Testing the Efficacy of Part-of-Speech Information in Word Completion

Afsaneh Fazly and **Graeme Hirst**
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada M5S 3G4
{*afsaneh, gh*}*@cs.toronto.edu*

Abstract

We investigate the effect of incorporating syntactic information into a word-completion algorithm. We introduce two new algorithms that combine part-of-speech tag trigrams with word bigrams, and evaluate them with a test-bench constructed for the purpose. The results show a small but statistically significant improvement in keystroke savings for one of our algorithms over baselines that use only word *n*-grams.

1 Introduction

A *word-completion* utility facilitates the typing of text by a user with physical or cognitive disabilities. As the user enters each keystroke, the program displays a list of the most likely completions of the partially typed word. As the user continues to enter letters, the program updates the suggestion list accordingly. If the intended word is in the list, the user can select it with a single keystroke or mouse-click. For a user with physical disabilities, for whom each keystroke is an effort, this saves time and energy; for a user with cognitive disabilities, this can assist in the composition of well-formed text. A number of word-completion utilities are available commercially; but their suggestions, which are based on *n*-gram frequencies, are often syntactically or semantically implausible, excluding more-plausible but lower-frequency possibilities from the list. This can be particularly problematic for users with certain cognitive disabilities, such as dyslexia, who often are easily confused by inappropriate suggestions.

In this study, we explore the addition of syntactic information to word completion, developing new algorithms in which the part-of-speech tags of words are used in addition to the words themselves to improve the accuracy of the suggestions. We hypothesize that this will reduce the likelihood of suggesting words that are syntactically inappropriate in context and hence will result in greater savings. Details not presented here are given by Fazly (2002).

2 Related work

Early word-completion programs, mostly developed in the 1980s, were based on very simple language models. They suggest high-frequency words that match the partially typed word, and ignore all the previous context; *e.g.,* Swiffin et al. (1985). To provide more-appropriate suggestions, some programs look at a larger context by exploiting word bigram or trigram language models. *WordQ* (Nantais et al., 2001; Shein et al., 2001), developed at the Bloorview MacMillan Children's Centre, Toronto, *Profet* (Carlberger et al., 1997a, 1997b), and a program by Bentrup (1987) are statistical and adaptive programs that incorporate *n*-gram language models. These programs employ information on the user's recency and frequency of use of each word to adapt to the user's typing behaviour.

The research closest to ours here is perhaps that of VanDyke (1991), Wood (1996), and Garay-Vitoria and Gonzalez-Abascal (1997), who employed parsers, requiring a considerable amount of work to reparse the partial input sentence every time a new word is completed by the user. Carlberger et al. (1997b) incorporate part-of-speech tag information about words. They first obtain a probability estimate for the tag of the next word and then suggest words using this probability estimation and word bigram models. Copestake (1997) used the part-of-speech tag bigrams collected from a small and unbalanced corpus, along with word recency information. She compared the results with that of a simple frequency-based algorithm.

FASTY, a multilingual word-prediction system (Matiasek et al., 2002), and a prediction system for Spanish developed by Palazuelos (2001), both exploit syntactic knowledge in the form of part-of-speech tag statistics and grammar. *FASTY* uses collocation-based statistics to include long-distance relationships among words; it is mostly concerned with the prediction of nominal compounds (*e.g.,* in German). Palazuelos's system is designed for Spanish and hence direct comparison with our results below is not possible.

In this work, we introduce several new word-completion algorithms that exploit part-of-speech tag information about words, and we employ a large, balanced corpus for training. And unlike most previous work that used only one performance measure, we introduce a number of measures to evaluate and compare different algorithms in different experimental conditions.

3 Word-completion algorithms

Suppose the user is typing a sentence and the following sequence has been entered so far:

$$\ldots w_{i-2}\, w_{i-1}\, w_{i_{prefix}}$$

where w_{i-2} and w_{i-1} are the most recently completed words, and $w_{i_{prefix}}$ is a partially typed word. Let W be the set of all words in the lexicon that begin with the prefix $w_{i_{prefix}}$. A word-completion algorithm attempts to select the n most-appropriate words from W that are likely to be the user's intended word, where n is usually between 1 and 10. The general approach is to estimate the probability of each candidate word $w_i \in W$ being the user's intended word, given the context.

3.1 Baselines: Unigram, Bigram, and WordQ

The Unigram algorithm simply suggests the n highest frequency words of W. The Bigram algorithm instead uses an estimation of the bigram probability $P(w_i \,|\, w_{i-1})$ to select the most likely words for the desired sentence position. These are our baseline algorithms. In addition, we compare our results with the adaptive n-gram algorithm of WordQ.

3.2 Syntactic algorithms

We introduce two algorithms—Tags-and-Words and Linear Combination—that use part-of-speech tag information in addition to word n-grams in order to maximize the likelihood of syntactic appropriateness of the suggestions. Both algorithms assume the presence of a part-of-speech tagger that annotates words with their most likely part-of-speech tags incrementally as the user types them in. In different ways, each attempts to estimate the probability $P(w_i \,|\, w_{i-1}, t_{i-1}, t_{i-2})$, where t_j is the part-of-speech tag of word w_j. These algorithms are described in more detail in the following sections. (We also tried an algorithm that used only part-of-speech tags; its performance was inferior to that of the Bigram algorithm; see Fazly (2002).)

3.2.1 Tags and Words

The Tags-and-Words algorithm combines tag trigrams and word bigrams in a single model. This algorithm estimates the probability $P(w_i \,|\, w_{i-1}, t_{i-1}, t_{i-2})$ with the following formula:

$$\begin{aligned} & P(w_i \,|\, w_{i-1}, t_{i-1}, t_{i-2}) \\ &= \sum_{t_i \in T(w_i)} P(w_i, t_i \,|\, w_{i-1}, t_{i-1}, t_{i-2}) \\ &= \sum_{t_i \in T(w_i)} P(w_i \,|\, w_{i-1}, t_i, t_{i-1}, t_{i-2}) \\ &\qquad \times P(t_i \,|\, w_{i-1}, t_{i-1}, t_{i-2}) \\ &\approx \sum_{t_i \in T(w_i)} P(w_i \,|\, w_{i-1}, t_i) \times P(t_i \,|\, t_{i-1}, t_{i-2}) \; , \end{aligned}$$

where $T(w_i)$ is the set of possible part-of-speech tags for w_i.

The conditional independence assumptions among random variables t_{i-2}, t_{i-1}, t_i, w_{i-1}, and w_i are depicted in a Bayesian network in Figure 1. Applying these conditional independence assumptions, the desired probability can be estimated by the formula above.

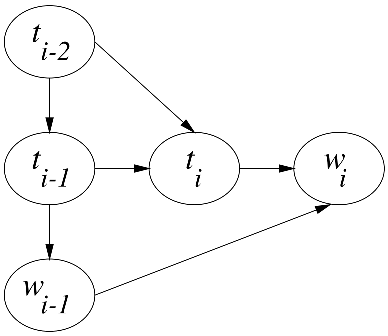

Figure 1: Bayesian network of conditional independence between words and part-of-speech tags.

To estimate $P(w_i \,|\, w_{i-1}, t_i)$, we first rewrite it using Bayes's rule:

$$
\begin{aligned}
& P(w_i \,|\, w_{i-1}, t_i) \\
& \quad = \frac{P(w_{i-1}, t_i \,|\, w_i) \times P(w_i)}{P(w_{i-1}, t_i)} \\
& \quad = \frac{P(w_{i-1} | w_i, t_i) \times P(t_i \,|\, w_i) \times P(w_i)}{P(w_{i-1}, t_i)} \\
& \quad \approx \frac{P(w_{i-1} | w_i) \times P(t_i \,|\, w_i) \times P(w_i)}{P(w_{i-1}, t_i)}
\end{aligned}
$$

$P(w_{i-1}, t_i)$ and $P(w_{i-1} | w_i)$ are then rewritten using Bayes's rule and $P(t_i \,|\, w_{i-1})$ is replaced by $P(t_i)$ assuming the conditional independence of t_i and w_{i-1} given t_{i-1}:

$$
\begin{aligned}
& P(w_i \,|\, w_{i-1}, t_i) \\
& \quad \approx \frac{P(w_i \,|\, w_{i-1}) \times P(w_{i-1}) \times P(t_i \,|\, w_i) \times P(w_i)}{P(w_i) \times P(t_i \,|\, w_{i-1}) \times P(w_{i-1})} \\
& \quad \approx \frac{P(w_i \,|\, w_{i-1}) \times P(t_i \,|\, w_i)}{P(t_i)}
\end{aligned}
$$

The likelihood of a candidate word w_i thus can be estimated by the following formula:

$$
\begin{aligned}
& P(w_i \,|\, w_{i-1}, t_{i-1}, t_{i-2}) \\
& \quad = \sum_{t_i \in T(w_i)} P(w_i, t_i \,|\, w_{i-1}, t_{i-1}, t_{i-2}) \\
& \quad \approx \sum_{t_i \in T(w_i)} \frac{P(w_i \,|\, w_{i-1}) \times P(t_i \,|\, w_i) \times P(t_i \,|\, t_{i-1}, t_{i-2})}{P(t_i)} \\
& \quad = P(w_i \,|\, w_{i-1}) \times \sum_{t_i \in T(w_i)} \frac{P(t_i \,|\, w_i) \times P(t_i \,|\, t_{i-1}, t_{i-2})}{P(t_i)}
\end{aligned}
$$

3.2.2 Linear Combination

The Linear Combination algorithm combines two models: tag trigram and word bigram. In the first model, we attempt to find the most likely part-of-speech tag for the current position according to the two previous part-of-speech tags, and then look for words that have the highest probability of being in the current position given the most likely tag for this position and the previous word. The second model is the simple Bigram model. The final probability is a weighted combination of the two models:

$$
\begin{aligned}
& P(w_i \,|\, w_{i-1}, t_{i-1}, t_{i-2}) \\
& \quad = \alpha \times P(w_i \,|\, w_{i-1}) + (1-\alpha) \\
& \qquad \times \max_{t_i \in T(w_i)} [P(w_i \,|\, t_i) \times P(t_i \,|\, t_{i-1}, t_{i-2})],
\end{aligned}
$$

where $0 \le \alpha \le 1$ is the coefficient of the linear combination that weights both factors of the probability estimation. An important issue in this approach is finding the optimal value of α, which must be determined experimentally.

4 Experimental methodology

4.1 Corpus

We used the British National Corpus World Edition (BNC) (Burnard, 2000), a corpus of English texts from different eras and different genres that has been tagged by the Constituent Likelihood Automatic Word-tagging System (CLAWS) using the C5 tagset. Our training and test data sets were disjoint subsets of the BNC, randomly selected from the written language section; they contained $5{,}585{,}192$ and $594{,}988$ words, respectively (except in a later experiment described in Section 5.6 below).

4.2 Method

We developed a generic *testbench* to perform various experiments with different word-completion algorithms in varying conditions. The testbench contains three major components: a *simulated user* that "types" in the test text; a *completion program* that can employ any of the completion algorithms to suggest words to the simulated user; and a set of *language models*, derived from the training

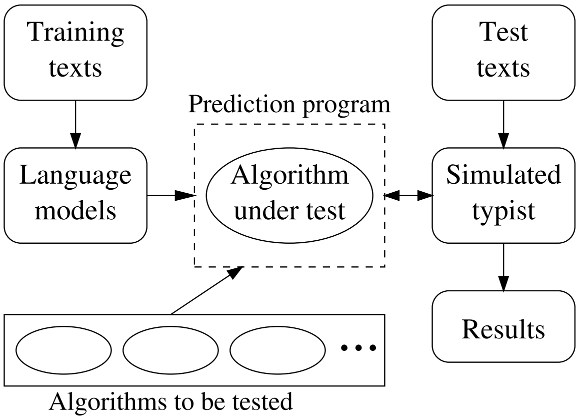

Figure 2: Architecture of the word-completion testbench used in the experiments.

corpus, for the algorithms to draw on. The architecture of the testbench is shown in Figure 2.

The simulated user is a "perfect" user who always chooses the desired word when it is available in the suggestion list. (For real users, this is not always the case, especially for people with cognitive disabilities or when the suggestion list is long.) The completion program permits any completion algorithm to be plugged in for evaluation. The *n*-gram language models, both for words and part-of-speech tags, were generated with the CMU–Cambridge Statistical Language Modeling Toolkit (Clarkson and Rosenfeld, 1997). Word-given-tag probabilities, $P(w|t)$, were collected by a Perl program written for the purpose.

4.3 Experimental conditions

In addition to the word-completion algorithms themselves, the parameters that we varied in our experiments were the following:

Coefficient α: α is the coefficient of the Linear Combination algorithm that ranges from 0 to 1, giving more weight to either factor of the probability estimation. We tried all values from 0 to 1 in 0.1 increments.

Repetition of suggestions in consecutive completions: If we assume that the user is (close to) perfect, we may avoid repeating suggestions that have previously not been accepted by the user for a particular prefix. We hypothesized that if we avoid suggesting already-unaccepted words, the likelihood of having the intended word among the new suggestions will be higher. Thus, we compared the results in both conditions: avoiding repetition, or not doing so.

Maximum number of suggestions (n): It is clear that the higher the number of words in the suggestion list, the greater the chance of having the intended word among the suggestions. But, larger values for n impose a cognitive load on the user as they make the search time for the desired word longer, and it is more likely that the user will overlook the word they are looking for. Different users of word-completion utilities may prefer different values for this parameter, according to their level and type of disabilities. We selected the values 1, 5, and 10 for n to measure how this parameter affects the performance.

4.4 Performance measures

Hit rate (HR): The percentage of times that the intended word appears in the suggestion list. A higher hit rate implies a better performance.

Keystroke savings (KS): The percentage of keystrokes that the user saves by using the word-completion utility. A higher value for keystroke savings implies a better performance.

Keystrokes until completion (KuC): The average number of keystrokes that the user enters for each word, before it appears in the suggestion list. The lower the value of this measure, the better the algorithm.

Accuracy (Acc): The percentage of words that have been successfully completed by the program before the user reached the end of the word. A good completion program is one that successfully completes words in the early stages of typing.

5 Results

5.1 Comparison of the algorithms

The completion algorithms are compared when $n = 5$ and the suggestions are allowed to be repeated in consecutive completions. The results are presented in Table 1. The value of α in the Linear Combination algorithm is 0.6 (see Section 5.2.)

The Linear Combination algorithm has the best values for all the performance measures. A paired

Algorithm	%HR	%KS	KuC	%Acc
Unigram	30.77	45.06	2.02	89.03
WordQ	34.04	49.88	1.76	89.43
Bigram	35.25	51.08	1.69	90.75
Tags+Words	35.26	51.08	1.69	90.78
Linear ($\alpha = .6$)	36.23	51.98	1.64	91.80

Table 1: Performance measures for the algorithms when $n = 5$ and suggestions may be repeated.

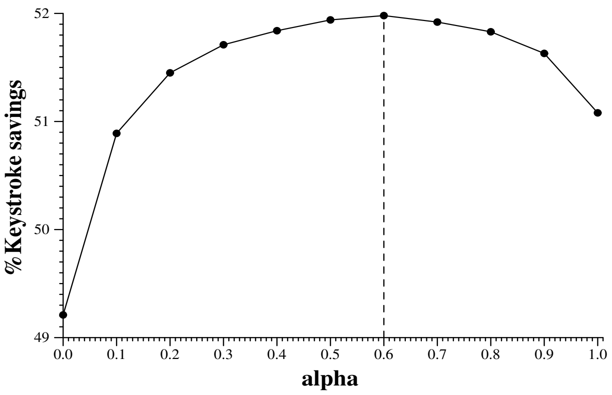

Figure 3: Keystroke savings of the Linear Combination algorithm for different values of α when $n = 5$ and suggestions may be repeated.

ANOVA test and a Tukey-Kramer post test, with confidence interval set to 95%, on the values of keystroke savings for all five algorithms showed that performance of the Linear Combination algorithm is better than that of the other algorithms and the difference is statistically significant ($p < 0.001$). It also showed that differences among keystroke savings for Unigram, WordQ, Bigram, and Linear algorithms are statistically significant ($p < 0.001$), but that between Bigram and Tags-and-Words is not ($p > 0.05$). But two different completion algorithms may have a similar performance according to our measures and yet differ in the situations in which each succeeds or fails. Thus, a more-detailed analysis of the results of these two algorithms will be given in Section 5.7.

5.2 Effect of coefficient α

An experiment with the Linear Combination algorithm was conducted to investigate the impact on performance of changing the value of α. Figure 3 shows the change in keystroke savings as α ranges from 0 to 1.

Suggestion list size	%HR	%KS	KuC
$n = 1$	21.30	34.40	2.61
$n = 5$	35.26	51.08	1.69
$n = 10$	39.69	55.90	1.43

Table 2: Effect of maximum suggestion list size on performance of the Tags-and-Words algorithm when suggestions may be repeated.

According to the experiments, the best value for α is 0.6, giving somewhat more weight to word *n*-grams than to tags. Thus, we set α to 0.6 in every other experiment with the Linear Combination algorithm. Assigning a constant value to α may not be the best approach. For some words, the PoS tag information may help more than others and thus the best approach might be to define α as a function of the word and tag *n*-gram probability distributions. Discussion of this approach is not within the scope of this work.

5.3 Effect of suggestion list size

We investigated the effect on performance of changing the maximum suggestion list size *n*. Table 2 shows the results for $n = 1$, 5, and 10 for the Tags-and-Words algorithm.

The results show that the algorithm gives more appropriate suggestions as the number of words in the suggestion list increases, resulting in higher hit rate and keystroke savings and fewer number of keystrokes needed before completion. The best value for this parameter in real-world word-completion utilities depends on the type and level of the users' disabilities. If the users have learning or cognitive disabilities, suggesting fewer number of words decreases the cognitive load imposed by the system. For users with physical disabilities, saving physical effort, *i.e.,* saving as many keystrokes as possible, is usually most important and hence suggesting more words (*e.g.,* $n = 10$) is desirable.

5.4 Effect of repeating previously suggested words

We hypothesized that if previously suggested words are not repeated, the likelihood of suggesting the appropriate word will increase. Thus, we

Repeat?	%HR	%KS	KuC
Yes	35.26	51.08	1.69
No	36.19	52.44	1.62

Table 3: Effect of prohibiting repeated suggestions on performance of the Tags-and-Words algorithm when $n = 5$.

Tagset	%HR	%KS	KuC
C5	35.26	51.08	1.69
Coarse	34.87	50.77	1.71

Table 4: Results for Tags-and-Words algorithm with fine- and coarse-grained tagsets; $n = 5$ and suggestions may be repeated.

conducted an experiment to observe the change in performance if the previously suggested words that have been rejected by the user are not suggested for the same position. For this experiment we used the Tags-and-Words algorithm. Results are shown in Table 3. Although the performance increase is small, the ANOVA test shows that the difference is statistically significant ($p < .0001$).

5.5 A coarse-grained tagset

We hypothesized that using a coarser-grained tagset than the one used to tag the BNC (C5 with 61 tags) might improve the performance. It is because some distinctions made by C5 tags might be too fine-grained to help suggest syntactically appropriate words. Thus we designed a new tagset of 28 tags that conflates many of the finer-grained distinctions in C5. For example, in the new tagset, there is no distinction among various inflected forms of the verb *be*.

We retagged the training and test data with the coarse-grained tagset, and compared the results of the Tags-and-Words algorithm on this data with those on the original data. Hit rate, keystroke savings, and keystrokes until completion for both conditions are presented in Table 4. In both experiments, $n = 5$ and suggestions may be repeated.

The results show that using the coarser-grained tagset causes a small decrease in performance. However, according to the results of ANOVA test, the difference is not statistically significant ($p > 0.05$). The reason for the decrease might be an inappropriate selection of tags in which some of the fine distinctions among syntactic categories that are necessary for the task of word completion are not considered in the new coarse-grained tagset.

Algorithm	Training data	%HR	%KS	KuC
Bigram	5.6 M	35.25	51.08	1.69
	81 M	37.43	52.90	1.55
Tags+words	5.6 M	35.26	51.08	1.69
	81 M	37.49	53.30	1.53

Table 5: Effect of training-data size on performance.

5.6 A larger training-data set

To investigate the effect of training-data size on the performance of the Bigram and Tags-and-Words algorithms, we trained them on a 14.5-times larger subset of the BNC, around 81 million words (instead of around 5.6 million words), and tested them on a one-million-word subset of the same corpus (instead of around 600,000 words). The results are shown and compared with those of the previous experiments in Table 5. There is only a small improvement in the performance when the larger training-data set is used.

5.7 Comparison of errors

It is possible that two different algorithms could have a similar performance by our measures and yet differ in the situations in which each succeeds or fails. We analyzed our results to see if this was the case.

A comparison of the Bigram and Tags-and-Words algorithms is presented in Table 6. In the table, each entry a_{ij} (the entry in row i and column j) shows the number of words that were correctly suggested after i keystrokes by the Bigram algorithm and after j keystrokes by Tags-and-Words. In particular, entries on the diagonal show the number of occasions in which the behavior of the algorithms was identical. For example, 198,359 tokens were correctly completed after one keystroke by both algorithms, whereas 505 tokens were completed by Tags-and-Words after one keystroke but by Bigram only after two keystrokes.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	206709	395	0	0	0	0	0	0	0	0	0	0	0	0	0
1	463	198359	486	14	0	0	0	0	0	0	0	0	0	0	0
2	0	505	89581	401	19	2	0	0	7	1	0	0	0	0	0
3	0	34	415	81793	385	32	0	3	1	0	0	0	0	0	0
4	0	0	34	358	44056	157	16	4	7	0	1	0	0	0	0
5	0	0	0	25	129	16598	69	10	2	0	1	1	0	0	0
6	0	0	0	9	20	42	8263	11	0	2	2	1	0	0	0
7	0	0	0	0	12	22	12	5201	10	2	0	0	0	0	0
8	0	0	3	0	1	6	6	13	3893	6	0	1	0	0	0
9	0	0	0	0	0	1	5	2	2	2964	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	2	2153	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	14	1928	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	1271	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	969	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	568

Table 6: Comparison of the performance of the Bigram and Tags-and-Words algorithms. The entry in row i and column j shows the number of words that were correctly suggested after i keystrokes by the Bigram algorithm and after j keystrokes by Tags-and-Words.

As can be seen in the table, the behavior of the algorithms was almost always identical. Out of 668,490 trials, they correctly completed the word after the same number of keystrokes 664,306 times (99.37%). On those occasions on which the algorithms differ, the difference was rarely more than one keystroke, and the algorithms were almost equally divided as to which did better: The Bigram algorithm was better by one keystroke on 1,920 (0.29%) trials and by more than one keystroke on 129 (0.02%) trials; the Tags-and-Words algorithm was better by one keystroke on 1,955 (0.29%) trials and by more than one keystroke on 180 (0.03%) trials.

6 Conclusion

We have presented and tested two algorithms that use part-of-speech information to improve the syntactic appropriateness of the suggestions in word completion. It should be noted that nothing in the algorithms relies on the particular keyboard used; reduced keyboards, such as T9 (Kushler, 1998) or Kühn and Garbe's (2001) six-key set-up, will of course give greater ambiguity in the input, but our algorithms may still be applied.

We found that the keystroke savings of the Linear Combination algorithm was significantly better than the baseline algorithms, which used only word n-grams, and the other syntactic algorithm, Tags-and-Words. Nonetheless, the improvement, while statistically significant, was not large, and might not be considered worth the considerable extra cost that it requires. For example, the Bigram algorithm was about 6.5 times faster than the Linear Combination algorithm on our test data. It is important, after all, that word-completion appear instantaneous to the user.

The relatively small improvement is possibly because word-bigram probabilities implicitly capture tag-bigram probabilities to a considerable extent as well. Thus, there might be a high overlap between the information that is used by the word-based and the tag-based models. This result is consistent with that of Garay-Vitoria and Gonzalez-Abascal (1997), who found that the use of probabilistic grammar rules in word completion gave only small improvements (which they did not test for statistical significance.) The result is also consistent with that of Banko and Brill (2001), who showed that in confusion-set disambiguation, having a very large training corpus is more efficacious than having a smarter and stronger classifier.

Conditional independence assumptions Both syntactic algorithms make independence assumptions in order to calculate the probability $P(w_i \mid w_{i-1}, t_{i-1}, t_{i-2})$ using separate n-gram models of words and PoS tags. For example, in estimating the above probability, it is assumed that t_i and w_{i-1} are conditionally independent given the tag of the previous word, t_{i-1}. Although this is a reasonable assumption to make, it is not clear how it may affect the results. Without making this independence assumption, it would have been necessary to calculate the probabilities $P(t_i \mid w_{i-1})$ and $P(w_{i-1} \mid w_i, t_i)$ directly from the corpus.

Therefore, employing hybrid n-gram models for words and tags might result in more accurate syntactic algorithms. For example, an algorithm could estimate the joint probability of having a word w_i and its part-of-speech tag t_i in the current position, given the previous words and their associated part-of-speech tags, $P(w_i, t_i \mid w_{i-1}, t_{i-1}, w_{i-2}, t_{i-2})$, directly from the corpus as a hybrid n-gram model.

Errors from the CLAWS tagger Inaccuracy in tagging might also be a factor. The BNC's tagging has a 3–4% error rate, and many "ambiguous" tags. This means that although tags add a new source of information to the completion algorithm, more uncertainty is introduced as well.

Future work So far, we have carried out only performance measures on our algorithms, and more evaluation remains to be undertaken in order to better understand their abilities and limitations. For example, we have not yet assessed the improvement (if any) in syntactic correctness of suggestions that our algorithms actually afford, nor have we conducted any user studies. In the next phase of the work, we will add criteria for semantic coherence to the algorithms, adapting measures of semantic similarity (Budanitsky, 1999) to the task.

Acknowledgments

This research was supported by a grant from Communications and Information Technology Ontario. We are grateful to Fraser Shein and Tom Nantais of the Bloorview MacMillan Children's Centre, Toronto, for advice and assistance in all stages of this research.

References

Michele Banko and Eric Brill. 2001. Scaling to very very large corpora for natural language disambiguation. *Proceedings, 39th Annual Meeting, Association for Computational Linguistics*, 26–33, Toulouse.

John A. Bentrup. 1987. Exploiting word frequencies and their sequential dependencies. *Proceedings, 10th Annual Conf. on Rehabilitation Technology*, 121–123. RESNA.

Alexander Budanitsky. 1999. Lexical semantic relatedness and its application in natural language processing. Technical report CSRG-390, Dept of Computer Science, University of Toronto. *www.cs.toronto.edu/compling/Publications/*

Lou Burnard, 2000. *Reference Guide for the British National Corpus (World Edition)*. *www.hcu.ox.ac.uk/BNC*, 2nd edition.

Alice Carlberger, Johan Carlberger, Tina Magnuson, Sharon Hunnicutt, Sira E. Palazuelos-Cagigas, and Santiago Aguilera Navarro. 1997a. Profet, a new generation of word prediction: An evaluation study. *Proceedings,ACL Workshop on Natural language processing for communication aids*, 23–28, Madrid.

Alice Carlberger, Tina Magnuson, Johan Carlberger, Henrik Wachtmeister, and Sheri Hunnicutt. 1997b. Probability-based word prediction for writing suport in dyslexia. *Proceedings of Fonetik'97 Conference*, vol. 4, 17–20.

Philip Clarkson and Ronald Rosenfeld. 1997. Statistical language modeling using the CMU–Cambridge toolkit. In *Proceedings of European Conference on Speech Communication and Technology (EUROSPEECH)*, 2707–2710, Rhodes, Greece.

Ann Copestake. 1997. Augmented and alternative nlp techniques for augmentative and alternative communication. In *In Proceedings of the ACL workshop on Natural Language Processing for Communication Aids*, 37–42, Madrid.

Afsaneh Fazly. 2002. Word prediction as a writing aid for the disabled. Master's thesis, Dept of Computer Science, University of Toronto, January. *www.cs.toronto.edu/compling/Publications/*

Nestor Garay-Vitoria and Julio Gonzalez-Abascal. 1997. Intelligent word prediction to enhance text input rate (a syntactic analysis-based word prediction aid for people with severe motor and speech disability). In *Proceedings of the Annual International Conference on Intelligent User Interfaces*, 241–244.

Michael Kühn and Jörn Garbe. 2001. Predictive and highly ambiguous typing for a severely speech and motion impaired user. In Constantine Stephanidis, editor, *Universal Access in Human–Computer Interaction (Proceedings of UAHCI-2001)*. Lawrence Erlbaum Associates.

Cliff Kushler. 1998. AAC using a reduced keyboard. In *Proceedings of CSUN 13th Annual Conference on Technology for Persons with Disabilities*.

Johannes Matiasek, Marco Baroni, and Harald Trost. 2002. FASTY — A multi-lingual approach to text prediction. In K. Miesenberger et al (eds.) *Computers Helping People wih Special Needs*, Springer-Verlag.

Tom Nantais, Fraser Shein, and Mattias Johansson. 2001. Efficacy of the word prediction algorithm in WordQ. In *Proceedings of the 24th Annual Conference on Technology and Disability*. RESNA.

Sira E. Palazuelos Cagigas. 2001. *Contribution to Word Prediction in Spanish and its Integration in Technical Aids for People with Physical Disabilities.* PhD thesis, Universidad Politécnica de Madrid.

Fraser Shein, Tom Nantais, Rose Nishiyama, Cynthia Tam, and Paul Marshall. 2001. Word cueing for persons with writing difficulties: WordQ. In *Proceedings of CSUN 16th Annual Conference on Technology for Persons with Disabilities*.

Andrew L. Swiffin, J. Adrian Pickering, John L. Arnott, and Alan F. Newell. 1985. Pal: An effort efficient portable communication aid and keyboard emulator. In *Proceedings of the 8th Annual Conference on Rehabilitation Technology*, 197–199. RESNA.

Julie A. VanDyke. 1991. A syntactic predictor to enhance communication for disabled users. Technical Report 92-03, Dept of Computer and Information Sciences, University of Delaware.

Matthew E.J. Wood. 1996. *Syntactic pre-processing in single-word prediction for disabled people*. Ph.D. thesis, Dept of Computer Science, University of Bristol.