Designing Soft Keyboards for Brahmic Scripts

Lauren Hinkle

Abstract

Soft keyboards, because of their ease of installation and lack of reliance on specific hardware, are a promising solution as an input device for many languages. Developing an acceptable soft keyboard requires frequency analysis of characters in order to design a layout that minimizes text-input time. This paper proposes using various development techniques, layout variations, and evaluation methods for soft keyboard creation for Brahmic scripts. We propose that using optimization techniques such as genetic algorithms to develop multi-layer and/or gesture keyboards will increase the speed at which text can be entered.

1 Introduction

In an increasingly fast paced world, being able to input text quickly is important to all users. Standard Roman-alphabet based languages have been studied in detail, and it is important to now develop efficient keyboards in languages that have not been highly researched. Many Indic languages have only rudimentary keyboards that were developed so that users could input text, not with the goal of optimization. As a result, many of these keyboards have not had a significant impact because they are difficult to learn and inefficient use.

Soft keyboards allow a user to input text with and without a physical keyboard. They are versatile because they allow data to be input through mouse clicks on an on-screen keyboard, through a touch screen on a computer, cell phone, or PDA, or by mapping a virtual keyboard to a standard physical keyboard. With the recent surge in popularity of touchscreen media such as cell phones and computers, well-designed soft keyboards are becoming more important everywhere.

For languages that don't conform well to standard QWERTY-based keyboards that accompany most computers, soft keyboards are especially important. Brahmic scripts (Columas et al. 1990) have more characters and ligatures than fit usably on a standard keyboard. A soft keyboard allows any language to have custom layouts based on the frequency of character and ligature use within that language that do not conform to a standard keyboard. While physical keyboards have been designed for Brahmic scripts, such as (Joshi et al. 2004), they are cumbersome and difficult to learn. The development and spread of soft keyboards would allow the use of an easier to learn and more efficient keyboard.

Web service providers such as Google.com and Wikipedia.com have developed soft keyboards for Brahmic languages, but these are not optimized. Computer users are often forced to use English keyboards to tediously type their script. This limits people's ability to use computers and thus connect themselves to the many resources computers can provide. The development of soft keyboards for these languages has the dual benefit of allowing more people to be able to use a computer in their native language and to prepare users for touch-screen technologies.

Developing a soft keyboard for a Brahmic script is important, but in order for it to be usable it is necessary to design it such that it is easily learned as well as efficient.

An efficient soft keyboard must optimize the speed with which a user can input text using only one input device (for example a stylus, finger, or mouse). A user must be able to select the desired characters as quickly as possible. Brahmic scripts pose an important problem that must be overcome: the number of characters that should appear on the keyboard to maximize efficiency. Brahmic scripts have more characters than the standard Roman-based languages most people are familiar with. For example, the Eastern Nagari script has 37 consonants, 11 vowels, and 4 special characters that can be used individually as well as combined to create about 250 different ligatures.

In addition to deciding whether ligatures should appear on the keyboard, the layout of the characters must be designed with efficient text-entry in mind. More frequently used characters should be placed centrally, and characters that are often used together should be located close to one another. This would minimize the travel distance between characters being selected and therefore increase the input speed.

2 Related Work

While efficient soft keyboards have not been widely developed for Brahmic scripts, there has been much research in techniques for development of soft keyboards in English.

Soft keyboard layouts have developed from simple and familiar to seemingly arbitrary, yet designed by computers to be as efficient as possible. Early techniques for English soft keyboards used either alphabetic ordering or the traditional QWERTY layout. While these are still the most commonly used keyboards, great enhancements have been made in optimization. MacKenzie et al. were among the first to design a layout based on character frequency. They used Fitt's Law and trial-and-error hand-placement of frequently occuring bigraphs to develop the OPTI keyboard (MacKenzie and Zhang 1999). Keyboards were further improved upon by using computers to iterate over many solutions to computation-heavy algorithms from physics such as Hook's Law and a Metropolis random walk algorithm. Machines developed and tested increasingly efficient soft keyboards that were able to reach theoretic upper-bounds of text input of approximately 42 wpm (Zhai et al. 2000). The next major step in efficiency was undertaken by employing genetic algorithms. The keyboards generated using this technique were more efficient for every layout tested than any previous soft keyboard (Raynal and Vigouroux 2005). The impressive performance of genetic algorithms leads to the idea of using other optimization techniques for keyboard layout. While such techniques have not been applied to keyboards aimed at the general public, ant colony optimization strategies have been applied to virtual keyboards designed to make text input simpler for people with disabilities (Colas et al. 2008).

If these steps worked to create efficient soft keyboards for English, we hypothesize they will work as well for other languages.

Brahmic soft keyboards are currently at a point in their development analogous to where English soft keyboards were a decade ago. They are currently organized in traditional alphabetic ordering or in consonant-vowel groups. However, there have been several interesting techniques for input schemes employed. Although there are some designs for single layer keyboards, reminiscent of English soft keyboards (Sowmya and Varma 2009), this is not the trend in Brahmic keyboards.

As a result of the great number of characters in the alphabets, including diacritics, many Brahmic keyboard designers have opted for layered keyboards. A layered keyboard has multiple characters residing in the same location that are accessible by either hitting a button that toggles between layers (Rathod and Josh 2002) or rolling over base characters to reveal others (Shanbhag et al. 2002). Shanbhag's keyboard has a combination of these effects, with multiple layers that can be toggled between as well as a layer that has consonants grouped together and accessible by rolling over the group icon (Shanbhag et al. 2002).

Another technique to decrease the necessary number of keys is the use of gestures on the keyboard. In a gesture keyboard, the user draws any necessary diacritics for each character as they select it (Krishna et al.

2005). Gesture based keyboards have taken some of their ideas from English gesture-based keyboards such as T-Cube, in which a user selects different characters by flicking their mouse or stylus in different directions (Venolia and Neiberg 1994).

These techniques have the potential to increase the input speed of the user by decreasing the search time for characters and decreasing the necessary distance to travel between characters. However, no theoretic input speeds have been calculated for them. In addition, both techniques stand to benefit by reordering the keyboard by frequency of use of unigraphs and bigraphs, rather than alphabetically.

3 Brute-Force Brahmic Soft Keyboard Design

As a first attempt in designing a soft keyboard, we took a brute-force approach. We implemented a three-level pop-up menu based soft keyboard for Assamese, the Eastern-most Indo-European language and one that uses the Eastern Nagari variant of Brahmic scripts. The first level provided an individual key for each consonant (See Figure 1). It also had a single key for each of the following sets of characters: vowels, vowel diacritics, digits and special punctuation marks.

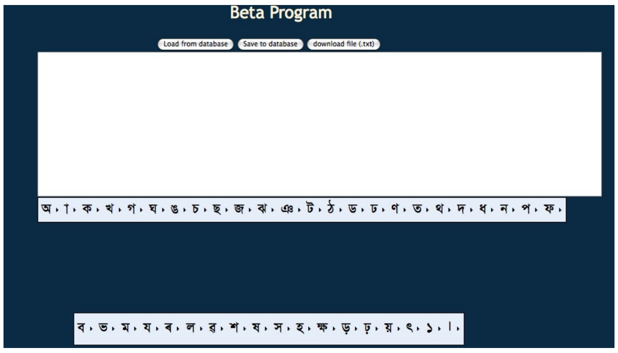

Figure 1: The top-level menu items for character entry. The 11 vowels are represented by the first menu item in the top row. The second item in the top row represents the medial diacritic representations of the eleven vowels. The last three items on the bottom row represent special characters, numerals and punctuation, respectively.

The second level of keys were organized as follows. When the first vowel, , which appears on the initial screen, is pressed on, a horizontal pop-up menu with the 10 other vowels appeared. Similarly, when one presses on the first diacritic , the diacritics for the ten other vowels appear in a horizontal pop-up menu. When one presses on any of the consonants at the first level, every combination of the consonant with vowel diacritics and every ligature in which the consonant participates shows up. The second level menus are rectangular. An example of the menu that pops up for is shown in Figure 2.

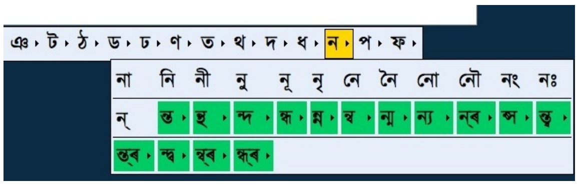

Figure 2: Second level menu (light blue in green) for a consonant.

There exists a third level of menus as well. Each ligature in the second layer has a sub-menu of all the vowel diacritics that can be added to it. The level 3 menu for a ligature is shown in Figure 3.

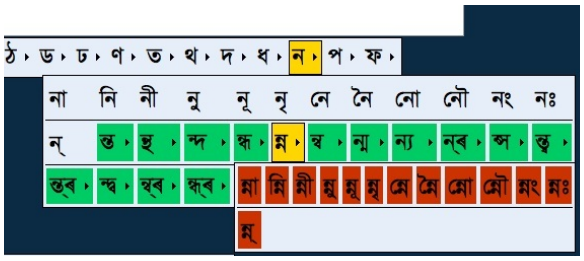

Figure 3: Third level menu (in red) for a ligature

Thus, in our three-level menu system, we provide direct access to every vowel, vowel diacritic, digit, special punctuation mark, consonant, and consonant-vowel diacritic combination, as well as every ligature-vowel combination. However, this made for an unweildy keyboard with access to over 3000 character combinations in which it is possible to type anything that can appear in print by menu traversal only.

Our tests on this keyboard were disappointing. We were unable to perform a theoretical analysis of this keyboard since we could not find any theoretical models to do so. Our one-time test with 5 volunteers led to a speed of 30 characters per minute or about 5 words per minute. Therefore, we looked for ways to develop more intelligent designs of keyboards. The rest of the paper discusses our new approach.

4 Soft Keyboard Development Using Optimization Techniques

We attempt to address the problems of Brahmic script text input by applying a combination of techniques previously used for the development of English keyboards and those design choices already used for Indic language layouts. We look at two approaches for improving input potential: layouts based on character frequency, and the use of machine learning techniques for character placement.

Keyboard layouts based on alphabetical ordering make the initial learning of the keyboard simpler, however, it is a tradeoff for efficient input later. Based on the improvements gained in English keyboards by organizing layouts by unigraph and bigraph frequency, we theorize that a similar approach will be equally beneficial for Brahmic script keyboards. The Emille Corpus for Assamese (Baker et al. 2003) was analyzed in order to develop tables of unigraph and bigraph frequencies. In addition initial research into ligature frequencies has been performed on the same corpus (Baker et al. 2003). Although ligatures have not yet been integrated into the soft keyboards being developed, future research into this is expected.

4.1 Design Choices

In creating soft keyboards, several design choices were made that may affect input speed. Some of these are discussed here.

In all of the keyboards designed, the spacebar is fixed below the grid of characters. When determining the distance between a digraph that contains a space, the center of the spacebar is used as the location the user would choose when typing. Although optimum input speed is obtained when the user always chooses the shortest path, this cannot be expected. Zhai et al. estimated that, given the choice of four spacebars, users chose the optimum space bar only 38% – 47% of the time (Zhai et al. 2000). Therefore, in order to have a conservative estimate of the upperbound for input time, perfect travel to and from thespacebar was not assumed. In addition, this decision avoids underestimates in movement calculations from "free-warping" in which the stylus enters a spacebar in one location and leaves it in an unrelated location, a common error in soft keyboard evaluation (Zhai et al. 2000).

We have chosen to create rectangular 'grid' keyboards in which each character occupies a square. This is the most common layout, however the Metropolis Keyboard (Zhai et al. 2000) and select others use hexagon keys and irregular, honeycomb shapes for the keyboard. Our decision to use a rectangular layout for all of our designs is a desire to have a similar layout among all stages of development in order to best compare them. It is believed that rectangular layered menus will be most efficient because square hierarchical menus have been shown, both theoretically and empirically, to be the most efficient type of menu selection (Ahlström et al. 2010). A desire to have a rectangular keyboard with square keys at one stage necessitates the need for a similar layout and key shape at all stages.

4.2 Genetic Algorithm Designs

Genetic algorithms are a technique in machine learning that is derived from the principles of natural selection. A genetic algorithm is composed of three major parts: a population of potential solutions, a fitness function which evaluates those solutions, and a method for reproducing and changing the population of potential solutions over time. In a given generation, each individual within the current population of potential solutions is evaluated by the fitness function and given a score. The higher an individual's score, the more fit, or closer to the optimum solution, they are considered to be. A new population of potential solutions is then created, in which the most fit individuals continue to survive and new individuals are created from fit individuals in the previous generation. These new individuals can be a combination of previous individuals, or can be developed by mutating previous individuals. Many iterations of this occurs until the solution, or an approximate for it, is found.

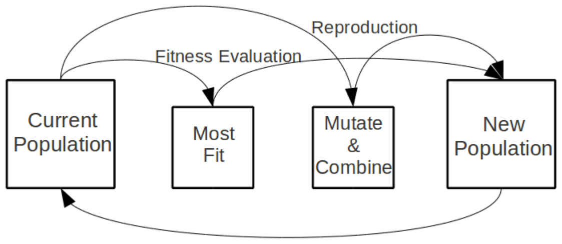

Figure 4: Evolutionary cycle of a population of chromosomes in a genetic algorithm.

In developing a genetic algorithm for soft keyboards, each individual in the population, called a chromosome, is a different layout for a keyboard. Each chromosome contains a number of genes equal to the number of characters in the alphabet being considered. Each gene is unique. In the initial genetic algorithm, a simple one-layer, rectangular keyboard was developed. Each chromosome was given a score in its fitness function that represented the inverse of its mean time to type a character in that layout, as calculated by Fitt's Law. Thus chromosomes with lower mean times were given higher fitness scores. The fastest layouts were kept for the next generation, and chromosomes were chosen for gene mutations (swapping the locations of two characters) and combinations with other chromosomes (in which part of one layout was adopted by another layout) with likelihood of being chosen proportional to their fitness. These newly created chromosomes as well as the best performing chromosomes become the population to be evaluated in the next round.

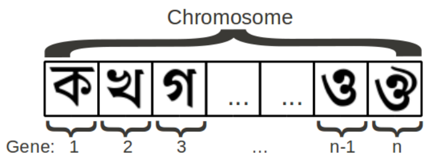

Figure 5: Example chromosome for an Assamese keyboard in which each character in the Assamese alphabet is a gene in the chromosome. In an $a \times b$ rectangular keyboard, the first a genes represent the first row, and the second a genes represent the second row, etc.

In our genetic algorithms, we allowed the population of keyboards to continue evolving until a single layout was considered the most fit in the population for a predetermined number of continuous generations.

There are many variables in genetic algorithms that can be adjusted, including but not limited to: the number of chromosomes in a population, the number of generations the genetic algorithm runs for, the number of required stable generations (one chromosome is consistently the most fit), the chance of mutation, and how many chromosomes were preserved between generations. There are too many variables to fully test all possibilities. When determining what values to use for the chance of mutation, we tested a range of percentages between .001 and .15 and found our best results with a chance of approximately .08. This is a higher mutation rate than many genetic algorithms use, and further testing may show that better or faster results are found with a different mutation rate. However, we used mutation rates of .08 for most of our tests whose results are reported in this paper. We also chose to preserve approximately 10% of each population.

We chose to focus our attention on varying the number of chromosomes in a population and the number of required stable generations. For each keyboard designed using genetic algorithms we varied the population size between 10 chromosomes and 1000 chromosmes. Although the results varied, generally it seems that the best performing keyboards were developed with larger population sizes. The number of required stable generations were varied between 10 and 100. A couple of tests were run with a stable generation requirement of 250 and 500 and the results were not any better. Having a high number of stable generations didn't seem to improve the results. Most of the best keyboards had a stable generation requirement of 15 to 35. Although many tests were run for each type of keyboard developed, only the best keyboards from each category are reported in this paper; we note the number of chromosomes used to develop that keyboard as well as the number of stable generations required.

5 GA-Based Flat English Soft Keyboards

The keyboards previously designed for Brahmic scripts have two downfalls as comparators for our keyboards: they have only been evaluated by human volunteers, and there has been little attempt to optimize them. As a result, in order to determine that the techniques we use to design keyboards are indeed designing theoretically efficient layouts, English keyboards were designed and evaluated in parallel with Brahmic keyboards. This allowed us to see whether the techniques create keyboards that are competitive in a language where much research for optimization has taken place. If these keyboards are theoretically competitive in English, we hypothesize that the same techniques applied to the development of Brahmic keyboards will also create keyboards that will be competitive in terms of efficiency even though no previous work on theoretic analysis of input speed has previously been performed.

	j	z			
q	u	g	m	p	x
k	c	n	i	o	v
b	l	a	e	r	f
y	d	s	t	h	w
space					

Figure 6: English soft keyboard developed using genetic algorithms. This keyboard has a predicted upper-bound of 38.6 wpm based on Fitt's law. While this is not as high as other upper-bounds, the suspected difference is the placement of the spacebar across the bottom of the keyboard rather than including it as a gene in the chromosomes in the genetic algorithm.

5.1 Fitness Function Used: Fitt's Law

Fitt's Law (Fitt 1992) is a technique used for evaluating a theoretic upper-bound of words per minute (wmp) in stylus-based input systems. While Fitt's Law has been widely used for evaluation of soft keyboards in English, it has not previously been applied to Brahmic script keyboards. It calculates the mean time in seconds to type a character, which can then be used to determine wpm (Zhai et al. 2000).

Fitt's Law finds the average time to move between a character i and a character j in an alphabet with n characters by looking at the distance apart the characters are, D_{ij}, as well as the frequency with which that digraph occurs, P_{ij}. Given a width W_j for each key j, and an index of performance IP, the mean time in seconds to type a character is:

$$\bar{t} = \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{P_{ij}}{IP}\left[log_2(\frac{D_{ij}}{W_j}+1)\right]$$

Previous work in the field uses an IP of 4.9 (MacKenzie and Zhang 1999) (Raynal and Vigouroux 2005) (Zhai et al. 2000). In order to maintain consistency in evaluation, our calculations also use this value.

In the tests performed on the designed English keyboards, the standard of five characters per word is used in our computations. Thus, given the mean time in seconds, $\bar{t}$, for typing a character, the calculation for wpm is $wpm = \frac{60}{5\bar{t}}$.

The fitness function used to develop both English and Assamese flat keyboards was an inverse of Fitt's Law. In order to maximize the input speed of the user, our fitness function sought to minimize the mean time to travel between characters. In other words, the fitness function evaluated each layout accoriding to Fitt's Law. The higher the Fitt's score the keyboard received, the greater the mean time to travel between characters, and the lower the fitness score given to the keyboard. The higher the fitness score of a keyboard, the more likely it was to be kept for the next generation and to be selected for crossover and mutation.

5.2 Evaluation

The only soft keyboards developed by us for English that we have tested were done using a genetic algorithm that employed Fitt's Law as the fitness function. These keyboards were designed to parallel the most basic Brahmic keyboard in which only one layer is used. While our designed English keyboard is not the fastest according to Fitt's Law, it is competitive with other keyboards. Although its theoretic upper-bound is less than 40 wpm, as compared to 43.1 wpm (Zhai et al. 2000) and 46.4 wmp (Raynal and Vigouroux 2005), there is a considerable enhancement over using a standard QWERTY layout as a soft keyboard. We hypothesize that the reason our keyboard is slower is that we used a fixed space-bar at the bottom of the keyboard rather a central button (Zhai et al. 2000) or including multiple space buttons (Raynal and Vigouroux 2005). The result is an increase in the average travel distance to and from the space-bar, the most frequently occurring character. A test with human volunteers was

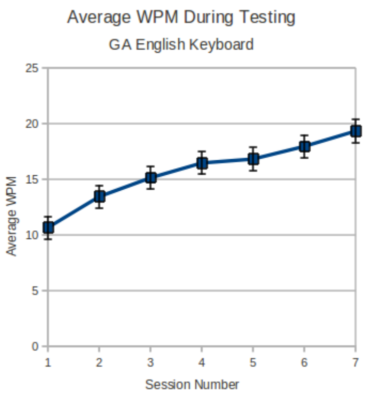

Figure 7: Graph of volunteers' average improvement over multiple ten-minute sessions at typing on the English keyboard designed using genetic algorithms. Average user input speed is increasing with each session.

carried out to show that the typing trend was approaching the expected words per minute as predicted by Fitt's Law. Ten volunteers typed random phrases on the keyboard during ten minute sessions over a period of a week. The phrases used were from MacKenzie and Soukoreff's published collection (MacKenzie et al. 2003). This test was not intended to determine input speed of long-term users, but to determine if the keyboard is easily learned and to show that input speed does approach the predicted speed. The results suggest that user's typing speed does increase and will continue increasing after long term use. Therefore, we claim that the genetic algorithm used to design this keyboard, as well as using Fitt's law to evaluate it, are both valid for developing and evaluating soft keyboards, and we will apply them to Brahmic scripts as well.

6 GA-based Flat Brahmic Keyboard

As a basis to compare the results of our genetically designed keyboards to, we first evaluated a flat, rectangular, alphabetic keyboard. In order to maintain the comparison, the diacritics were also added to the keyboard. Several layouts, which varied in number of columns and rows, were evaluated using Fitt's Law. In addition, the placement of the diacritics was adjusted. In all trials, the vowels and consonants were kept in alphabetic order and the diacritics were kept in a group, but the input speed was calculated with three different layouts: with the diacritics appearing before the vowels, after the vowels, and after the consonants. The fastest resulting alphabetic keyboard, shown in Figure 8, was an 8×8 square with the diacritics appearing after the consonants. An evaluation using Fitt's Law predicts an input speed of 25.06 wpm.

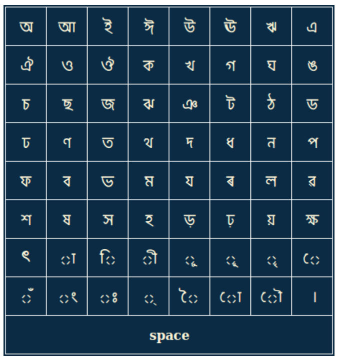

Figure 8: Alphabetically Organized Keyboard for Assamese with diacritics located after consonants. This keyboard is the predicted fastest alphabetic keyboard we tested and has a theoretic input of 25.06 wpm.

The first Brahmic script keyboard designed using genetic algorithms was a simple, single-layer keyboard for Assamese. The keyboard maintained the 8×8 grid of the 64 consonants, vowels, and diacritics that was used in analyzing an alphabetically organized keyboard. It is an improvement on the alphabetic keyboard, as its design considers the bigraph frequencies of Assamese and attempts to minimize the travel distance between characters in frequently occurring bigraphs. It was designed using a population of 1000 layouts which were allowed to evolve until a single keyboard had been the “most fit” for 25 generations. Although many tests were performed with varying numbers for population size and required stable generations, this was the best performing keyboard we found. The resulting keyboard has an expected input of 34.23 wpm according to Fitt’s law, a promising initial result. When designing `GAG1` and `GAG2`, Raynal and Vigouroux used populations of 20, 000 chromosomes and continued their algorithm until a single keyboard had been the most fit for 500 generations (Raynal and Vigouroux 2005).

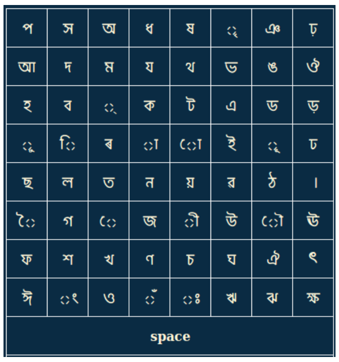

Figure 9: Assamese soft keyboard developed using genetic algorithms. This keyboard has a predicted upper-bound of 34.23 wpm based on Fitt’s law.

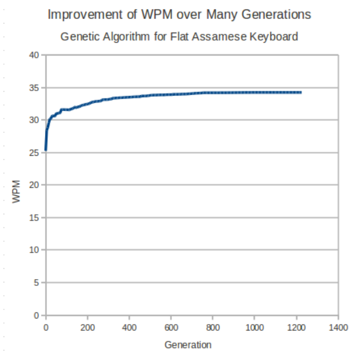

Figure 10: A graph of the growth of the improvement of the most fit keyboard in the genetic algorithm. There were 1000 keyboard layouts in each generation and the algorithm continued until the population had been stable for 25 generations.

In determining words per minute for Assamese soft keyboards, several measurements had to be obtained. The Emille Corpus for Assamese was used to calculate frequencies for all unigraphs and bigraphs. In addition, it was used to obtain the average number of characters per word in Assamese, which was calculated to be approximately six (Baker et al. 2003).

7 Hierarchical GA-Based Brahmic Keyboard

The initial use of genetic algorithms for designing soft keyboards for Brahmic scripts based on character frequency is very promising. A second technique tried for improving input speed is the development of multi-layer keyboards. A multi-layered keyboard allows menus or extra keys to be placed on top of an original keyboard. Genetic algorithms have previously been shown to produce high efficiency in the development of hierarchical menus (Matsui and Yamada 2008). While hierarchical menus and layered keyboards have some differences in design, the techniques used to evaluate them will be similar. For example, a two layer keyboard in which the second layer is accessible by rolling over characters in the first layer can be represented by a two-level menu.

7.1 Diacritic Menu as Second Layer

In an attempt to reduce the number of keys on the screen as well as improving input speed, the technique of adding a diacritic menu to each consonant was tried, When a consonant on the main level of the keyboard is selected by pressing the mouse button down, a diacritic menu appears around the selected consonant. The user can either release the mouse button on the chosen consonant to type a bare consonant, or they can drag the mouse and release it above one of the diacritics to add it to the consonant.

The diacritic menu allows selection of the 12 most frequently occurring diacritics. They were hand-placed in the diacritic menu such that the four most frequent diacritics appear immediately above, below, and to the side of the selected consonant. The next four most frequently occurring diacritics appear in the location immediately diagonal from the selected consonant. This is intended to allow the fastest possible selection time. Future work may include using optilization techniques to place the vowels, rather than hand-placing them.

The diacritics appear in the same location

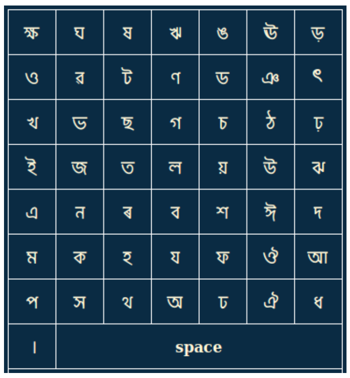

Figure 11: A layered keyboard for Assamese. Consonants and vowels are organized on the bottom layer using a genetic algorithm.

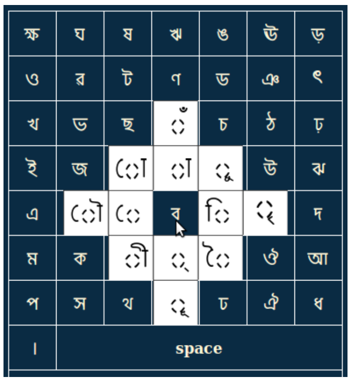

Figure 12: The second layer appears when a consonant is pressed down. The most common diacritics are available to be added to the consonant. They always appear in the same location in the diacritic menu.

for each consonant regardless of frequency with which they are used together. This is anticipated to facilitate faster learning of the diacritic menu by minimizing the visual search time for each consonant and decreasing the number of locations that must be memorized by the beginner in order to become an expert.

Diacritic sub-menus are not new, they have been widely used to decrease the number of visible keys. However, to improve upon this idea, we combined the ideas of diacritic menus and the organization by frequency of a base layer of consonants and vowels. We designed a genetic algorithm that took into account the diacritic layer with its fixed diacritic locations within that layer. The idea behind the algorithm is that if there is a frequently occurring trigraph *abc* with a diacritic as the central character, the best location for the third character, *c*, is as close as possible to directly below the diacritic *b* that appears in the menu off of *a*.

Our genetic algorithm was run with a population of 500 individuals with a stable generation requirement of 15. The resulting keyboard, seen in Figure 12, has an expected theoretic input speed of 40.24 wpm. This is a significant improvement over the single-layer, alphabetically organized Assamese keyboard.

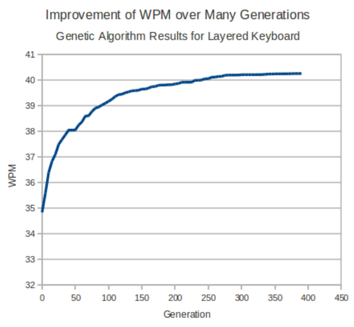

Figure 13: A graph of the growth of the improvement of the most fit keyboard in the genetic algorithm. There were 500 keyboard layouts in each generation and the algorithm continued until the population had been stable for 15 generations.

For comparison, a diacritic layer was added to the alphabetic keyboard. The resulting keyboard had an expected input of 33.94 wpm.

7.2 Vowel Menu

Assamese vowels are used infrequently, the most frequent character occuring only 1.36% of the time. Because of this, we hypothesized that the input speed could be increased by placing all of the vowels in a separate vowel menu. A single button that includes all of the vowels is a common organization technique in previous Brahmic keyboards. This technique has the benefit of reducing the number of characters on the board to be memorized as well as allowing the remaining characters to be closer together. Immediately, it seems that it would improve the theoretic input speed according to both Fitt's Law and Hick-Hyman's law. To test this, the vowels were placed in set positions, based on frequency just like the diacritic menu, in their own menu on the bottom left of the screen, shown in Figure 14.

A genetic algorithm was designed and run to create a keyboard with only consonants in the base layer, a second layer of diacritics attached to each consonant, and a vowel menu button.

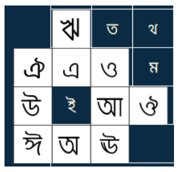

Figure 14: Vowel menu appears around the most commonly used vowel. The vowel menu was located in the lower left of the keyboard.

The keyboards designed using this algorithm were evaluated with the same methods used to evaluate the diacritic menu. However, the best keyboards developed using the vowel menu had, on average, an expected input rate that was 5 wpm less than those expected for the keyboards that had no vowel menu. The keyboard with the fastest expected input rate that included a vowel menu was 35.6 wpm. While further research into different layouts of the vowel menu and different locations for it may create a keyboard that has an input expectation similar to those without a vowel menu, 5 wpm is a significant gap to bridge.

7.3 Fitness Function Used

While an inversion of Fitt's Law was an effective fitness function for a single-level soft keyboard, multi-level keyboards need a different one. Multiple layers means that traversal time between layers needs to be taken into account, both in evaluation and in determining the most fit keyboards. In order to do this, we applied the ideas from the evaluation of hierarchical menus performed by (Ahlström et al. 2010). They built on a technique for analyzing the search and selection time of items in a hierarchical menu from (Cockburn et al. 2007) that combines Fitt's Law, Hick-Hyman's Law, and the Steering Law and applied it to hierarchical menus in which each menu and submenu is a square grid of the available options. The rectangular, multi-level soft keyboards we designed are an application of a rectangular hierarchical menu, and so their techniques for evaluation were used in the fitness function of our genetic algorithms (Cockburn et al. 2007).

Hick-Hyman's Law (Seow 2005) predicts the amount of time a person will take to make a decision given the number of choices they have. The user's reaction time is modelled by

$$T = b\sum_{i=1}^{n} p_i \log_2(\frac{1}{p_i} + 1)$$

where n is the number of choices and p_i is the probability of that choice being chosen. A user must make a decision about what character to press when their selection changes. The Hick-Hyman law can be applied to the original, base-layer of the keyboard as well as to the diacritic second layer.

The Steering Law claims that the time to move the cursor through a constrained two-dimensional "tunnel" to move from one level to the next is a linear function of the ration between the tunnel length and width. The Steering Law is very important in classic hierarchical menus and pie menus. When the hierarchical menu is a square grid, however, the ratio reduces to one and the time to move into a new level approaches the predicted time to move between two points as predicted by Fitt's Law (Cockburn et al. 2007). The diacritic menu that appears around the selected consonant is mostly square, but includes some outcroppings. The Steering Law does not need to be taken into account when selecting consonants and vowels on the base-layer.

The fitness function used in the our development of Assamese soft keyboards with layers attempts to minimize the sum of the time spent searching for the next character (Hick-Hyman's Law), moving between locations (Fitt's Law) and moving between layers (The Steering Law). The fitness function gets the mean time to type a character by summing three different "types" of ways a character can be typed. There are two types of characters that can be typed. Base characters reside in the bottom layer of the keyboard and diacritics appear in the second level. The three types of inputs that must be taken into account are a base character typed after another base character, $\bar{t}_{bb}$, a base character followed by a diacritic character, $\bar{t}_{bd}$, and a diacritic character followed by a base character, $\bar{t}_{db}$.

$$\bar{t} = \bar{t}_{bb} + \bar{t}_{bd} + \bar{t}_{db}$$

The goal of the fitness function, as before when the average time was computed only with $\bar{t}_{bb}$, is to reward chromosomes that have smaller $\bar{t}$ values with higher fitness scores.

In calculating $\bar{t}$, the three averages are computed as follows. $\bar{t}_{bb}$ is the sum of Fitt's Law and Hick-Hyman's Law for each pair of characters on the base layer. Steering's Law does not apply to the base layer because the ratio between the width and height of the "tunnel" being traversed is 1. $\bar{t}_{bd}$ is the sum of Hick-Hyman's Law and the Steering Law for each diacritic that can be chosen from each base character. $\bar{t}_{db}$ is more complicated because it requires calculating the distance from each diacritic to each base character, and all possible locations of each diacritic must be taken into account. This requires trigraph analysis of each base-diacritic-base combination. $\bar{t}_{db}$ is the sum of Hick's Law and Fitt's Law for all base-diacritic-base trigrams.

7.4 Implementation of Text Prediction

Text prediction has been shown to help improve input speed in several input domains such as texting and as an input aid for persons with disabilities (Wobbrock and Myers 2006). Although Brahmic script input does not fall under those domains, it has been hypothesized that well designed and implemented text prediction could increase the input speed of the average computer user (Anson et al. 2006). Although text prediction has not caught on as a general typing aid because it requires that the user switch their focus from the keyboard on which they're typing to the prediction suggestions, this may not be as large of a problem when typing with soft keyboards. The study performed by Anson et al. suggests that text prediction algorithms used in conjunction with soft keyboards actually improve the input speed. They hypothesize that this is because the predictions are physically closer to where the typist is focused, and so requires less time to use (Anson et al. 2006). The result of this study motivates us to incorporate text prediction into our Brahmic keyboards.

Unfortunately, there has been little theoretic work on the time required to consider the predictions suggested. Since the typist will glance at the list of suggestions after every character they type regardless of whether the word they are writing appears there, it is apparent that poorly designed and implemented text prediction could easily decrease input speed. Empirical tests have shown that vertically listed suggestions tend to be faster, more suggestions increases the input time, and that using n-grams helps with prediction suggestion when n, the number on words used, is not too large (Garay-Vitoria and Abascal 2005).

Our implementation of text prediction uses unigram, single word, and n-gram, multi-word, prediction. Once a user has begun typing a word, the three most frequently occuring words beginning with the letter the user has typed are displayed for them to choose from. If the user does not select one of those words but instead types a second character, the most frequently occuring unigrams beginning with those letters are suggested. This continues until the user has either typed the full word or selected a word from the suggestion list. When a word has been completed by either method, three words are suggested as the next word. Next word predictions are made by using bigrams and trigrams, of which the next word is the last word in said n-grams. The three most frequently occuring trigrams using the two most recently typed words as the first and second word in the trigram are suggested. If there are no trigrams, then bigram prediction, with the most recently typed word as the first word in the bigram, is used instead. Using an n-gram model is hoped to improve accuracy of predictions.

The frequencies for both unigrams, bigrams, and trigrams are based on our analysis of the Emille corpus for Assamese (Baker et al. 2003). Our predictions are listed vertically on both sides of the keyboard, as can be seen in Figure 15. The top word has the highest frequency and the bottom word has the lowest frequncy of those listed. Locating the suggestions on both sides of the keyboard is hoped to minimize gaze and travel time for the typist, as they can use whichever prediction list is closest to the last character they typed.

Our current work regarding text prediction lies in the realm of using the previously typed words in a document to predict the

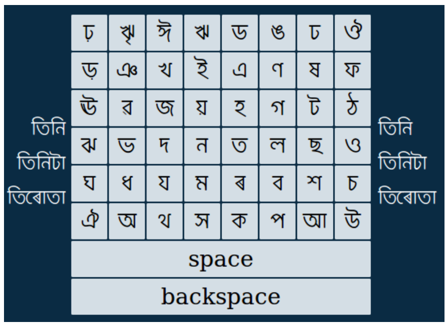

Figure 15: Text prediction suggests three words beginning with the letters already typed that have the highest frequency of occurence among all unigrams. The suggests appear on both sides of the keyboard.

words that will be typed. Typists tend to reuse the same vocabulary, at a rate of approximately 70% (Tanaka-Ishii et al. 2003). Incorporating the frequency of use of certain words by a particular user would make the word suggestions more accurate. Many previous solutions to the problem of user-affected frequencies have used a personalized dictionary that the typist can add words to. However, this is not used as often as it could be because of the inconvenience of adding words. In addition, this technique does not account for a difference in personal vocabulary frequency. A better approach would allow for dynamic frequencies in the text prediction.

8 Future Work

Despite the progress that has been made, there are still many tactics and techniques that could be tried to develop a better keyboard.

Success with genetic algorithms suggests that other optimization techniques can also be applied to soft keyboard development. Such techniques as ant colony algorithms and particle swarm optimization will be investigated and applied to Brahmic soft keyboards.

Additionally, future research into the implementation of gestures, in which a certain movement with the stylus or mouse indicates a different level, will be investigated. Using gestures as a second or third layer above the original keyboard may allow for easier and faster vowel diacritic selection. It is expected that ultimately the best keyboards will be a combination of these tactics.

Another technique that may improve input speed is the inclusion of ligatures. Ligatures take more time to input than other characters because a single ligature requires three or five characters to be selected on the keyboard for a single character to be displayed in the output. Inclusion of a few commonly used ligatures may increase input time by decreasing the number of key strokes that must be made.

Perhaps the most important step in this research will be empirical testing. Tests with volunteers on the layered and predictive Assamese keyboards will be performed. Testing the Assamese keyboards will reveal whether they are easy to learn and use by people. Although they may be theoretically efficient, keyboards are not worthwhile if they're not usable. In addition to the benefits of empirical testing on the genetically engineered Assamese keyboards, tests will provide a basis to compare the results of testing the keyboards that employ text-prediction. Because we have not yet found a way to theoretically analyze the input speed of keyboards with text prediction, empirical testing is essential. Such a test would compare the input speed of volunteers on one of the previously mentioned keyboards as well as their speed on the same keyboard that includes text prediction.

Assamese is not the only language that could benefit from an optimized keyboard. We are looking into developing keyboards, using the same techniques as used with Assamese, for other Brahmic languages such as Gujarati and Kannada. Although many languages share characters, differences in frequency of use of those characters will cause a keyboard optimized for one language to not be very efficient for another language. The benefit of a soft keyboard is that anyone using it would be able to switch the layout to the one they are most familiar with, it is not necessary to have standard character placements for all users in a single county or area.

9 Conclusion

Soft keyboards have the benefit of not being limited to a specific layout as physical keyboards are. They are versatile in shape and can be varied by language in order to be most efficient for whoever is using them. Theoretic analysis using techniques such as Fitt's Law, Hick-Hyman's Law, and Steering's Law were used to analyze various layouts, and showed that reorganizing keyboards such that characters in frequently occuring bigraphs appear near one another results in a keyboard with a higher expected input rate.

Table 1: Summary of Expected Input Speeds

Keyboard Type	Expected WPM
Flat Alphabetic	25.06
Layered Alphabetic	33.94
Flat GA-Designed	34.23
Layered GA-Designed	40.24

Assamese text input can be improved from an expected input of approximately 25 wpm using an alphabetically ordered keyboard to approximately 34 wpm merely by reordering the keys. This improvement has also been seen in English, and can likely be applied to other Brahmic scripts. In addition, applying this technique of organization by frequency using through genetic algorithms to layered keyboards can lead to another significant increase in expected input to approximately 40 wpm. It is interesting to note that alphabetic keyboards that apply layering techniques are still outperformed by a flat keyboard organized by frequencies. This suggests that the most important technique for improving input speed is frequency organization. This would mean that having efficient keyboards designed for each language is more important than a general keyboard design for a shared script. The beauty of soft keyboards is that they allow this versatility. Once efficient keyboards have been developed for a language, it can be available for speakers of that language everywhere they go.

References

Ahlström D., Cockburn A., et al., *Why it's Quick to be Square: Modelling New and Existing Hierarchical Menu Designs.* Dallas, PA, USA: College Misericordia, 2010.

Anson D., Moist P., et al., *The Effects of Word Completion and Word Prediction on Typing Rates Using On-Screen Keyboards.* Atlanta, GA, USA: CHI 2006.

Baker P., Monmarche N. et al., *EMILLE Corpus: Assamese.* European Language Resources Association, 2003.

Cockburn A., Gutwin C., and Greenbrug S., *A Predictive Model of Menu Performance.* San Jose, CA, USA: CHI, 2007.

Colas S., Hardy A. et al., *Artificial Ants for the Optimization of Virtual Keyboard Arrangement for Disabled People.* Tours, France: Laboratoire d'Informatique de l'Universite de Tours, 2008.

Columas, F., *The Writing Systems of the World*, Cambridge, MA: Basil Blackwell, 1990.

Fitts, P. M., *The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement*, Journal of Psychology: General, Volume 121, No. 3, pp. 262-269 (reprint of the same article from Journal of Experimental Psychology, 47, 381-391, 1954), 1992.

Garay-Vitoria N., Abascal J., *Text Prediction Systems: A Survey.* Donostia, Spain: KReSIT IIT Bombay, 2005.

Joshi A., Parmar V. et al., *Keylekh: A Keyboard for Text Entry in Indic Scripts.* Mumbai, India: KReSIT IIT Bombay, 2004.

Krishna A., Ajmera R., Halarnkar S. and Pandit P., *Gesture Keyboard - User Centered Design of a Unique Input Device for Indic Scripts.* Mumbai, India: HP Laboratories, 2005.

MacKenzie S. and Soukoreff R. W., *Phrase Sets for Evaluating Text Entry Techniques.* Ft. Lauderdale, FL, USA, 2003. 1992.

MacKenzie S. and Zhang S. X., *The Design and Evaluation of a High-Performance Soft Keyboard.* Pittsburgh, PA, USA, 1999.

Matsui S. and Yamada S., *Genetic Algorithm Can Optimize Hierarchical Menus.* Florence, Italy, 2008.

Rathod A. and Joshi A., *A Dynamic Text Input Scheme for Phonetic Scripts like Devangari.* Mumbai, India: IIT Bombay, 2002.

Raynal M. and Vigouroux N., *Genetic Algorithm to Generate Optimized Soft Keyboard.* Portland, OR, USA, 2005.

Seow, S., *Information Theoretic Models of HCI: A Comparison of the Hick-Hyman Law and Fitts' Law*, Human-Computer Interaction, Volume 20, pp. 315-352, 2005.

Shanbhag S., Rao D., and Joshi R. K., *An Intelligent Multi-Layer Input Scheme for Phonetic Scripts.* 1em plus 0.5em minus 0.4emMumbai, India, 2002.

Sowmya V. B. and Varma V., *Design and Evaluation of Soft Keyboards for Teluga.* Pune, India, 2009.

Tanaka-Ishii K. et al., *Acquiring Vocabulary for Predictive Text Entry through Dynamic Reuse of a Small Corpus.* Tokyo, Japan: University of Tokyo, 2003.

Venolia D., and Neiberg F., *T-Cube: A Fast, Self-Disclosing Pen-Based Alphabet.* Boston, MA, USA: Human Factors in Computing Systems, 1994.

Wobbrock J. O., and Myers B. A., *From Letters to Words: Eficient Stroke-based Word Ccompletion for Trackball Text Entry.* Portland, OR, USA: ASSETS, 2006.

Zhai S., Hunter M., and Smith B. A., *The Metropolis Keyboard - An Exploration of quantitative Techniques for Virtual Keyboard Design.* San Diego, CA, USA: IBM Research Center, 2000.