1K2-IOS-1b-2

A Personalized Thai Consonant-only Input Method for Smartphones

Panithan Ballungpattama[*1] Piyoros Tungthamthiti[*1] Cholwich Nattee[*1]
[*1]School of Information and Communication Technology
Sirindhorn International Institute of Technology
Bangkok, Thailand
panithan.ballungpattama@student.siit.tu.ac.th, xylz_jo@hotmail.com, cholwich@siit.tu.ac.th

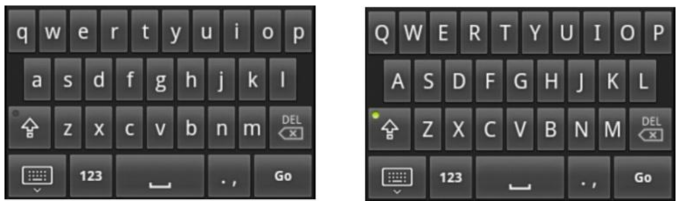
Due to the portable size of mobile phones, text input method using a virtual keyboard on smartphones is inconvenient task for all users, especially for Asian language (e.g. Chinese, Japanese, Thai etc.), which has a lot of characters and symbols. This paper presents a development of a text input method based on Thai language and introduces a Thai consonant-only keyboard, and an improvement in candidate generation algorithm for handling Thai text input on smartphones with touch screen. The proposed layout shows only Thai consonants instead of the combination of Thai consonants and vowels. User inputs only a sequence of consonants that is a part of the intended word. Then, we propose an approach based on the concept of Personalized Candidate Selection technique to choose a set of most appropriate words before making any suggestion to the user. In comparison between the proposed technique and the existing Thai keyboards in term of the number of keystrokes required for inputting words, the experimental results show that the proposed approach helps save 49.71%.

1. Introduction

Smartphones have become one of the most important devices for communication in the modern world. It is because there is a very high demand from busy people who want to be able to use computers anywhere and anytime. A smartphone is a mobile phone that includes advanced functionality beyond making phone calls and sending text messages. A smartphone is a personal digital assistant or a computer that will allow you to have the ability to send and receive e-mails and edit documents, display photos, play videos and surf the web. The device has a compact size and very light weight, which makes it convenient to carry around. These characteristics seem to satisfy the need of busy people very well. However, there are some problems arising from its portability. It is obvious that a full-sized keyboard cannot be used due to its large size. So, it is necessary that a new input method (e.g. characters) needs to be developed to increase the performance of the device according to speed and accuracy.

Text input methods offer a great potential in typing on mobile and handheld devices, which have limited and fixed size of screen area. Generally, text input method will allow users to input some characters, then suggested candidate words will be generated according to the algorithm of each input method. For example in consonant-only Thai keyboard, when 'ก' is given as an input, these words "กะ", "เกาะ" and "กา" are generated as word candidates.

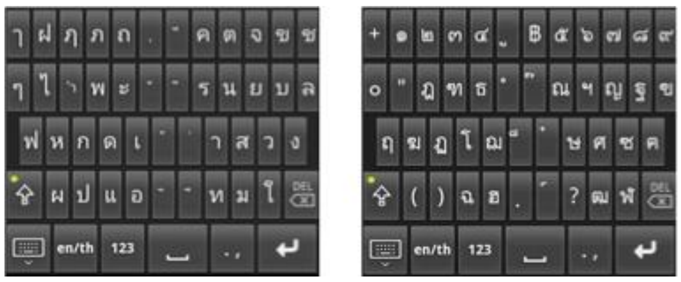
Nowadays, QWERTY (shown in Figure 1: Standard keyboard layout for English language) is one of the most popular keyboard layouts for smartphones. The reason is because the layout is the same as a physical keyboard and most of the smartphone users would be familiar with it.



(a) Normal (b) Shifted

Figure 1: Standard keyboard layout for English language

However, when it comes to Thai language, which has a lot of alphabets, the keyboard consists of 2 layers (shown in Figure 2). They are normal and shifted layers. Each layer has different number of keys and sizes. In comparison to the QWERTY keyboard, Thai language keyboard has more characters, vowels and tone marks. This makes the button size become smaller and causes a decrease in accuracy and typing speed. So, the layout needs to be further developed to improve an efficiency of the input method.



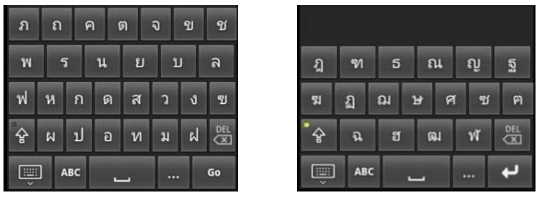
(a) Normal (b) Shifted

Figure 2: Standard keyboard layout for English language

This paper will introduce a new Thai keyboard layout, which will be composed with only Thai consonants and also explain how to improve an efficiency of the consonant-only Thai keyboard layout by increasing the accuracy and speed of typing [Masui 1999]. The layout will be developed according to Kedmanee layout, which is the standard keyboard layout for Thai language. We also propose a way for the soft keyboard to recommend the probable words according to the user input. This will allow the keyboard to have a larger key size and less keystroke is required. Candidate generation using set and personalized candidate selection techniques will be used to improve the accuracy of the algorithm for candidate suggestions.

2. Consonant-only Thai Input Method

Ballungpattama et al. [Ballungpattama 2011] has proposed a Consonant-only Thai Input Method. Due to the large number of Thai characters, vowels and tone marks, consonant-only Thai keyboard will be implemented to reduce the number of buttons and increase the size in order to increase efficiency of usage. Since vowels and tone marks are removed, the smallest width of keys on out proposed keyboard is roughly 12.5 percent, which is about 6 millimeters on our device, which is not the optimal width of the button size at 9.6 millimeters as Parhi et al. [Parhi 2006] suggested. The consonant-only keyboard will be created based on Kedmanee layout. So, users can get familiar with the keyboard easily. Figure 3 shows our proposed layout for normal and shifted keyboard. Without vowels and tone marks on our proposed keyboard, we propose a technique to predict a list of high possibility words. From a sequence of consonants input by the user, our proposed technique retrieves a list of words composed of all consonants in the input sequence from the pre-constructed dictionary, and ranks the candidate words based on their occurrence frequency. Based on this idea, users do not need to spell the full word.



(a) Normal (b) Shifted

Figure 3: Keyboard layouts for consonant-only Thai keyboard

The consonant-only Thai keyboard also provides a word suggestion functionality, which generates a list of candidate words even if the user input a few characters. This process designed the dictionary based on the inverted index from the Boolean retrieval technique [Manning 2008]. An inverted index consists of two part e.g. a *dictionary* (a list of words) and *postings* (a list of keywords). This is the structure behind the preparation of a dictionary and the candidate generation algorithm. Data structures used are a word list, a dictionary, and postings; all of this is illustrated in Figure 4.

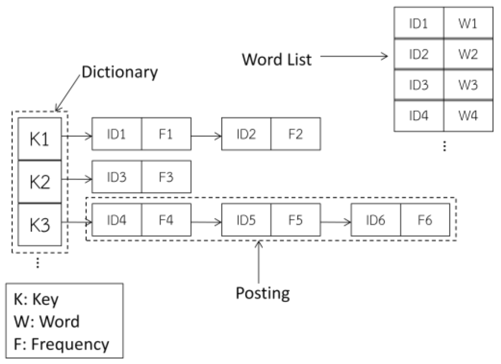


Figure 4: Structures used in Consonant-only Thai Keyboard

The *word list* stores the words' ID and the words themselves. One ID will be associated to one word. When the candidate list is generated, the program uses the word list to fetch words from IDs and send words to the CandidateView. This word list is generated from the dictionary generation algorithm.

A *dictionary* is the structure, which stores words into postings according to their *keys*. There are two types of keys: single- consonant keys from Thai characters 'ก' (Ko Kai) to 'ฮ' (Ho Nokhuk), including a rarely used 'ฤ' (Ru); and pairs (double-consonant keys) from "กก" to "ฮฮ" for a total of 2,025 keys.

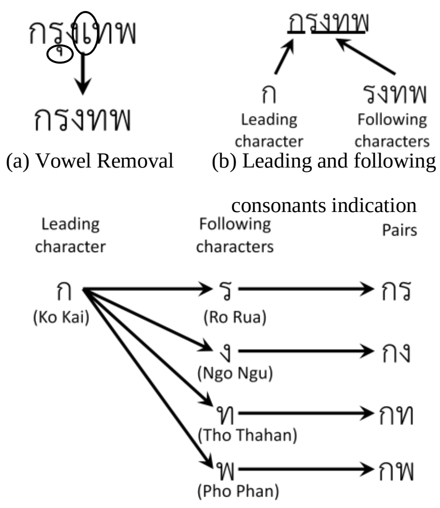
Postings store words ID and frequencies. Normally, each postings is associated with one words' ID, however, there are many words in Thai which contains only one consonant but have different vowels and tone marks, for example, some of the words which have only a consonant 'ก' are "กะ" (to estimate), "เกาะ" (island), and "กา" (crow); whereas an English word which has only one alphabet is an alphabet itself. Therefore, postings must be implemented in a way that can contain one word and a pointer to the next word. This principle is also true for keys with two characters. Double-consonant keys, or *pairs*, consist of one leading and following consonants. The first consonant in a word is considered a *leading* consonant. Other words will be treated as *following* consonants, for example, a pair กก contains the words "กก" (a kind of tree), "กากบาท" (cross), and "เกะกะ" (cluttered).

3. Algorithms

Our proposed method used these two important algorithms. They are the dictionary generation and the word suggestion algorithm.

3.1 Dictionary Generation

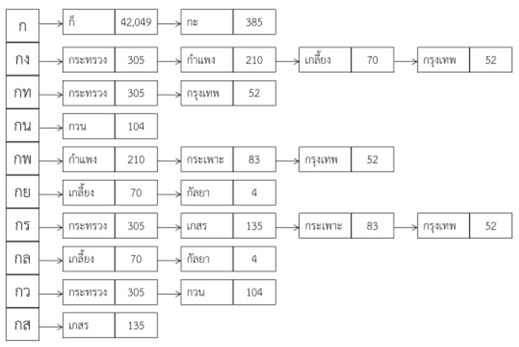
This algorithm takes a word, removes vowels and tone marks, indicates keys from consonants, and put a word in keys. Figure 5 shows a dictionary generation method for a word กรุงเทพ (Bangkok). This process is considered a preprocessing because it will occur only when installing the application into smartphones and when users use this application after turning on smartphones.



(c) Determine pairs to put words

Figure 5: Three steps of dictionary generation

According to the dictionary generation method, possible words will be generated as candidates. Inverted index will be created and possible words and their frequencies will be stored within the index. Figure 6 is how the complete inverted index looks like.

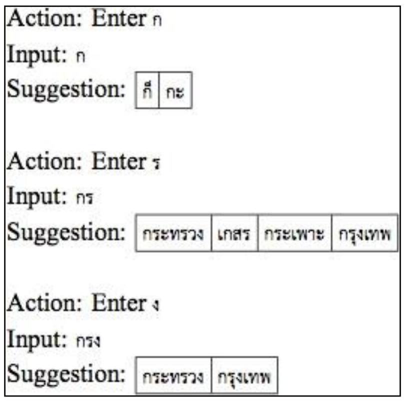


ก	ก็	42,049	กะ	385				
กง	กระทรวง	305	กำแพง	210	เกลี้ยง	70	กรุงเทพ	52
กท	กระทรวง	305	กรุงเทพ	52				
กน	กวน	104						
กพ	กำแพง	210	กระเพาะ	83	กรุงเทพ	52		
กย	เกลี้ยง	70	กัลยา	4				
กร	กระทรวง	305	เกสร	135	กระเพาะ	83	กรุงเทพ	52
กล	เกลี้ยง	70	กัลยา	4				
กว	กระทรวง	305	กวน	104				
กส	เกสร	135						

Figure 6: An complete inverted index

3.2 Candidate Words Generation

The candidate words generation, or *word suggestion*, choose the appropriate set of words according to user input and choose them as word candidates. This algorithm works differently for different input length. When user inputs one or two consonants, obviously, a key for fetching candidates will be an input itself. If the input has at least three consonants, the keys will be indicated using a combination of leading and following consonants as in Figure 5c. Then, the algorithm compares lists and removes words that are not in every list while keeping words in all lists. Finally, candidate words are put into a list before sending to the candidate view. Figure 6 shows candidate words when a user giving different inputs.



Action: Enter ก
Input: ก
Suggestion: ก็ | กะ

Action: Enter ร
Input: กร
Suggestion: กระทรวง | เกสร | กระเพาะ | กรุงเทพ

Action: Enter ง
Input: กรง
Suggestion: กระทรวง | กรุงเทพ

Figure 6: An example of output shown with given inputs

4. Candidate Word Generation Algorithm Improvements

4.1 Initial Implementation

The initial implementation of the consonant-only keyboard will be done according to the candidate generation algorithm in the previous section. For an input with one or two characters, a candidate will be obtained from the respective pair. For an input with three or more characters, the candidate list of pairs will be stored in a two-dimensional array. Each row represents a word list of each pair and each column in a row represents each word in a pair. Words in lists are arranged by their frequency in the descending order. Words that appear in all rows will be chosen as candidates.

4.2 Implementation Improvement

A new candidate generation algorithm is proposed. This new method keeps track of a candidate words list from a previous input; with that, the candidate list for a new input can be created by just a few altering from the previous list without having to generate the candidate list all over again. Then, a candidate list from each step will be stored in (or obtained from) a stack. The new method will do different operations according to how the input length change, but the most prominent operation used is an intersection for sets and peeking for a stack. The intersection of two sets is the set containing all of the elements contained in either set [Oracle inc. 1995]. Peeking a stack gets an object on top of the stack without removing it from the stack. Let *n* be a length of current input and *x* be any number less than *n*, the pseudocode is shown in Figure 7.

```
START CandidateGenerationSetStack
  IF(oldInputLength == 0 and newInputLength == 1)
    //case 1: length change from 0 to 1
    do nothing;

  ELSE IF(oldInputLength == 1 and newInputLength == 2)
    //case 2: length change from 1 to 2
    create a set from a key;
    push a set into a stack;

  ELSE IF(oldInputLength == 2 and newInputLength == 1)
    //case 3: length change from 2 to 1
    pop a set from the stack;

  ELSE IF(oldInputLength == x and newInputLength == x+1)
    //case 4: length change from x to x-1, x = 2, 3, ...,
    n-1
    create a new set from current key;
    intersects the previous set with the current set;
    push the newly intersected set into a stack;

  ELSE IF(oldInputLength == 3 and newInputLength == 2)
    //case 5: length change from 3 to 2
    pop a set from the stack;
    peek a top set from the stack;

  ELSE IF(oldInputLength == x and newInputLength == x-1)
    //case 6: length change from n-1, n-2, ..., 4
    pop a set from the stack;
    peek a top set from the stack;

  ELSE IF(oldInputLength == 0 or 1 and newInputLength ==
    0)
    //case 7: delete all input or backspace even if there
    are no input
    clear set;
    clear stack;

  END IF
END CandidateGenerationSetStack
```

Figure 7: Candidate generation using a set and a stack

4.3 Improving Usability

Consonant-only Thai Keyboard provides a list of candidate to users when users type in consonants. In an old implementation, words in a candidate list are picked from key(s) and then arranged by frequencies; the frequencies are taken from BEST2010 word list into a form of text file before presenting words to the user. Still, arranging words based on only the frequency seems inefficient. If a user would like to choose one word with has a very low frequency, but a user often use it; that word will be very far away, almost at the end of a candidate list. Users have to scrolling to find that word, which are very time consuming when done many times. To solve this problem, the keyboard should be smart enough to remember which word a user has used frequently; the keyboard should do something to increase the priority of that word so that it will appear in a better location, while still retaining the frequency value.

In addition to the rank increase for frequently used words, the word similarity should be taken into account too. For example, a user would like to choose a word ปืน (gun), the user type two consonants ปน (Po Pla and No Nu), but our keyboards recommends long words like เปลี่ยนแปลง (to change), ประธานาธิบดี (president), ประชาสัมพันธ์ (public relations).etc , leaving a word ปืน (Gun), falling far behind because it has less frequency than these long words. Our keyboard should be able to improve the rank of words which are more similar to input too.

4.4 Personalized Candidate Selection Technique

Instead of calculating words order using only a frequency, the order will be calculated using a probability points. The formula to calculate the probability point is:

$$TotalPoint = \alpha p_f + \beta p_d + \gamma p_u$$

P_f, or *frequency points*, are points given to words' frequency. Now, a frequency is just one part of the point calculation.

P_d, or *distance points*, are points given to edit distance between user's input and words in list. An *edit distance*, also known as *Levenshtein Distance* [Levenshtein 1966], is a minimum number of *edit operations* to transform one string to another string [3]. The operations are inserting a character into a string, deleting a character from a string, and replacing a character with another character. A distance point is inverse-proportional to a Levenshtein Distance because more Levenshtein Distance means more difference between an input and words [Gilleland 2009]. For example, the Levenshtein distance between an input "กรง" (cage) and a word "กรุงเทพ" (Bangkok) is 4. Table 1 shows the transformation from "กรง" to "กรุงเทพ"

Table 1: Levenshtein distance computation for กรง and กรุงเทพ

Step	Operation	Before Change	After Change
1	Insert ุ (Sara U)	กรง	กรุง
2	Insert เ- (Sara E)	กรุง	กรุงเ
3	Insert ท (THO THAHAN)	กรุงเ	กรุงเท
4	Insert พ (PHO PHAN)	กรุงเท	กรุงเทพ

P_u, or *user points*, are points given to words that were chosen by the user. Every time a user chooses a word, its user point increases by one. Since this value affects points calculation, user points and chosen words are kept in another file separated from dictionary and index files; this file, along with other files, are loaded when the onCreate method is called. In addition to that, user point files will change its content every time a user choose a word, so that a rank improvements can be seen immediately the next time a user try to select the same word.

a, b, and g are weights given to frequency points, distance points, and user points, respectively.

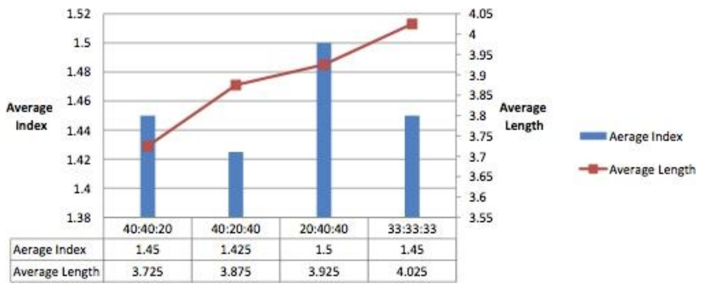
5. Experiments and Results

5.1 Finding a suitable weight

According to the Personalized Candidate Selection Technique, we would like to know which weight distribution would give out the best result. An experiment to determine a suitable weight for each point is conducted. This experiment is conducted on the test set which consists of 20 words randomly chosen from the corpus with varying lengths and frequencies. Five users need to type all these words correctly before proceeding to next word. The value of α, β, and γ that will be used in the test are:

- Give more weight to α and β (40:40:20)
- Give more weight to α and γ (40:20:40)
- Give more weight to β and γ (20:40:40)
- All has the same weight (33:33:33)

The sum of the ratio must be one hundred, with an exception of the fourth case where we would like to distribute all weight equally, which is ninety-nine. Also, note that one weight should not be one hundred percent and left the other two weight become zero; that is because the point from other value will never be used, thus making the candidate generation algorithm being affected by only one value. Figure 8 shows the result for different weights. The graph shows the average word index that users touch and the average input length before users touch the word shown on candidate view. From this graph, two ratios stand out. The ratio 40:20:40 let users choose the word easier because of the average index is the lowest with an exchange of a bit more keystrokes. The ratio 40:40:20 let users input the least amount of input length before showing a candidate list to select a word. Although the actual values are not very different between these four, we will consider more on the index because one more index behind may require users' effort to scroll the candidate list a bit further, taking a bit of typing time. Also, scrolling the list does not count toward the number of keystroke used. Therefore, the suitable weight for a test should be α equals to 40 percent, β equals to 20 percent, and γ equals to 40 percent.



	40:40:20	40:20:40	20:40:40	33:33:33
Aerage Index	1.45	1.425	1.5	1.45
Average Length	3.725	3.875	3.925	4.025

Figure 8: A graph, which shows index and input length for different weight distribution

5.2 Experiments

After we obtain the suitable weight for this experiment, we will start evaluating out proposed method. Another application has been created solely for a purpose of testing Consonant-only Thai Keyboard. The test program lets testers type fifty words that are shown on the screen. Testers have to type words correctly in order to advance to next words. The test set contains fifty words, which are directly taken from Table 2. Each word must be correctly typed before advancing to the next word. After testers type fifty words (one by one) correctly, the program will navigate a user to the summary page. The screen-shots from the test program are shown in Figure 9.

Table 2: Words length, frequencies, and number of words for each type

High frequency, Short words, 6 words	High frequency, Medium words, 5 words	High frequency, Long words, 6 words
Medium frequency, Short words, 5 words	Medium frequency, Medium words, 6 words	Medium frequency, Long words, 5 words
Low frequency, Short words, 6 words	Low frequency, Medium words, 5 words	Low frequency, Long words, 6 words

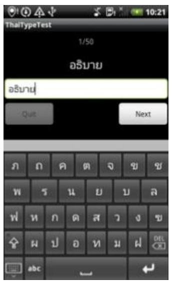


Figure 9: A screenshot of the test program

Three statistics will be used as the key indicators to measure the performance of Consonant-only Thai Keyboard:

- Keystroke Saving Rate (KSR): The percentage of the keystroke that can be saved.
- Time per Word (TPW): The average number of time in second for typing one word.
- Time per Keystroke (TPK): The average number of time in second between one keystroke and another keystroke.

Consonant-only Thai Keyboard is test against Droidsans Thai keyboard as an on-screen keyboard with Kedmanee layout. The results are shown in Table 3.

By using Droidsans, testers are required to type all the consonants, vowel, and tone marks, which is 100 percent of the keystroke typed; hence 0 percent of keystroke saved. The number of keystroke typed on Droidsans will be compared with the number of keystroke typed using Consonant-only Thai keyboard to determine the efficiency of Consonanr-only Thai keyboard. From Table 3, Consonant-only Thai Keyboard can save the keystroke with a satisfactory rate of 49.71 percent when compared to Droidsans. The keystroke reduction rate of Consonant-only Thai keyboard is obviously less than its earlier version, which achieve up to 71.29 percent. However, the earlier version performs very slowly. Therefore, we have to improvise the algorithm and data structure to makes it becomes faster, but at the same time, affecting the keystroke saving rate. Another important reason is the difference of the test method. In the prototype, the optimal text input does all possible tapping for the best result, which could take plenty of time. In Consonant-only Thai Keyboard, testers are free to type in any order as they like, as long as the words are correct. Testers would not try typing in every possible combination because that will consume too much time. Also, the time per words shows that using Consonant-only Thai Keyboard makes typing become faster than Droidsans with the difference around 4 seconds. Surprisingly, the time per keystroke is slower than Droidsans. The cause of this might be testers' familiarity with Kedmanee keyboard makes them spell the words, even with vowels and tone marks, become faster than Consonant-only Thai Keyboard.

Table 3: An experiment results for Droidsans and Consonant-only keyboard

Keyboard	KSR (%)	TPW (s)	TPK (s)
Droidsans	0	10.84	1.09
Consonant-only Thai Keyboard	49.71	7.01	1.32

From the experiment, we also classified the testers into two groups: the beginners and the experienced testers. There are 4 testers in the beginners group and 3 testers in the experienced tester group. Both groups have been given a short introduction to Consonant-only Thai Keyboard and a demonstration on how to use it. The beginners are allowed to use our proposed keyboard for once or twice whereas the experienced users are allowed to try our proposed keyboard for as long as they want. The empirical results are shown in Table 4.

Table 4: An experiment results for different types of users

Tester	KSR (%)	TPW (s)	TPK (s)
Behinner	45.96	8.40	1.52
Experienced	53.45	5.63	1.12

From Table 4, we could see that the experienced testers can save the keystroke up to 53.45 percent, which more keystroke is saved than beginners do. This is because the experienced users know which consonant should be input next in order to make the candidate and a target word appear. However, the number of keystroke saved by the experienced testers is less than we anticipated. We expected them to save up to 65 percent. Still, the time saved by word and keystrokes are within our expectations. The experienced users perform slightly faster than beginners. This is because they are more familiar to Consonant-only Thai Keyboard than beginners; they are allowed to try our proposed keyboard as long as they like.

The next thing we would like to discuss is the keystroke saving rate for words with different length: short, medium, and long. The rate will be shown in percentage when compared with Droidsans Thai keyboard. The result is shown in Table 5.

Table 5: Keystroke saving rate for different word lengths for Consonant-only Thai Keyboard

Word Length	Short	Medium	Long
KSR (%)	33.19	49.89	55.89

From Table 5, we could see that the longer the words are, the more percentage of the keystroke saved. This is because the extra keystrokes from pressing shift button and selecting a word could make the number of keystroke almost equal to the number of consonants in words. Thus, less keystroke reduction rate is achieved. There are two reasons to support the reduction of keystrokes. Firstly, our proposed keyboard does not require users to input vowels and tone marks; users are allowed to type only consonants. Even if users type all consonants in a word, the keystroke will be less than a full word in case of words with vowels; or equal to a full word if that word contains only the reduced or changed form of the vowels or contains อ (O Ang), which is considered as both a consonant and a vowel. Secondly, words chosen for testing have different frequency and length. Some of chosen words are not usually appear in general usage such as specific nouns, Thai royal words, or words borrowed from foreign languages; these words often contain uncommon consonants. For a same word, after inputting the leading consonant, inputting rarely used consonants first yields a better chance for a target word to appear than typing frequently used consonants. All of the reasons specified earlier contributed to the keystroke saving rate using Consonant-only Thai Keyboard.

6. Conclusion

We have introduced several ways to improve a prototype of Consonant-only Thai Keyboard to minimize the execution time and maximize the usability. Firstly, a single text file, which makes up the dictionary and the corpus are split into two binary files with proper formatting for faster loading. Secondly, the implementation of the candidate generation algorithm is changed from 2D array to set and stack to be able to keep track of the candidate list from previous input and for better performance. Thirdly, a formula to calculate the order of words now uses the probability point, which is the sum of frequency points, distance points, and user points; with appropriate weight distributed. Finally, a test program is created for users to type specified words as a mean for evaluation. The results show the overall performance data such as time taken, the word per minute, the number of second per word and keystroke; and the keystroke saving rate for each word type, with a decent keystroke saving rate of 49.71 percent when compared to normal Thai keyboard.

References

[Masui 1999] T. Masui, "POBox: An Efficient Text Input Method for Handheld and Ubiquitous" in *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, 1999.

[Parhi 2006] Pekka, "Target size study for one-handed thumb use on small touchscreen device," in *Proceedings of the 8th conference on Human computer interaction with mobile devices and services*, 2006.

[Manning 2008] Christopher, "*Introduction to Information Retrieval*", Cambridge University Press, 2008

[Ballungpattama 2011] Panithan, "A Consonant-only Thai Input Method", *In proceedings of the International Conference on Knowl- edge, Information and Creativity Support Systems(KICSS2011)*, 2011.

[Oracle inc. 1995] Oracle inc., "The Set Interface", *http://docs.oracle.com/javase/tutorial/collections/interfaces/set.html*

[Levenshtein 1966] V. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals", *Soviet Physics Doklady*, vol. 10, 1966.

[Gilleland 2009] M. Gilleland, "Levenshtein Distance, in Three Flavors", 2009, *http://www.merriampark.com/ld.htm.*