MAP-Based Error Correction Mechanism for Five-Key Chording Keyboards

Adrian Tarniceriu, Bixio Rimoldi, Pierre Dillenbourg
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Lausanne, Switzerland
e-mail: adrian.tarniceriu@epfl.ch, bixio.rimoldi@epfl.ch, pierre.dillenbourg@epfl.ch

***Abstract*—Because of different designs, different text input devices have different error patterns. If we consider these aspects when designing an error correction mechanism, we can obtain significantly lower error rates. In this paper, we propose and evaluate a spelling algorithm specifically designed for a five-key chording keyboard. It is based on the maximum a posteriori probability (MAP) criterion, taking into account a dictionary model and the probability that one character is typed for another. These probabilities are determined experimentally. For the considered evaluation text, the proposed method reduced the error rate from 10.11% to 2.17%. As comparison, MsWord and iSpell reduced the error rate to 5.15% and 6.69%, respectively.**

***Keywords*-error correction; chording keyboard; maximum a posteriori probability; confusion matrix**

I. Introduction

Most of us use mobile computing devices, such as smartphones or tablets, and would like to use them even more, but there are situations when we cannot easily access their services. For example, when walking in a crowded place, we should focus on what happens around us rather than on the mobile device. This limitation in usability is mainly due to the input interface, which requires visual commitment.

A way to reduce visual constraints while typing is given by chording keyboards. This type of keyboard enables users to generate a character by simultaneously pressing a combination of keys, similarly to playing a note or a chord on a musical instrument. For a keyboard with five keys, there are 31 combinations in which at least one key is pressed. This is enough for the 26 letters of the English alphabet and five other characters. If the keys are in a position that is naturally under the fingertips, a person can type using the fingers of one hand, without committing the eyes to the keyboard. Therefore, we will be able to use a mobile device even during activities for which vision is partially or entirely committed, such as walking in crowded spaces, jogging, or riding a bike (for example, we can place the keys around the phone, or on a bike handlebar).

The reason why chording keyboards are not popular is that they require training. Compared to a QWERTY keyboard, where users can "hunt and peck" from the beginning, for chording keyboards the mapping between keys and characters has to be learned before being able to type. The effort needed to do so depends on the keyboard type and mapping and can vary by several hours. In previous studies [1], we described a five-key chording keyboard and a key-to-character mapping that can be learned in less than 45 minutes. After 350 minutes of practice, the average typing rate was around 20 words per minute (wpm) with a maximum of 31.7 wpm, comparable to iPhone, Twiddler [2], or handwriting [3]. Without correcting the mistakes, the average character error rate at the end of the studies was 2.69%. Automatically correcting these mistakes will probably increase the keyboard's ease of-use and typing speed, because users will not have to stop typing in order to correct errors. In addition, typing in a dynamic environment will probably lead to more errors, so efficient error correction becomes even more important in these situations.

In this paper, we continue our work on error correction mechanisms for five-key chording keyboards [4][5]. Whereas our previous work only considered substitution errors (when a character is replaced by another character), now we will also consider errors such as deletions (when a character is omitted), insertions (when an additional character is inserted), or split or merged words. The correction mechanism is based on the maximum a posteriori probability (MAP) principle [6] and for each typed word, it provides a list of possible candidates and chooses the one that is the most likely. Moreover, it takes into consideration the particularities of the text input device. This is motivated by the fact that different devices lead to different error patterns, and knowledge about these patterns can be used to improve the error correction methods.

The paper is organized as follows. Section II presents a brief overview of existing text error correction mechanisms. In Sections III and IV, we describe the proposed error correction algorithm and the data set used for evaluation. Section V presents the error correction results. In Section VI, we conclude the paper.

II. Related Work

Work on automatically correcting misspelled words in computer-typed text began in the 1960s [7] and the algorithms' efficiency has steadily increased since then. Nevertheless, the correction rates are still far below 100% and improving them remains a challenge.

Traditionally, error detection and correction mechanisms functioned at word level. Non-words are identified in a typed text and the most likely corresponding words are suggested from a dictionary. The appearing errors are defined at character level and can be classified into three categories: deletions, when a character is omitted; insertions, when an additional character is inserted; substitutions, when a character is substituted by another character. Other, more complex, approaches take into account the context, grammatical and semantical rules, and also detect errors such as missing words, wrong phrase structure, misused inflections, or others.

A detailed overview of the commonly used correction techniques is presented by Kukich in [8]. Research in spelling error detection and correction is grouped in three main categories:

1) Non-word error detection:
   Groups of *n* letters (*n*-grams) are examined and looked up in a table of statistics. The strings that contain non-existing or highly infrequent *n*-grams are considered errors.
2) Isolated-word error correction:
   Each word is treated individually and considered either correct or incorrect. In the latter case, the incorrectly spelled word is compared to entries from a dictionary. Based on similarities between the typed word and dictionary words, a list of possible candidates is proposed. These candidates can be provided using several techniques:
   - minimum edit distance techniques consider the minimum number of editing operations required to transform a string into another. A basic example is to consider the dictionary word that can be obtained from the typed word with a minimum number of insertions, deletions, and substitutions;
   - similarity key techniques map each string to a key which is similar or identical for similarly spelled strings. In this way, the key for a misspelled string can point to similarly spelled candidates from the dictionary. The advantage of this approach is that the misspelled string is not compared to all entries in the dictionary;
   - rule-based techniques propose candidate words by using knowledge of the most common errors;
   - probabilistic techniques, which consider transition and confusion probabilities. The first ones provide the probability that a letter is followed by another given letter (the values are language dependent). Confusion probabilities estimate how often a letter is typed instead of another letter (the values are text-input device dependent);
   - among other possible methods, *n*-gram techniques and neural net techniques can also be efficiently used.

   Most isolated word error correction methods do not correct errors when the erroneously typed word is in the dictionary. For example, if *farm* is typed instead of *form*, no error will be detected. Moreover, these methods cannot detect the use of wrongly inflected words (for example, *they is* instead of *they are*).
3) Context-dependent error correction:
   These methods try to overcome the drawbacks of analyzing each word individually by also considering the context. Errors can be detected by parsing the text and identifying incorrect part-of-speech or part-of-sentence *n*-grams. Or, if enough memory and processing power are available, tables of word *n*-grams can be used. Other approaches consider grammatical and inflectional rules, semantical context, and can also identify stylistic errors.

Most of the methods presented above can be applied to any typed text, regardless of the input device. As various input techniques become more and more popular, the classic correction techniques have been improved to consider both the text and the device particularities. Goodman et al. [9] presented an algorithm for soft keyboards that combines a language model and the probabilities that the user hits a key outside the boundaries of the desired key. Kristensson and Zhai [10] proposed an error correction technique for stylus typing using geometric pattern matching. The T9 text input method for mobile phones can also be included here, as it considers the correspondence between keys and characters to predict words.

An error correction algorithm for chording keyboards is presented by Sandnes and Huang [11]. Firstly, they classify chording errors in three categories similar to the character errors: deletions, when the user does not press one of the required keys, insertions, when the user presses an extra key, and substitutions, when the user makes a mistake between adjacent fingers. Then, starting from the assumption that most words have very few errors, they describe an algorithm that can correct words that contain one deletion, insertion, or substitution.

III. Algorithm

The algorithm that we propose focuses on individual errors, without considering any contextual information, and is based on the maximum a posteriori probability principle. As we designed it to correct errors from text typed with a five-key chording keyboard, we will name it 5keys-MAP. The first part of the algorithm was presented in our previous work, but, to make this paper self-contained, we will also present it in the following.

A. MAP Algorithm

The starting point is the noisy channel approach [12], where the typing process is seen as sending information over a communication channel. The symbol at the channel input, x, is the word to be typed and the channel output, y, is what has actually been typed (Figure 1).

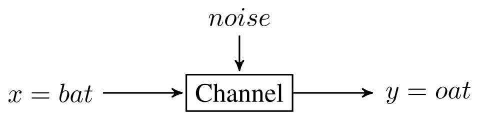

Figure 1. Typing seen as a sending information over a noisy channel

The MAP algorithm will find the string $\hat{x}$, which is the most likely in the sense of maximizing the posterior probability $p(x|y)$ over all $x \in \mathcal{S}$. The set $\mathcal{S}$ contains all the possible candidate strings. If we denote by $p(x)$ and $p(y)$ the distributions for the channel input and output, respectively, then

$$\begin{aligned} \hat{x} &= \operatorname*{argmax}_{x\in\mathcal{S}} p(x|y) && (1)\\ &= \operatorname*{argmax}_{x\in\mathcal{S}} \frac{p(y|x)p(x)}{p(y)} && (2)\\ &= \operatorname*{argmax}_{x\in\mathcal{S}} p(y|x)p(x), && (3) \end{aligned}$$

where (2) follows from Bayes' rule.

Because our goal is to design a spelling algorithm, we can reduce the set of candidates from all possible strings to dictionary words. Then, assuming that the typing of each letter depends only on the intended letter and not on previous or successive letters, we can write

$$p(y|x) = \prod_{i=1}^{i=N} p(y_i|x_i)\,, \qquad (4)$$

where y_i it the ith letter of the typed word, x_i is the intended letter, and N is the word length. The conditional probability $p(y_i|x_i)$ is the probability that the character y_i is typed in lieu of x_i. The prior probability, $p(x)$, is given by the frequencies of the dictionary entries in English language.

For example, given the typed word $y = oat$ and the candidate $x = bat$, we need to compute the posterior probability $p(bat|oat)$. This is proportional the product between the likelihood $p(oat|bat)$ and the prior probability $p(bat)$. We will denote this product by $F(oat|bat)$:

$$\begin{aligned} F(oat|bat) &= p(oat|bat)p(bat) && (5)\\ &= p(o|b)p(a|a)p(t|t)p(bat). && (6) \end{aligned}$$

The prior probabilities, $p(x)$, were obtained from the British National Corpus, containing approximately 100 million words [13]. From this set, we only considered the items that appeared at least five times, obtaining a dictionary with $100\,944$ entries (including inflected forms, such as declensions and conjugations). The confusion probabilities, $p(y_i|x_i)$, were estimated experimentally.

The method described so far is the same as in our previous work on error correction, and can be applied to substitution errors, when the intended and the typed words have the same length. In the following, we will extend it to also consider error types such as missing or extra characters, or concatenated or split words. For this, besides the prior and confusion probabilities, we will use the probabilities that a letter is added or deleted from a word, the probability that a space character is added, and the probability that a space character between words is deleted. These values will be estimated experimentally from the same data set as the confusion probabilities.

B. Error Types

Before explaining the algorithm, it is useful to enumerate the errors that we will consider.

- Substitutions, when one letter is replaced by another (e.g., *hous**a*** instead of *hous**e***).
- Additions, when an extra letter is added to a word (e.g., *hous**a**e* instead of *house*). The probability to add a letter to a word is denoted as p_{Add}.
- Deletions, when a letter is missing from a word (e.g., *hous* instead of *hous**e***). The probability to delete a letter is denoted as p_{Del}.
- Extra space, when a word is split by an added space character (e.g., *hou se* instead of *house*). The corresponding probability is denoted as p_{SpAdd}.
- Missing space, when the space between consecutive words is missing (e.g., *thehouse* instead of *the house*). The corresponding probability is denoted as p_{SpDel}.
- Replacing a letter by a space (e.g., *ho se* instead of *house*).
- Replacing a space by a letter (e.g., *theohouse* instead of *the house*).
- Any combination of two of the above-mentioned errors.

C. Error Correction Algorithm

For every typed word, where by typed word we mean a set of letters separated by space characters, we will consider as candidates the words or bigrams obtained through the above mentioned operations. Then, we will determine the posterior probabilities using the MAP rule, and choose the most likely candidate. In case of substitutions, the algorithm is the same as in the previous subsection. For other error types, we also use the probabilities of the corresponding operations. In (8) and (10), we provide two examples for computing the posterior probabilities. As now we also analyze sets of two words, we need to know word bigram frequencies. These will also be estimated from the British National Corpus.

If we want to consider split words too, we have to go one step further and analyze groups of two consecutively typed words. For example, if the two words are *dictio* and *nary*, a candidate will be *dictionary*, and the posterior probability is given in (12).

The algorithm can be summarized in three steps, enumerated below. To make things clearer, we also provide an example, when the typed text is *"the dict ionary istoo heavty"*, and the intended text is *"the dictionary is too heavy"*.

1) Analyze each individual word:
   For each of *the*, *dict*, *ionary*, *istoo*, and *heavty*, we find the most likely candidates, named individual candidates, and the posterior probabilities. The results are given in lines 2 and 3 of Table I. We mention that the posterior probabilities from the table have been scaled to avoid working with very small numbers.

TABLE I. THE MOST LIKELY CANDIDATES AND THE POSTERIOR PROBABILITIES FOR EACH TYPED WORD AND FOR GROUPS OF TWO TYPED WORDS

Typed word	*the*	*dict*	*ionary*	*istoo*	*heavty*
Individual candidate	***the***	*diet*	*i nary*	***is too***	***heavy***
Posterior probability	99.99	0.66	0.01	0.86	0.45
Split candidate	*theodicity*		-		
Posterior probability	0.001		-		
Split candidate		***dictionary***		-	
Posterior probability		0.04		-	

2) Analyze groups of two consecutive words:
   At this step, we take groups of two consecutive words and check if they can be part of a split original word. The possible candidates (named split candidates) and the corresponding probabilities are provided in lines 4 - 7 of Table I. A "-" sign means that there were no candidates.
3) Decide if a word was split:
   The last step is to decide between the candidates from the previous steps, by comparing the individual and split probabilities from Table I. If the probability of the split candidate is higher than at least one of the individual probabilities, we decide that a word was split. The probability for *theodicity* is smaller than both the probabilities for *the* and *diet*, so we decide that there was no split. However, the probability for *dictionary* is higher then the probability for *i nary*, so we decide that for the typed bigram *dict ionary*, the intended word was *dictionary*. The candidates in bold font from the table are the most likely.
   Assume now that the probability of the individual candidate *diet* is 0.0001. This is lower than the probability for *theodicity*, but the typed word *dict* cannot be part of two split words. To solve this, we will assign it to the split candidate with higher probability, in our case this being *dictionary*.

IV. EVALUATION DATA

In order to gather enough data to evaluate the proposed algorithm, we asked 10 students from our university to type using a chording keyboard prototype. The prototype has the keys placed around a computer mouse and is presented in Figure 2. We designed the prototype in this way because we wanted the subjects to see a practical application of a chording device: allowing typing and screen navigation at the same time, with only one hand. The buttons are placed so that they can be easily operated while holding the mouse with the palm. The keyboard is designed using an Arduino Pro Mini microcontroller board and communicates with the computer by Bluetooth.

The participants were asked to type for 10 sessions of 30 minutes each, while sitting at a desk. Each session consisted of three rounds of 10 minutes, separated by breaks of two minutes. In the beginning of each round, the participants warmed up by typing each letter of the alphabet. During the warm-up, a help image showing the key combination for the letter to be typed was displayed. Afterwards, the help image was no longer available and the participants typed sentences chosen from a set considered representative for the English language [14]. These sentences were pre-prepared before the experiment to contain only small letters and no punctuation

$$F(housse|house) = p(housse|house)p(house) \tag{7}$$
$$= p(h|h)p(o|o)p(u|u)p(s|s)p_{Add}p(e|e)p(house) \tag{8}$$

$$F(thedob|the\ dog) = p(thedob|the\ dog)p(the\ dog) \tag{9}$$
$$= p(t|t)p(h|h)p(e|e)p_{SpDel}p(d|d)p(o|o)p(b|g)p(the\ dog) \tag{10}$$

$$F(dictio\ nary|dictionary) = p(dictio\ nary|dictionary)p(dictionary) \tag{11}$$
$$= p(dictio|dictio)p_{SpAdd}p(nary|nary)p(dictionary) \tag{12}$$

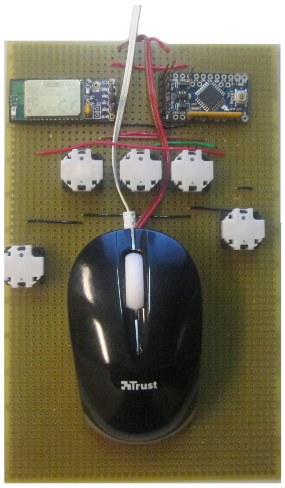
Figure 2. Chording keyboard prototype used during the typing study

signs (for example, *"February is the shortest month."* was changed to *"february is the shortest month"*).

A Java application was designed to display the text to be typed and to monitor the pressed keys. A screenshot of the application is shown in Figure 3. The top-left window contains the text to be typed and the bottom-left window represents the typing area. The help image is displayed on the right.

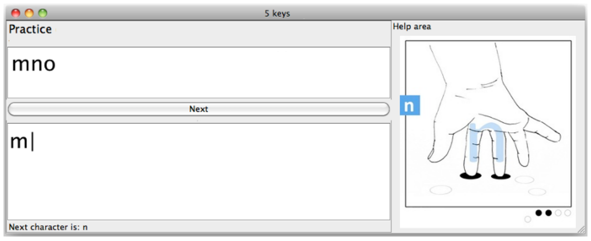

Figure 3. The interface of the Java application used during the study

Because we wanted to evaluate an error correction mechanism, we instructed the participants not to correct their mistakes (however, this was not enforced and they could delete typed text). As a reward for the time commitment during the experiment, they received a fixed monetary compensation for the first nine typing sessions. To provide additional motivation, for the last session, the reward was proportional to the number of typed words and to the typing accuracy.

The total amount of data gathered during the experiment consists of 40 640 words, of which 4109 (10.11%) contain errors. Of these, 3120 (75.93%) are substitution errors. The remaining 989 errors occurred when people did not type a letter (e.g., *hous* instead of *house*), typed an extra letter (*house**e*** instead of *house*), the space between words was missing (*thehouse* instead of *the house*), or when whole words were missing, added, or the topic of the sentence changed.

The total number of typed characters is 220 910, from which 6428 (2.91%) are errors. We used these characters to determine the confusion matrix, which is a square matrix with rows and columns labeled with all the characters that can be typed. The value at position ij shows the frequency of character j being typed when i was intended. The values are given as percentages from the total number of occurrences for character i and represent the confusion probabilities used by the algorithm.

V. Correction Results

The error correction mechanism was implemented in MATLAB and Python. To avoid overfitting, we used 10-fold cross-validation when determining the confusion matrix and evaluating the algorithm. As references, we used MsWord and iSpell. For each typed word, these algorithms return an ordered list of candidates. We considered the first one, which is the most likely, as the correction result. In addition, we compared the results with those from our previous work, which only focuses on substitution errors.

The results for both all-error-types and substitution-only scenarios are shown in Figure 4. The error rate after applying the algorithm is 2.17%, and, as expected, is lower than when considering only substitution errors (3.69%). For MsWord, considering all error types does not bring such a big improvement, the error rates being 5.15% and 5.57%, respectively. It is surprising that when we consider more error types, the error rates increase for iSpell. The most probable explanation is that for each typed word there are more candidates now, some of different lengths, and it is more difficult to choose the correct one. Or that more correctly typed words are changed by this algorithm.

In Table II, we show the initial error distribution and how many errors of each type were not corrected by the algorithm. It can be noticed that the correction method is less efficient for deletions, when a word has been split, and for combined error types. In the case of deletions, the typed

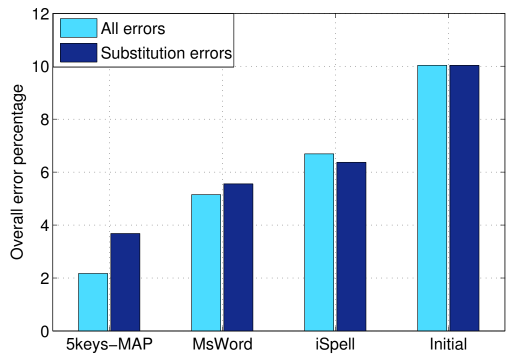

Figure 4. Overall error rates for the 5keys-MAP, MsWord, and iSpell algorithms, when considering all error types and only substitution errors, respectively

TABLE II. ERROR DISTRIBUTION FOR THE EVALUATION TEXT, BEFORE AND AFTER APPLYING THE CORRECTION ALGORITHM

Error type	Substitutions	Additions	Deletions	Extra space	Missing space
Before	3120	248	187	12	93
After	522	17	77	8	14
Error type	Letter → space	Space → letter	Combined	Other	Added errors
Before	134	156	130	29	
After	66	6	69	29	68

word length becomes smaller, and shorter words are usually more difficult to correct [5]. The same reasoning can be applied in the case of split words: the two components are considered as individual words, and their length is obviously smaller than the length of the correct word.

The results of the 5keys-MAP algorithm are clearly better than for MsWord and for iSpell. However, one should not forget that the dictionaries used by the three methods are not the same, and this can affect the results. Moreover, our algorithm is specifically designed for a five key chording keyboard, while MsWord and iSpell can be applied to any text input device with the same results.

VI. CONCLUSION

In this paper, we presented a MAP-based error correction mechanism for five-key chording keyboards. It is an isolated-word correction method, focusing on individual words without considering the context.

For the evaluation text, the algorithm reduced the error rate from 10.11% to 2.17%. This is more than two times lower than for MsWord (5.15%) and more than three times lower than for iSpell (6.69%). This advantage is due to the MAP algorithm, which takes into account the prior distribution of words and the device-dependent confusion probabilities.

The comparison between our algorithm, MsWord, and iSpell was done by only analyzing the first proposed candidate. We chose this approach because one possible use of chording keyboards is in dynamic environments, such as walking in crowded places or riding a bike, when users cannot continuously look at the typed text. Therefore, the error correction mechanism should run automatically, without requiring user supervision. In more static situations (for example when the keys are placed around a computer mouse), the most likely candidates can be displayed and the user will choose the desired one.

The correction algorithm was designed for a specific keyboard and mapping, but can be easily adapted to other input devices by updating the confusion matrix. One possibility to improve the algorithm is to implement an adaptive approach, starting with a general confusion matrix and update it based on what one types. Words that are typed more often can have their prior probability increased, becoming more likely than other candidates. Another natural improvement is to consider the typing context.

REFERENCES

[1] A. Tarniceriu, P. Dillenbourg, and B. Rimoldi, “Single-handed eyes-free chord typing: A text-entry study,” International Journal On Advances in Intelligent Systems, vol. 7, no. 1,2, pp. 145–155, 2014.

[2] K. Lyons et al., “Twiddler typing: one-handed chording text entry for mobile phones,” Proc. of the SIGCHI Conference on Human Factors in Computing Systems (CHI ’04), Vienna, Austria: ACM, 2004, pp. 671–678.

[3] I. S. MacKenzie and R. W. Soukoreff, “Text Entry for Mobile Computing: Models and Methods, Theory and Practice,” Human-Computer Interaction, vol. 17, pp. 147–198, 2002.

[4] A. Tarniceriu, B. Rimoldi, and P. Dillenbourg, “Error correction mechanism for five-key chording keyboards,” The 7th International Conference on Speech Technology and Human-Computer Dialogue (SpeD ’13), Cluj-Napoca, Romania, October 2013.

[5] A. Tarniceriu, B. Rimoldi, and P. Dillenbourg, “Fine-tunning a map error correction algorithm for five-key chording keyboards,” ECUMICT 2014, Lecture Notes in Electrical Engineering, L. De Strycker, Ed. Springer International Publishing, vol. 302, pp. 153–165, 2014.

[6] S. Kay, Fundamentals of statistical signal processing: estimation theory. Prentice-Hall, 1993.

[7] C. R. Blair, “A program for correcting spelling errors,” Information and Control, vol. 3, no. 1, pp. 60 – 67, 1960.

[8] K. Kukich, “Techniques for automatically correcting words in text,” ACM Comput. Surv., vol. 24, pp. 377–439, 1992.

[9] J. Goodman, G. Venolia, K. Steury, and C. Parker, “Language modeling for soft keyboards,” Proc. of the 7th International Conference on Intelligent User Interfaces (IUI ’02), New York, NY, USA: ACM, 2002, pp. 194–195.

[10] P.-O. Kristensson and S. Zhai, “Relaxing stylus typing precision by geometric pattern matching,” Proc. of the 10th International Conference on Intelligent User Interfaces (IUI ’05), New York, NY, USA: ACM, 2005, pp. 151–158.

[11] F. Sandnes and Y.-P. Huang, “Non-intrusive error-correction of text input chords: a language model approach,” Annual Meeting of the North American Fuzzy Information Processing Society, 2005 (NAFIPS 2005), June 2005, pp. 373 – 378.

[12] M. D. Kernighan, K. W. Church, and W. A. Gale, “A spelling correction program based on a noisy channel model,” Proc. of the 13th Conference on Computational Linguistics - Volume 2 (COLING ’90), Stroudsburg, PA, USA: Association for Computational Linguistics, 1990, pp. 205–210.

[13] URL: http://www.kilgarriff.co.uk/bnc-readme.html [accessed: 2015-03-10].

[14] I. S. Mackenzie and R. W. Soukoreff, “Phrase sets for evaluating text entry techniques,” Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI ’03), Fort Lauderdale, Florida, United States: ACM, 2003, pp. 766–767.