HMS2005: Predictive text entry using bigrams

Myrtille Dedianne and Robert Nilsson

17th January 2005

Abstract

Nowadays, hundred millions of SMS are sent everyday all around the word and become common in our everyday-life. Then, the efficiency of text entry method in mobile phones is more and more important. As a previous team project already worked on, in order to improve it by using bigram prediction, we decided to continue their work and improve it. In this paper, we describe the system, which is called HMS[3], and how we improved it. This improved version will be called HMS2005. Firstly, the code was reviewed to make it work in English or in whatever language. Secondly, the code was reviewed to make it work quicker and usable. Finally, we added a key, which stops the prediction and falls back to T9 when typing. For the tests, we involved 7 international persons and measured the time and the number of key pressed needed to entry a text. We showed that keys pressed needed were reduced of 20%, but time-consuming was increased of 15%. However, we noticed a difference between people who were or not trained by the new system, which can false the final results. This could be measure in good conditions of a real experience.

1 Introduction

To enter SMS with our mobile phones, we use to use methods we are offered, like T9. But these are not perfect yet and can be optimized to be more usable, more flexible, more efficient, easier to learn, quicker type a SMS message, with less stress etc.

We chose to work on this subject in order to improve the way to write SMS messages. A previous team of 3 Swedish students (Hasselgren, Montnemery, Svensson) have had already worked on it in this course and their system was named HMS [3]. We studied how they built it and how to improve it. Our improved version is called HMS2005 and can be found as an applet at this address: `http://www.orbstation.com/hms2005`.

After, we analysed the different methods already existing and began to code.

2 Evolution of different methods

With a keyboard of 12 keys, we can measure the efficiency of different methods using the number of keystrokes per character or KSPC[4] and the time-consuming to entry the text.

Since the beginning of SMS messages, several predictive text entry methods were developed, in order to improve the efficiency of the multi-press method. In this method, the user has to press once a key to type the first letter of the key, twice for the second, three times for the third et cetera. It is still used but requires more than one of keystroke per character and takes time. We will present 3 other predictive text entry methods.

2.1 T9, by Tegic

This method is a single-press method using unigrams. The user presses once key per character and the program matches the sequence to words in a dictionary. In many cases, only one word is possible given the sequence, otherwise, a list with other possibilities is suggested and the user chooses. The KSPC is then reduced roughly to 1 and is less time-consuming, whereas the beginning of using it is quite disturbing. Many mobile phones use this method nowadays.

But other implementations are iTAP by Motorola[2] and eZiText by Zi Corporation[1], which suggest the next word you intend to type.

2.2 HMS (for Swedish), Lund Institute of Technology, Sweden

This method is a single-press method using unigrams and bigrams, i.e. two consecutive words[3]. It was developed by 3 students of LTH, Lund (Sweden), and uses context. Typing a sequence of keys, the system considers the previous constant word typed, matches it in the dictionary of bigrams (which gives the most frequent words which can follow this word), and gives in real-time the entire word it could be. In this implementation, the bigrams are always prioritized over the unigrams. Available for Swedish, this method reduced KSPC of 7% on SMS messages and 13% on News[3].

2.3 Other methods

There are other methods, like LetterWise[5] or Less-tap[6].

LetterWise is a system that does not use a stored dictionary of words, but a small database of prefix information to disambiguate user keystrokes (Eatoni Corporation, 2003). Its published KSPC is 1.1500[5].

Less-tap method uses a remapped keyboard as a complement for single-press or multi-press methods. Since the keyboard is built on the alphabetical order, it does not take in account the frequency of the most common characters used in different languages. For example in English, "e" is the most common character used, but is in the second place on his key and shares it with "d" and "f". The keyboard could be remapped, mixing all characters and their order, to reduce the KSPC required to 1.4412[6]. However, this proposal could be difficult to be accepted by people who use to use the actual keyboard.

3 Dictionary and corpus

3.1 Dictionary compilations

Our first aim was to make it work in English, and independent of the language. To achieve this it was important that the data files were stored in an internationalizable and platform independent way. Hence we, quite naturally, chose to use Unicode. After that we collected a dictionary of English unigrams. We could have chosen between 2 dictionaries available on the Oxford's documents which are publically available. The first was small (254 kb, about 27000 words) and contained most common abbreviations, places and names, but did not contain all the inflected words. The second was big (3 Mb, 10 millions words), and was a mixed file of all Moby's dictionaries available, with all inflected words, abbreviations, places and names. The first was too small to be used, not enough complete, and the second was too big, requires too much memory and given too many not common words. So we decided to use one between both as a compromise. We found a dictionary on the web (UK English wordlist v1.01, from the website of Brian Kelk, Cambridge, UK: `http://www.bckelk.uklinux.net/`) containing inflected forms, and we combined it with the small one, containing most common abbreviations, names and places. The final dictionary is now 686 kb for 67 485 words.

3.2 Corpus collection and statistics calculation

A suitable corpus for text entry on mobile phones should contain mostly everyday English. However, must corpora are compiled from either news or literature and the language use from these two sources can differ quite much from everyday use. Therefore we decided to collect our own corpus. After some contemplation we decided that Usenet contains large volumes of text which quite close to everyday use. The problem with Usenet is that it also contains a large amount of, for us, unwanted content such as spam and binaries. However, certain news groups are more likely to contain usable text than others and hence we decided to limit our sampling to 118 subgroups of *alt.politics*, *alt.society* and *soc.* From these groups we collected 57 181 messages over a period of nineteen days. This was then compiled into a corpus of roughly ten million words.

First the uni- and bi-gram statistics were calculated from the corpus at runtime. Obviously this slowed down the application considerably as it could take over 2 minutes to compile all statistics with all n-grams present, see section 3.3. In order to avoid this lag when the application started it was decided to precompute all statistics. This was done through the use of a Python script which read in the normalized corpus and created one data file for

each desired n-gram, such as uni-, bi- and so forth. In our case we decided to only use uni- and bi-grams as the frequency of trigrams and higher was too low to yield any noticable effect. The application and its support scripts are however designed in such a way that adding support for higher n-grams is easy to do, see section 5.1.

3.3 Memory considerations

We noticed early on that this application requires a lot of memory. As a matter of fact in our initial revisions of the dictionary and statistics files it required up to 300 MB of RAM. It is important to note here, also, that the memory requirements are largely due to the fact that the data structures are written more with the aim of being easy to understand and maintain rather than to optimize memory consumption. Furthermore Java in itself creates a fairly large overhead when it comes to memory usage. However, we decided that we needed to reduce the amount of data used in the application. This was done by both limiting the size of the dictionary to only include more common words, less pronouns and so fourth. Furthermore a frequency cut-off was applied to both the unigrams and bigrams. Entities with a frequency of xx and yy respectively were removed. This measure ensured that the application consumed less than the 96 MB limit which is standard for Java applets.

Naturally the reduction of the dictionary and the cut-off reduces the accurary of both the uni- and bi-gram prediction but it is our distinct impression that it does not degrade the performance of the application by a great deal. One would of course have to conduct more thourough investigations of this matter to draw a more firm conclusion. Furthermore we believe that the fact that the application can run without problems both as an applet and from the command line is more important than the accuracy gained from lowering the cut-offs and increasing the dictionary size.

4 Combining bigram statistics, prediction and T9

First, we made a system using bi-gram statistics, prediction and unigram statistics at the same time, for each key pressed. The bigrams were obviously prioritized over the unigrams. But a problem was high-lighted: many times when typing a short word, longer words were given instead of having words with only the length of the number of key pressed, because of the prediction. For example, typing "of" after "no", the words given before "of" were "next", "need", "means", "news", "mention", "new", "mercy" and "official". Itincreased a lot the number of KSPC, to go down in the list in order to select it, whereas using T9, "no" appears first in the list. Then we decided to add a key, the "yes" key, which allows stopping the prediction and restricts the length of words given to the number of key already pressed. In the previous example, pressing this key would have reduced the list of words given to "me", "ne" and "of", because we pressed only 2 keys.

This kind of key is not natural at the beginning to using it, but permits to combine advantages of both HMS and T9 methods, i.e. prediction with bigrams and statistics of unigrams without prediction.

Moreover, we added two checkboxes, which are "prediction" and "context" and work in real-time. With them, the system is flexible and the user can choose to use the different advantages of HMS. In that sense, we can differentiate the 2 advantages of HMS compare to T9. With both check-off, the system works like T9. It is useful, then, to be able to choose the right method for the right word. The user can switch from one to the other when typing, to take the best advantage of the different methods (for example, we will choose "context-on" and "prediction-off" to use T9 with context, or choose "context-on" and "prediction-on" to write a long sentence in a good language, or choose "context-off" and "prediction-off" for short words, et cetera).

Finally, we added a functionality usually well appreciated by the users: the ability to learn. That means that each time a word is selected, its frequency is increased by 1. Then, more and more, the most common words used by the user become the first in the list and it is easier and quicker for him, decreases the KSPC.

However, as the system is settled and reloaded each time, it is not possible to save the scores of each user.

5 Implementation

As mentioned earlier one of our goals was to enable the application to be as flexible as possible when it comes both the depth of n-grams used and to have structures that are language independent. Of these the latter is the largest importance if this is do be deployed in real use. This is achieved by making the data structure in the application abstract enough to allow for different key pad encodings without having to rewrite any code. How this is done is discussed in section 5.1. One prerequisite for the data structure to work in this manner is to be able to translate a press on a button into a set of characters, e.g. the button labeled "2" on the keypad, see *figure 1*, represents the characters "a", "b", "c" and "2". By reversing this, e.g. saying that "a" maps onto the button "2", we can encode a word as a series of key presses given a certain encoding scheme, or key map. By stating that the buttons each have an index and that the first button is "1" with index 0. By using this the word "hello" would in English be encoded as the following sequence 3-2-4-4-5.

Another important aspect of any application is its user interface. In our case There was not really much choice as to how the application should look as its task was to mimic the front, or user interface, of a mobile phone and there is already a de facto standard for this. Hence the GUI is laid out with a text area above a keypad which the user can press with the mouse. One addition is the list of proposed words to the right of the text area and keypad. Furthermore we added two checkboxes to turn word prediction and context awareness on and off. The user interface of HMS2005 can be seen in *figure 1*.

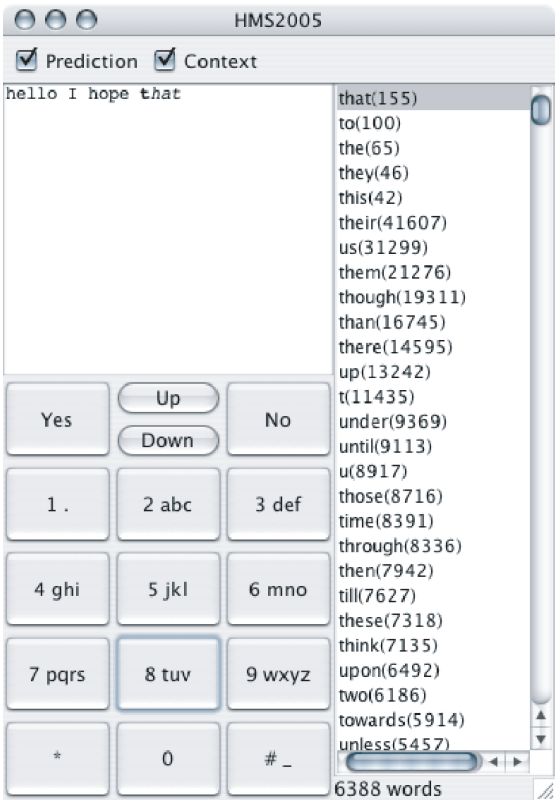

Figure 1: The user interface of HMS2005

5.1 Data structure

The application stores its data about words and their probabilities (unigrams) as well as bigrams in a common structure, a word tree. In the tree the nodes are connected by arcs which represents one of the buttons labeled 1 through 9, see *figure 1*. Each node also can contain a list of words which this node is said to represent. Furthermore the tree can optionally contain a link to a different word tree for each word, more on this later. And last, and quite naturally a node can have up to nine references to subtrees. Hence if we take the example with "hello" from above we would, starting from the root node, take the fourth arc followed third and so forth. After the sequence has been followed we would have reached the node containing the word "hello". With this solution word prediction just becomes the task of compiling a list of words from the subtree of the current node as well as the list of words contained in the node.

We can use this basic structure to include bigram information as well by, as hinted earlier, for each word in a node also link to another word tree, the bigram tree. When a word has been accepted by the user the bigram tree corresponding to that tree is stored. As the user types the next word the application traverses both trees with the same input and when the prediction lists are generated the list from the bigram tree is prepended to the unigram list. Of course this method could be repeated an arbitrary number of times to provide which ever n-gram depth desired. A schematic view of the data structure can be seen in *figure 2*.

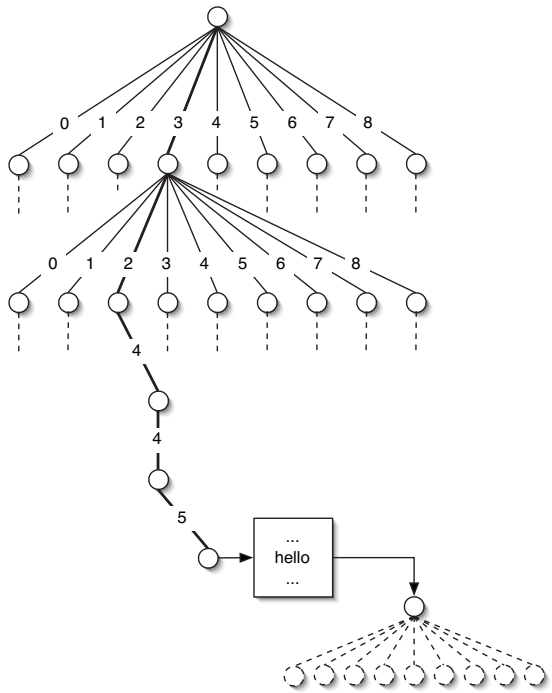

Figure 2: The data structure used in HMS2005 with the path leading to "hello" highlighted

The data for the application is stored as plain text lists with one entity per line and where each line consists of either just a word, for the dictionary, or the frequency followed by one or more words for uni- and bi-grams. When the application starts these are added to the tree in a such a way that only words in the dictionary are allowed, this to avoid unwanted words from the corpus to enter the tree. Of course this approach slow the loading process if there are many unwanted words but as the loading time was acceptable after cut-offs were applied, see section 3.3, we decided to keep this approach as it simplified our work.

The time consumption of this data structure depends largely of the length of the words stored. Of course going from one node to another can be done in constant time hence going from the root node to a certain node and finding a word within that node can be done in $O(n+m)$ time where n is the length of the word and m is the number of words within the destination node. When building a word prediction list the application has to first visit each node in the subtree and for each node a word list has to be compiled consisting of the words in that node concatenated with the words of that node's subtree. If we assume that adding a word to a list is $O(1)$ this phase will run in $O(nm)$ time where n is the number of nodes and m the number of words. The next phase will be sorting the list according to some criterion, usually frequency but it could also be alphabetically, and depending on the sorting algorithm used this can vary greatly. If we employ an fairly efficient sorting algorithm this phase can be done in $O(n \log n)$ time which would yield a total running time of $O(nm + n \log n)$. This can be optimized further of course, for example storing the words pre-sorted within the structure, but as the aim of this implementation is clarity and demonstration rather than efficiency and speed this must be considered adequate.

6 Evaluation

To evaluate our new method, we chose 2 factors of measurement: number of KSPC and time to entry the text.

We did first a "different participants for the same sentence" evaluation, i.e. each participant entries the same sentence in order to compare between participants, and secondly a "same participant for different sentences" evaluation, i.e. each participant entries 2 different sentences in order to compare between sentences.

A total of 7 participants were involved, typing firstly a sentence both in HMS and T9 (in this order), and secondly another sentence in the same way. The sentences were "I study at the University of Lund" and "Hello I hope you're fine and don't forget our meeting in Lund tomorrow morning".

Typing 2 different sentences was important because for the first one, people were disturbing by the new system, using it for the first time and trying to learn how it worked. There were friendlier with it for the second sentence.

The results were quite positive. Firstly, in terms of keystrokes per character, we highlighted an improvement of 20% (that means that you need 20% less key pressed to write the message). We can consider it as an average because we kept all the mistakes done by the users, which sometimes increase the KSPC instead of reducing it.

Secondly, in terms of time, we highlighted a weakness of 15% (that means that you need 15% more time to write the message). The problem is that the system disturbs the user at the beginning, mostly because of short words and because you have to remember the letters you have already typed.

Comparing trained and not trained people (both of us and other users), we noticed that for the more experienced users the time consumption of HMS and T9 were roughly the same although the KSPC for HMS was lower. Furthermore, we noticed a big difference in time consumption when it came to trained and untrained people. We measered a 200% speed increase for trained people and this can probably be explained by the fact that as you get used to using the bigram prediction you get more confident that the system will predict the correct word. Naturally this is probably also true for new users of T9.

But the most important thing which reduces the efficiency of the system is the length of words. HMS reduces a lot the KSPC for long words but not for short words. This can be seen in table 1.

word	**T9**	**HMS**
tomorrow	8 kp	4 kp
in	2 kp	3 kp

Table 1: Key press Comparision

The problem is that when typing short words, long words are proposed first and you have to press one key more (the "Yes" key or the "T9" key) to restrict the length or stop the prediction. This increases KSPC and time consumption and could be optimized. It can be seen from the following two figures, *figure 3* and *figure 4*, how the KSPC changes as the word length increases.

7 Conclusions

As we have seen, we tried to improve the HMS system, which is a predictive text entry method using context. We first made it works in English, collecting an English dictionary and a corpus of the closest language from SMS language we could. After, we improved it by adding a key which can stop the prediction and come back to the T9 method. This key permits to take both advantages of HMS and T9 methods when typing. Then we made the system intelligent by learning the words the most used by the user (without reload the system). After an evaluation, we showed an improvement of 20% using the KSPC as the measurement, and a decrease of 15% using the time as the measurement.

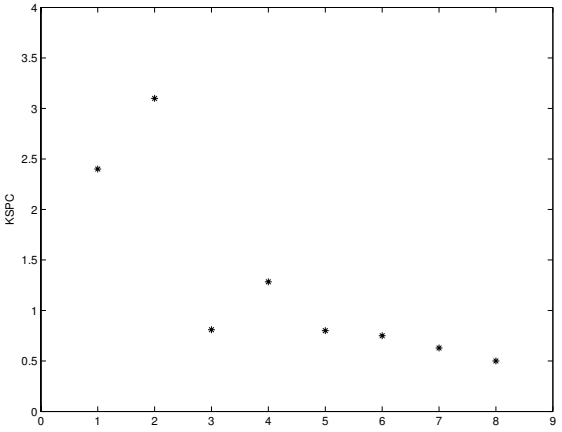

Figure 3: Word length versus KSPC for HMS

7.1 Improvements

But of course, this system can be optimized and improved again. Firstly, to be closer to the existing mobile phones, numbers, punctuation and special characters could be added in the keyboard. This could be used to do longer and more complex experiments and could be closer to the reality. Secondly, our results are not as closed to the reality as they could be because of the corpus we used to calculate the bigrams. We wanted to use free corpora of SMS but none exists yet today, so we used corpora from Usenet. This kind of corpora of SMS are collecting by researchers just now, because the technology is new and these corpora are needed to extend the researchs. We found one corpus of english SMS from Singapore, but not really useful because of strange words or names sometimes. Thus, we think that in a few years, more SMS copora will be available for researchs and evaluation will be closer to the reality.

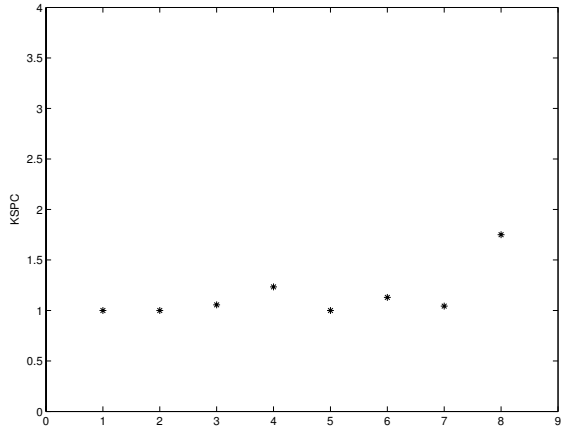

Figure 4: Word length versus KSPC for T9

References

[1] Zi Corporation. *eZiText*, 2002.

[2] Lexicus Devision. *iTap*. Motorola, 2001.

[3] Jon Hasselgren, Erik Montnemery, Pierre Nugues, and Markus Svensson. HMS: Predictive text entry. In *DAT171*, 2003.

[4] I. Scott MacKenzie. KSPC (keystrokes per character) as a characteristic of text entry techniques. In *http://www.yorku.ca/mack/hcimobile02.PDF*, 2002.

[5] Scott MacKenzie, Hedy Kober, Derek Smith, Terry Jones, and Eugene Skepner. Letterwise: Prefix-based disambiguation for mobile text input. In *http://www.eatoni.com/research/lwmt.pdf*, 2001.

[6] Andriy Pavlovych and Wolfgang Stuerzlinger. Less-tap: A fast and easy-to-learn text input technique for phones. In *http://www.cs.yorku.ca/ andriyp/papers/GI2003_Less-Tap.pdf*, 2003.