently fast time. WNSpell takes ∼13ms per word, while Aspell takes ∼3ms, Hunspell ∼50ms, and Ispell ∼0.3ms. Thus, WNSpell is a very efficient standalone spell corrector, achieving superior performance within acceptable runtime.

4.2 With Semantic Input

We test WNSpell with the semantic component on the original training set, this time with added synonyms. For each word in the training set, a human-generated related word is inputted.

With the addition of the semantic adjustments, WNSpell performs considerably better than without them. The results are shown in Table 5 and a graphical comparison in Figure 3:

Semantic Results (% Identified)

Method	Top 1	Top 2	Top 3	Top 10
with	87.4	93.0	96.5	99.1
without	77.5	88.5	91.2	96.1

Table 5

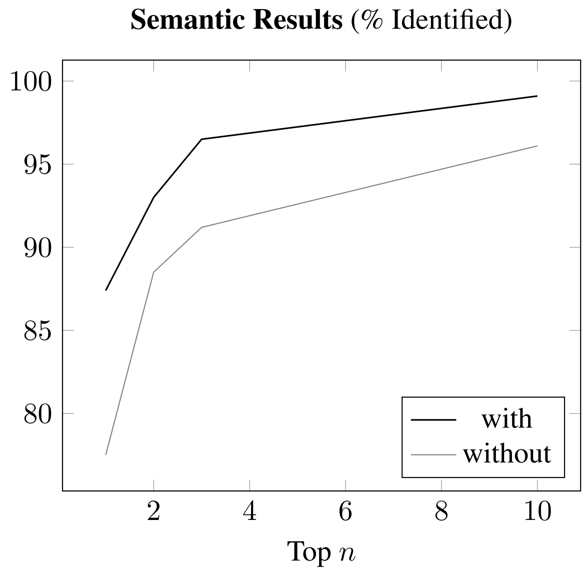

Figure 3

The runtime for WNSpell with semantic input, however, is rather slow at an average of ∼200ms.

5 Conclusions:

The WNSpell algorithm introduced in this paper presents a significant improvement in accuracy in correcting standalong spelling corrections over other systems, including the most recent version of Aspell and other commercially used spell correctors such as Huspell and Word, by approximately 20%. WNSpell is able to take into a variety of factors regarding different types of spelling errors and using a carefully tuned algorithm to account for much of the diversity in spelling errors presented in the test data sets. There is a efficient sample space pruning system that restricts the number of words to be considered, strongly improved by a phonetic key, and an accurate scoring system that then compares the words. The accuracy of WNSpell in correcting hard-to-correct words is quite close that of most peoples' abilities and significantly stronger than other methods.

WNSpell also provides an alternative using a related word to help the system find the desired correction even if the user is far off the mark in terms of spelling or phonetics. This added feature once again significantly increases the accuracy of

WNSpell by approximately 10% by directly connecting the idea word the user has in mind to the word itself. This link allows for the possibility of users who only know what rough meaning their desired word has or context it is in to actually find the word.

5.1 Limitations:

The standalone algorithm currently does not take into consideration vowel phonetics, which are rather complex in the English language. For instance, the query $spoak$ would be corrected into $speak$ rather than $spoke$. While a person easily corrects $spoak$, WNSpell would not be able to use the fact that $spoke$ sounds the same while $speak$ does not. Rather, all three have consonant sounds s, p, k and have one different letter from $spoak$. But an evaluation of edit distance finds that $speak$ is clearly closer, so the algorithm chooses $speak$ instead.

WNSpell, a spell corrector targeting at single-word queries, also does not have the benefit of contextual clues most modern spell correctors use.

5.2 Future Improvements:

As mentioned earlier, introducing a vowel phonetic system into WNSpell would increase its accuracy. The semantic feature of WNSpell can be improved by either pruning down the algorithm to improve performance or possibly using/incorporating other closeness measures of words into the algorithm. One possible addition is the use of some distributional semantics, such as using pre-trained word vectors to search for similar words (such as Word2Vec).

Additionally, WNSpell-like spell correctors can be implemented in many languages rather easily, as WNSpell does not rely very heavily on the morphology of the language (though it requires some statistics of letter frequencies as well as simplified phonetics). The portability is quite useful as WordNet is implemented in over a hundred languages, so WNSpell can be ported to other non-English WordNets.

References

D. Jurafsky and J.H. Martin. 1999. *Speech and Language Processing*, Prentice Hall.

R. Mishra and N. Kaur. 2013. “A survey of Spelling Error Detection and Correction Techniques,” *International Journal on Computer Trends and Technology*, Vol. 4, No. 3, 372-374.

K. Atkinson. “Spell Checker Test Kernel Results,” http://aspell.net/test/.

S. Deorowicz and M.G. Ciura. 2005. “Correcting Spelling Errors by Modeling their Causes,” *Int. J. Appl. Math. Comp. Sci.*, Vol. 15, No. 2, 275-285.

P. Norvig. “How to Write a Spell Corrector,” http://norvig.com/spell-correct.html.

K.W. Church and W.A. Gale. 1991. “Probability Scoring for Spelling Correction,” AT&T Bell Laboratories

M.N. Jones and J.K. Mewhort. 2004. “Case-Sensitive Letter and Bigram Frequency Counts from Large-Scale English Corpora,” *Behavior Research Methods, Instruments, & Computers*, 36(3), 388-396.

Corpus of Contemporary American English. (n.d.). http://corpus.byu.edu/coca/.