

OPTIMIZATION OF A TOUCH SCREEN
SOFT KEYBOARD FOR SPEED AND ACCURACY

by
RICHARD DAVISON

Advisor
DAN PLANTE

A senior research proposal submitted in partial fulfillment of the requirements
for the degree of Bachelor of Science
in the Department of Mathematics and Computer Science
in the College of Arts and Science
at Stetson University
DeLand, Florida

Spring Term
2011

TABLE OF CONTENTS

Table of Contents	2
Abstract	3
1. Introduction	3
2. Problems to Consider	4
2.1 Learning Curve	4
2.2 Keys Per Input Device	5
2.3 Tactile Feedback	5
2.4 Key Strokes Per Character	6
2.5 Words Per Minute	7
3. Related Work	7
3.1 Multi-tap	7
3.2 Full QWERTY Keyboard	8
3.3 Optimized Full Keyboard	9
3.4 Chording	10
3.5 Handwriting Recognition	11
3.6 Gestures	12
4. Possible Solutions	13
4.1 Full Keyboard Optimized for Two Thumbs	14
4.2 Chording Keyboard Optimized for Two Thumbs	15
5. Our Research	17
5.1 Layout Generation	18
5.2 Selection	18
5.3 Crossover	19
5.4 Mutation	25
6. Fitness Functions	25
6.1 Individual Letter Fitness	26
6.2 “Race” Layout Fitness	27
7. Human Testing And Results	27
8. Future Work	28
9. Bibliography	29
Appendix A – Human Testing Handout	34
Appendix B – Raw Data Analysis	35

ABSTRACT

Mobile devices are becoming more mainstream in the lives of many people today, whether it be for business, school, or general entertainment. However, one problem which has manifested itself because of the shrinking of many of these devices is that of the keyboard. While many devices feature hardware keyboards, a growing number are opting for software keyboards, some of which feature touch responses and others which do not, each with varying accuracy. In this research, we developed a new chording input method for use with two thumbs, along with a search algorithm to generate layouts for said input method, to look at mobile keyboards from a different point of view.

1. INTRODUCTION

Perhaps no single personal device defines pervasive technology and the 21st century like the modern cell phone; however, for all the advancements, from the original 1973 Motorola DynaTAC to the cutting edge hardware found within modern smart phones, text input is perhaps the most underdeveloped facet of the modern cellular device. One of the many problems plaguing smart phone development involves the fact that it is impossible to determine exactly how the user will use the device according to their own distinct biological characteristics. However, the smart phone designer must conform to a specific standard, which, among cell phones is either Multi-tap or a fully fledged keyboard. As such, the user is forced to adapt to the limitations caused by these inefficient input methods.

Currently, Multi-tap input methods have persisted since the inception of the cellular phone; however, despite massive efforts to improve the efficiency and accuracy of text input, the only widespread method that profoundly increases input speed is the use of a full QWERTY

keyboard. The root of the problem lies in the fact that speed is attained at the expense of accuracy. Although a success in a sense -- bringing about the ability to compose emails on the go -- a full QWERTY keyboard still does not solve the accuracy problem. Fitts' Law demonstrates the trade-off between accuracy and speed. Consider two small, circular sheets of paper right front of you. You want to draw dots in the center of each piece by alternating between which piece you put dots on. As the two pieces of paper are nearby, you can pencil in a dot quite accurately and quickly within the center of each respective piece. As you move the pieces of paper further and further apart, the accuracy and speed diminish dramatically.

$$MT = a + b \log_2(2A/W)$$

Figure 1. Fitt's Law [32] where

- A is movement distance
- W is width
- a and b are regression coefficients

2. PROBLEMS TO CONSIDER

2.1 LEARNING CURVE

One of the biggest problems to consider when searching for better text input methods is the issue of the learning curve. People are for the most part content using what they already know, even if not effectively. To introduce a new input method would essentially be suggesting to become less productive in the short term with only possibly marginal benefits in the end. Additionally, if there are too many input methods, a fragmentation of all the text entry methods can occur, leading to frustrated users who are potentially unable to use perhaps their friend's device because the learning curve associated with the friend's favorite input method is not worth the effort to learn. As such, users will be most comfortable typing solely in their chosen input method than any other. One way to combat this scenario is to try and utilize QWERTY-esque

layouts. It has been shown that input methods that retain a certain degree of QWERTY-ness to them are much more easily navigated and learned than other methods. [33]

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

Figure 2. A Quasi-QWERTY layout for stylus input. [33]

2.2 KEYS PER INPUT DEVICE

A common element in methods that improve accuracy is that each input device is limited in the number of keys that it is responsible for. A traditional full size keyboard is able to achieve such high speed and accuracy because for the most part, each finger has no more than six nearby keys that it is responsible for, each of which being normally bigger than the user's fingers. Typing in excess of 100 words per minute on such a keyboard is in fact relatively common due to this fact. Examples of other input methods that attempt to minimize on key dependencies include chording, gestures, handwriting and voice recognition, and Multi-tap (to some extent). Each of these methods manages to lower the number of keys per input device, and/or place the keys in a very predictable and simple arrangement.

2.3 TACTILE FEEDBACK

Tactile feedback exists to assist the user in gauging how their actions are affecting a device. It exists in various forms, particularly, in the case of mobile devices, in the form of physical, pressure sensitive buttons. There appears to be a trend however, to create devices which are entirely touch screen devices. One advantage of an entirely touch screen input method is the

ability to customize the locations of the buttons because nothing has to be physically altered on the device. However, disadvantages include no longer knowing where the boundaries of the keyboard buttons are and therefore a loss in accuracy. To assist in this ‘transition’, a type of tactile feedback was developed, and is demonstrated by devices such as the LG Voyager, which respond to touch events with the activation of a little motor inside the device to give the screen a small jolt so the user is aware that she is in fact activating something on the screen, not dissimilar to a physical keyboard where the user can feel individual keys as she pushes them. Despite not being able to be relocated, physical buttons allow users to control more buttons because they can be felt and their locations can be estimated by touch. Physical buttons also have the advantage that they can have different textures on their surfaces. As an example, on an accordion, the left hand is responsible for pushing buttons much like a mobile device with a physical keyboard; however, the buttons on an accordion have different surface textures serving to guide the performer’s fingers to push the right button. The different textures give the player incredible control navigating the ocean of otherwise identical buttons. The player can always know where all the other buttons are relative to the one it is currently touching. This is in contrast to a cell phone without feedback, or otherwise rudimentary tactile feedback (a la the LG Voyager), which does not allow the ability to blind type on such a device, as an accordion player is able to navigate her instrument.

2.4 KEY STROKES PER CHARACTER

One way of determining how effective a certain input method works is to calculate the average key strokes per character (KSPC). [3] The KSPC is a measurement of how many raw key strokes it took to input some text, this includes backspaces, carriage returns, navigational buttons, shift, etc... For example, assuming there were no errors, the average KSPC of typing the

word *cat* on a Multi-tap keyboard would be the sum of the number of key strokes it took to produce each letter all divided by the number of letters in the word *cat*. The letter *c* takes three key strokes, *a* takes one key stroke, and *t* takes just one key stroke as well. This can be represented as:

$$\text{KSPC}_{\text{avg}}(\text{cat}) = (3 + 1 + 1) / 3 = 1.667$$

Ideally, the KSPC should be minimized to reduce the amount of slow physical motion on the device.

2.5 WORDS PER MINUTE

The words per minute (WPM) expresses the speed at which a person types. A word, in this sense, is internationally defined to be five characters so as to make it possible to compare the typing speed of speakers of different languages with different traditional word lengths. The goal is to maximize the possible WPM.

3. RELATED WORK

3.1 MULTI-TAP

The Multi-tap entry method is one of the most common text entry methods on mobile devices. It has gained popularity from its inclusion on early phone keypads, and has remained the default input method for many years. On mobile devices utilizing Multi-tap, each key has a specific number of letters assigned to it (normally three, on occasion two, four, or more). To access the first letter represented on a key, the user is required to push the key once, upon which

time the corresponding letter will be represented on screen. If the user wishes to access the secondary or tertiary letter represented by a single key, she simply taps the key two, three, or however many times represented by the position of the letter in relation to the first letter.

Multi-tap's persistence throughout the history of the mobile device can perhaps be attributed to its ease of use. Other advantages include the ability of the user to blind type using the Multi-tap method, because of the limited number of keys and the simple orientation of keys as adjacent to one another. In this way, Multi-tap does not require visual search (unless using a touch screen with no feedback). Despite its enormous popularity as a standard for text input on mobile devices, the method has its drawbacks. For example, to access a specific letter on a key, there is a chance that the user will have to press the key multiple times; therefore the KSPC is greater than one, and hardly ideal. Thus, typing using a Multi-tap device is potentially much slower compared to a QWERTY keyboard.

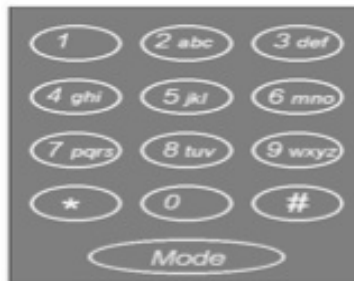


Figure 3. A Multi-tap keyboard [34]

3.2 FULL QWERTY KEYBOARD

The second most common text input method on mobile devices is a miniaturized version of the QWERTY keyboard popularized by the personal computer and typewriter. The ubiquity of the QWERTY keyboard allows users to avoid the steep learning curve of other more optimized layouts. Because of general user familiarity with the input method, users can generally achieve

much higher WPM than with other methods. Compared to Multi-tap methods, each character is represented by a single key, resulting in much lower KSPC in general, asymptotically approaching one keystroke per character, which is ideal. However, in translating the full QWERTY layout to the comparatively minuscule dimensions of a mobile device has also presented a host of problems for the user. In general, the sheer number of keys crammed into a small space makes the input layout difficult to navigate, resulting in unanticipated typing inaccuracies. Concurrent to this accuracy problem, the method requires that the user look at the keyboard while typing. Adjusting to the smaller size of a mobile keyboard, users generally employ their thumbs to type. This compares unfavorably to the tiered structure of a full size QWERTY keyboard, where multiple fingers are responsible for certain groupings of keys. For example, if a user is typing correctly, their left middle finger will be responsible for only three letters (E, D, and C), and their right index finger will control six letters in close proximity (J,U,Y,H,N, and M) on the other side of the keyboard, and so on for each additional finger. Now, compare this to a mobile keyboard where the thumbs are now responsible for entire “clouds” of keys -- perhaps even an entire half of the keyboard -- that were previously assigned to multiple fingers. This entails that blind typing will be extremely difficult if not impossible for almost everyone who uses a QWERTY layout on a mobile device.

3.3 OPTIMIZED FULL KEYBOARD

Optimized full keyboards are similar to a traditional full keyboard; however, they have been arranged to be more efficient in specific ways. In abandoning the somewhat arbitrary layout and form-factor of the traditional QWERTY format, the layout can be arranged in such a way to marginally increase speed and accuracy, but without solving the visual search problem (in other

words blind typing is still impossible). Advantages to optimized full size keyboards include much faster input speed than a QWERTY keyboard (assuming an input device that has been designed for the specific optimized layout). For example, the GAG layouts [1] locate the most common letters towards the center, and place the least common letters on the periphery of the input space. Per Fitt's Law, accuracy and speed show marginal increases due to the close proximity of the common keys to one another. However, because of the modifications to the standard layout, these new optimizations require a relatively high learning curve. Additionally, optimized keyboards may translate even more poorly to a smaller mobile device than a QWERTY keyboard if the common keys are organized on the basis of close proximity. For example, the usefulness of the previously mentioned GAG layouts decreases dramatically on a smaller device unless the user is using a stylus. If the user puts her thumb on a keyboard, covering a cluster of five keys, in this situation, the close proximity of the common letters actually becomes a liability and will most definitely decrease accuracy.

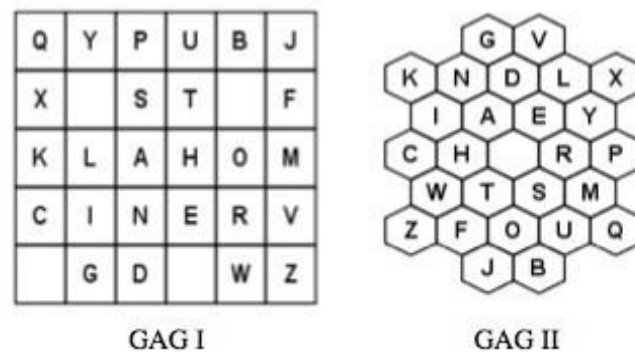


Figure 4. The GAG layouts optimized for English [1]

3.4 CHORDING

A chording text entry device seeks to redefine the way in which text input occurs. Rather than pressing a single key at a time, a chording device enables the user to press multiple keys at a

time. Different combinations of keys produce different letters, letter groupings, or even entire words. This can significantly reduce the keystrokes per character while keeping the device compact. Thus, blind typing is not only possible, but can be done quite quickly. The KSPC on such a device can be quite low, further increasing input speed. For example, experienced users of the Twiddler input method have been recorded typing in excess of 60 words per minute [36]. However, the format is held back by a litany of drawbacks preventing it from widespread adoption, chief of all being a clunky and unwieldy form factor. Current implementations suggest placing the input surface on the back of the mobile device, at the expense of form and awkwardness. Also, the orientation of the device as implemented on current cell phones could create user fatigue with prolonged use.



Figure 5. Twiddler keypad [36]

3.5 HANDWRITING RECOGNITION

Yet another new method for text input is handwriting recognition. The method entails taking advantage of the natural approach to writing with the human hand learned in grade school.

Ideally, the user should be able to instantly apply their knowledge of writing to the input space. Also, the method does not require visual search, and blind “typing” finally becomes possible. However, due to the limitations of artificial intelligence, particularly the difficulties of machines to correctly interpret the infinite nuances of individual human handwriting, these methods have not proven to be viable replacements for key based input. [26] When this method is implemented, the AI is required to “learn” the individual’s gestures and idiosyncrasies of the user, which is usually time consuming and imperfect. Even if we were to assume that a machine could have perfect handwriting recognition, the fact remains that people generally type much faster than they write. This relationship also illustrates the tradeoff between accuracy and speed.

3.6 GESTURES

Similar to handwriting recognition, gesture based input methods have been developed to take advantage of input via a stylus or finger. Certain implementations, such as 8pen [35] take advantage of computer recognition ability while eliminating the disadvantages associated with the inconsistencies of human handwriting. The 8pen method groups letters into four different zones around a center point. The letters are stacked so that crossing from one zone to another mimics the act of tapping multiple times on a key on a Multi-tap device, without the redundancy of multiple key strokes.

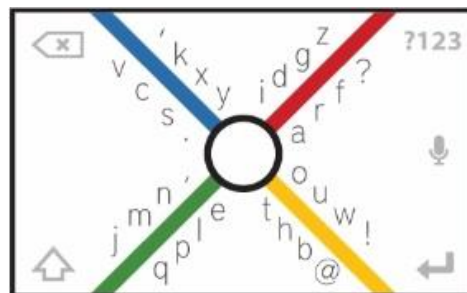


Figure 6. The 8pen input surface. [35]

Input in this fashion has the potential to be extremely quick and smooth, without the interruptions experienced while typing on a Multi-tap keyboard. In addition to increased speed, the method also offers the ability to blind type by placing every letter into only one of four zones. There are also additional gesture based input methods competing for adoption, such as EdgeWrite [9]. This method uses four keys orientated around each other in a square. Typing a letter consists of “drawing” the beginning and end of a letter's individual strokes across the four zones. Thus, each keystroke refers to an endpoint of a line segment. Such a method entails a high amount of keystrokes per character; however the simplicity may appeal to people with poor motor functions.

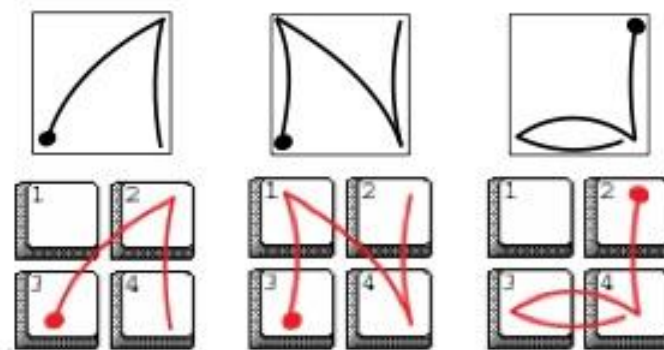


Figure 7. Writing the word *and* in EdgeWrite. [9]

4. POSSIBLE SOLUTIONS

The main issue with full keyboards as they are implemented today requires the thumbs of the user to be responsible for much more than they are capable of accurately typing. We believe an input method which minimizes keys and lays them out in a predictable pattern is most useful. Another possibility may be to use a full keyboard that is optimized for a pair of thumbs instead of a stylus.

4.1 FULL KEYBOARD OPTIMIZED FOR TWO THUMBS

On mobile phones, it may be most appropriate to want to create a keyboard layout that caters to two thumbs. To marginally increase accuracy and possibly speed, a keyboard layout could be devised to provide fewer ambiguities when the user is trying to type. In order to achieve this, we believe the most common letters should be placed far away from each other, and the letters of common digraphs (letter pairs) should be placed on opposite sides of the keyboard to alternate thumb use.

The reason for separating common letters is that when using a software keyboard on a small device such as an iPhone or Android, the keys are scaled down to fit on the device's screen. As a result, the keys very often become much smaller than the size of many people's fingers, which makes it difficult to type accurately. Certain advances have been made in software keyboard and linguistics research to autocorrect many erroneous key strokes by using dictionaries and probabilities to guess which words are being typed [7, 10, 18, 19, 30, 31], however, these methods have the serious limitation that when users try typing words which aren't in the internal dictionary such a proper nouns, abbreviations, or acronyms, the device may try to autocorrect to the nearest word, which may not necessarily be what the user was wanting to type.

Consider for example the placement of E and R in the QWERTY keyboard. Both letters appear quite frequently in most English text, however, by being adjacent to each other, a user with large or shaky hands may unintentionally press one for the other.

As for separating the letters of frequent digraphs into opposite sides of the keyboard, the availability of two pointers also has an effect in the accuracy of typing digraphs. For example, if one were to put two different colored stickers on a desk such that one comfortable rests under the

right hand, and the other under the left hand, by alternating strokes with both hands, both stickers could be rapidly and accurately be pointed at, whereas trying the same thing with just one hand drastically reduces both the accuracy and the speed of the pointing procedure. Due to this reasoning, we believe that by separating common digraphs to opposite sides of the keyboard such that one thumb presses the first letter, and the other thumb presses the second letter, one can achieve much better accuracy and speed while typing.

The disadvantages associated with a keyboard like this include the fact that it still fails to solve the accuracy problem, instead only improving it by a small margin. Additionally, the thumbs are still responsible for many more keys than they are able to effectively handle given the lack of appropriate tactile feedback. Finally, since there are so many keys, they will invariably be forced to cramp together to fit the form of the mobile device, hindering users with particularly large thumbs from being able to type any better than they already could with current existing methods.

4.2 CHORDING KEYBOARD OPTIMIZED FOR TWO THUMBS

Another possible solution to the different problems listed beginning on pg. 4 is to create a type of chording interface that caters to two thumbs. While not perfect, this method has the potential to drastically improve typing accuracy without sacrificing too much in terms of speed. Rather than placing all the keys on the screen at once, the letters are assigned five at a time to a key. To access the middle letter, the key can just be pressed on its own. To access any of the outer letters, the key must be pressed in conjunction with one of the black keys on the right side indicating the position of the outer letter. To access the modifier listed on the black keys, the black key with the corresponding modifier is pressed by itself. Users typing on this keyboard

would alternate between pressing one thumb at a time and two thumbs at a time to access letters and functions. The white keys would be controlled by the left thumb, and the black keys by the right (see Figure 8).

As far as solving the aforementioned problems, this keyboard has the potential to perform well. By limiting the number of keys per input device (the input devices being two thumbs), the thumbs gain a stronger sense of control. The keys themselves are no longer tiny and cramped, and are predictably spaced.

Although there is always a need for tactile feedback, the sheer lack of keys makes it much less a necessity than with other input methods with more keys per input device. Certainly, if this keyboard were implemented with physical buttons it would probably perform even better than on a plain touch screen device.

Additionally, despite the fact that two keys can be pressed in tandem to access a specific letter, pressing two buttons at the same time constitutes as just one key stroke. Since all of the letters are available without having to swap keyboards or press the same key multiple times (as in Multi-tap), this keyboard has a much lower KSPC value. Testing will have to be done to see how well this corresponds to the WPM, but it is possible this keyboard will be able to retain a high WPM value while significantly augmenting the accuracy.

Finally, several layouts can be tested to gauge the learning curve. The figure below depicts a possible quasi-QWERTY layout. While not perfect, a layout that tries to conform to the QWERTY standard will most likely result in a lower learning curve. For the new user, this will reduce the visual search required to begin using the keyboard. However, expert users might be interested in using a more optimized layout with possibly the most common letters situated in the

middles of the buttons, and perhaps common digraphs within the same button so that only the right hand has to change in order to type common letter pairs.

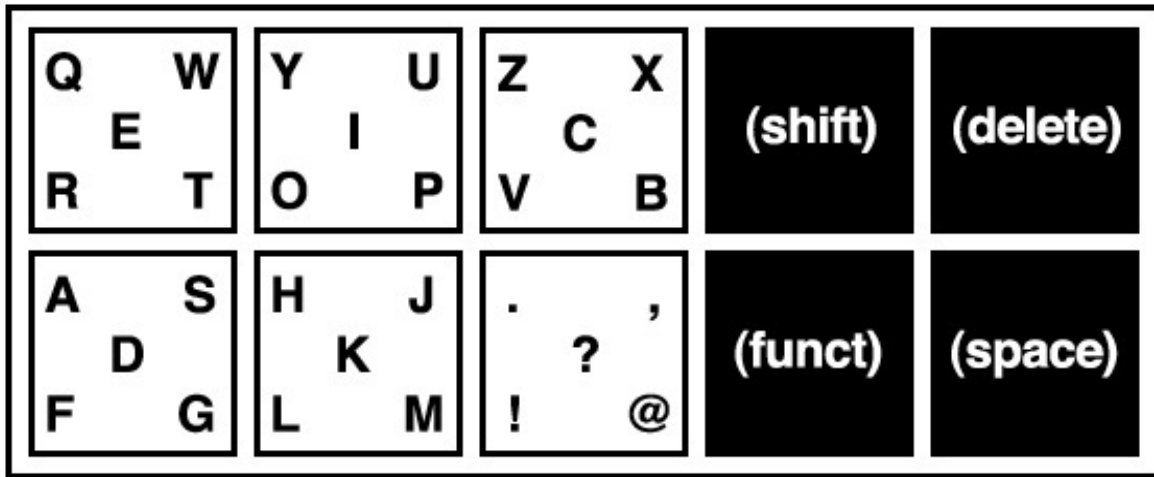


Figure 8. A possible two thumb chording keyboard implementation depicting a quasi-QWERTY letter layout.

5. OUR RESEARCH

For our research, we decided to implement the Chording Keyboard Optimized for Two Thumbs on the Android operating system. It was an excellent choice because Google allows us to change the global input methods, install programs without submitting the program for review (such as with iOS devices), as well as the fact that Android is a current popular operating system for mobile phones with widespread multi-touch support. We developed on the Motorola Atrix which was provided to us by the Math/CS department at Stetson University.

The research is divided into multiple sections. First, and most obvious, is the implementation of the actual program on the device, and the human testing. Additionally, we opted to test several letter layouts. The first, (QWERT), was to test a Qwerty-esque layout in which most of the letters were placed in a way to try to make the letters easier to find. The

second, (MITWS), was a handmade layout that used human intuition to try to create a layout that would minimize finger movement and follow some of the criteria listed in Section 6. The third and final layout, (ANEZS), was generated using a genetic algorithm, where the scoring was based on the criteria in Section 6 from a computer's point of view. The algorithm was allowed to run for long spans of time and was therefore able to make more comparisons based on actual numbers than MITWS. The next subsections deal with how the genetic algorithm was created and what it is comprised of.

5.1 LAYOUT GENERATION

After developing the key layout we needed to come up with a method to generate letter layouts that would perform well on the key layout. To do this we implemented a Genetic Algorithm (GA) that would use ideas taken from the biological process of reproduction that would breed new layouts based on the best qualities taken from the parent layouts. The high level steps involved in creating a genetic algorithm include: 1) Selection, which selects the candidate parents; 2) Crossover, to combine the parents to produce offspring; 3) Mutation, randomly mutate certain elements from the offspring. Additionally, we needed a fitness function to score a layout to determine how it was better or worse than any other layout. Due to the fact that our problem dealt with permutations as opposed to combinations, it became necessary to use more than one fitness function to score layouts: one to score the placement of the individual letters, and one to score the layout as a whole.

5.2 SELECTION

For our selection procedure, we simply chose to randomly pick two layouts. A natural selection constant is used as a base for the probability that the better of the two layouts is chosen.

The chosen layout becomes the first parent, and the process is repeated to produce the second parent. Using the natural selection constant allows us to sometimes (with a low probability) select the worse parent, which in turn keeps our gene pool from converging too quickly.

5.3 CROSSOVER

In this step, elements from one parent are swapped with elements of another parent, which in turn produces two children. Crossover was the most difficult part of the algorithm to implement. The classic example of crossover finds a way to represent the parent as a binary string which is then split at a random point and a piece is swapped with the other parent. For example:

parent 1	split 1	offspring 1
11111111	111 11111	111 10101
parent 2	split 2	offspring 2
01010101	010 10101	010 11111

The problem with using this method to generate offspring layouts is that the recombination of the layouts is a permutation problem and not a combination problem. Therefore, consider:

parent 1	split 1	offspring 1
RICHAD	RIC HAD	PLA HAD
parent 2	split 2	offspring 2
PLANTE	PLA NTE	RIC NTE

Using two subset layouts, when split and recombined, offspring 1 was left with two A's and offspring 2 was left with none. Therefore, trying to combine using the traditional methods proved impossible because it was possible to create offspring that had duplicate letters.

Additionally, the positions of the important keys were not preserved.

As a result, I experimented with two other methods (hereafter referred to as genetic operators) for the recombination. They were both created as generic genetic operators for permutation problems.

We first experimented with the genetic operator known as *order crossover* (OX) [37]. The order crossover picks two points to split the parents, copies the selected area to the offspring, and fills in the rest of the spaces with the unique elements in the other parent in the same order.

For example:

```
// select portion of p1
p1 = A | E C D | B F
p2 = C | F B E | A D

// copy p1 selection to offspring
off= - | E C D | - -

// delete elements in p2 that are in offspring
p2 = - | F B - | A -

// copy remaining p2 elements to offspring in order
// wrapping around if necessary
Off= A E C D F B
```

Although this genetic operator works great with other problems, it was the least effective operator for the recombination of the letters in the keyboard layout. There are six keys on the keyboard, due to the nature of the recombination the keys are shuffled around too much and the only information that is kept is the order of the keys.

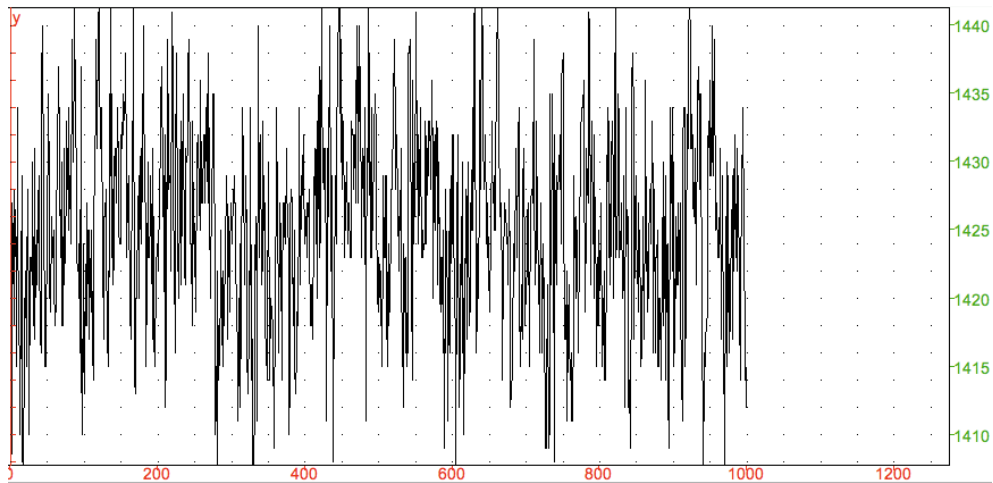


Figure 9. Best individual of each generation for 1000 generations using OX.

The second crossover technique we tried is called *permutation crossover* (PMX) [37], which recombines by creating two partition points in each parent, aligning both parents on top of each other, matching the elements, and then swapping the matched elements in each parent. For example:

```
// partition and align parents
p1 = A | B E D | C F
p2 = E | D C F | B A

// notice how:
// B matches with D
//
// swap B with D in both parents
o1 = A | D E B | C F
o2 = E | B C F | D A

// E matches with C, swap
o1 = A | D C B | E F
o2 = C | B E F | D A

// B matches with F, swap
o1 = A | D C F | E B
o2 = C | F E B | D A
```

The PMX crossover operator was on average the most effective crossover operator that we tried. In each generation, the best individual was almost always equal or better than the best individual of the previous generation.

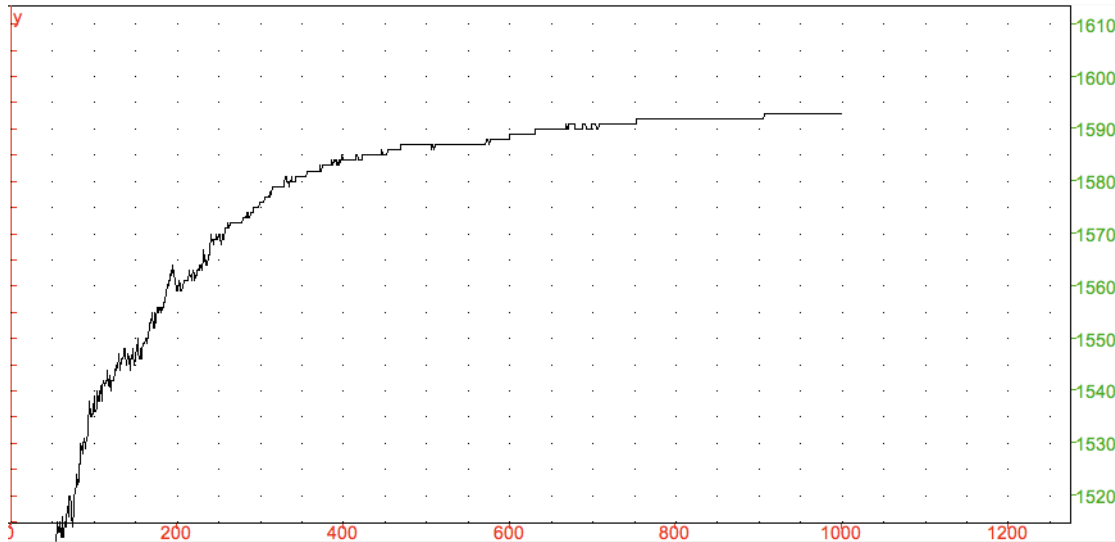


Figure 10. Best individual of each generation for 1000 generations using PMX.

The third crossover technique, called *cycle crossover* (CX) [37], recombines by finding a cycle, flagging the elements to keep them in place, and then filling in the empty spots with the elements from the other parent. For example, if we have two parents:

```
p1 = A E C D B F
p2 = C F B E A D
```

We start by looking at the letter A in parent one and then look at the letter in the same position in parent two, which is a C. The C in parent one then gets flagged to keep it in the same position, and we repeat the process until the cycle is complete. Here are the steps:

```
// Letter under A in p1 is C, so:
p2   = - F B E A D
temp = A - C - - -
```

```
// Letter under C in p1 is B, so:
p2   = - F - E A D
temp = A - C - B -
```

```
// Letter under B in p1 is A, so the cycle is complete.
// Now just fill in the missing spots with elements
// from p2:
```

```

p2   = - F - E - D
temp = A - C - B -
off1 = A F C E B D

```

```

// off2 can be created in the same way to produce:
off2 = C E B D A F

```

On average, the curve for this operator grew in a similar manner to PMX, however, the generated layouts on average scored lower than PMX. On first implementation, we were unimpressed with the results, and therefore decided to try to create our own operator for our problem. Our reasoning was that we wanted to try to guide the crossover to make smarter choices in the crossover.

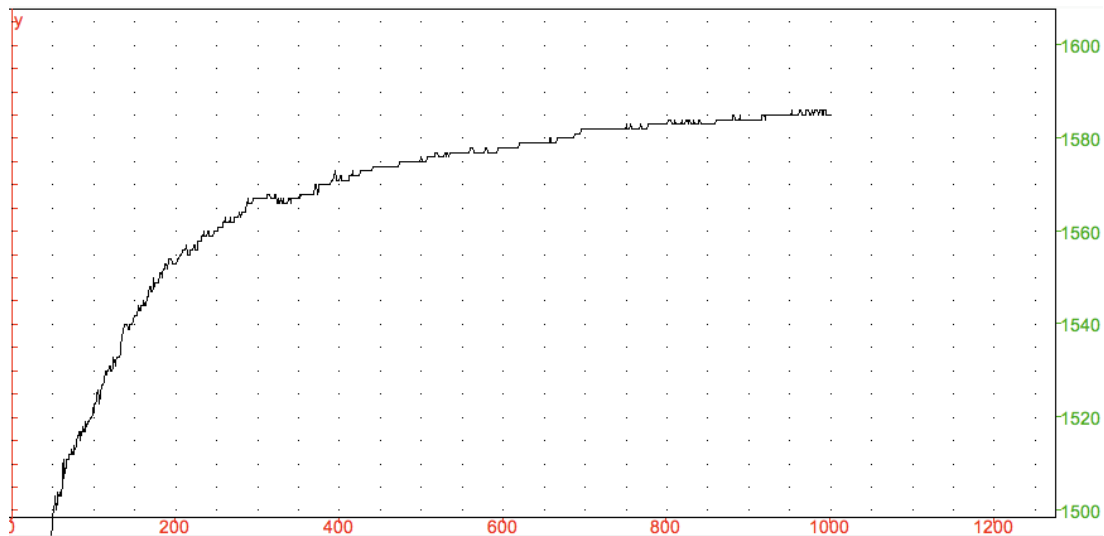


Figure 11. Best individual of each generation for 1000 generations using CX.

Our crossover function takes a list of letters from a parent and their weights generated by the fitness function. It creates an upper bound that is twenty percent higher than the highest scored letter. It then iterates through each of the letters and generates a number between zero and the upper bound. If the number is less than or equal to the score of the letter, the letter is flagged and survives in its position in the offspring. The remaining blank spots are populated from the other parent in order.

```

crossover(Keyboard parent1, Keyboard parent2) {
    Keyboard child;
    float upperBound = 0;

    // find largest weight
    foreach ( weights in parent1.weights )
        upperBound = (weight > upperBound) ? weight : upperBound;

    // increase it by 20%
    upperBound *= 1.2;

    // select which letters survive
    for ( i = 0 to parent1.size ) {
        if ( random(0,upperBound) <= parent1.weights[i] )
            child.layout[i] = parent1.layout[i];
    }

    // fill in the rest from parent2
    for ( i = 0 to parent2.size ) {
        if ( child.contains(parent2.layout[i]) ) {
            continue;
        } else {
            char letter = parent2.layout[i];
            inner: for ( j = 0 to child.size ) {
                if ( child.layout[j] == null )
                    child.layout[j] = letter;
                break inner;
            }
        }
    }
    return child;
}

```

We believed this method of recombination would be much more effective than the other methods because it preserves letters in their positions because they are in good positions. If the fitness function decides that the letters T, H, and E, are in good positions, then each of those letters should have high weights, and performing the crossover we should be more inclined to avoid separating them. And letters which are less often used, such as the letter Q should only very marginally benefit from being in any position, so it should more easily be moved around the keyboard to make space for the letters that really matter. However, in our final tests, we

discovered that this was not the case. Instead, our results were much different than expected.

Although our crossover produced much better results than order crossover, it was still inferior to CX and PMX.

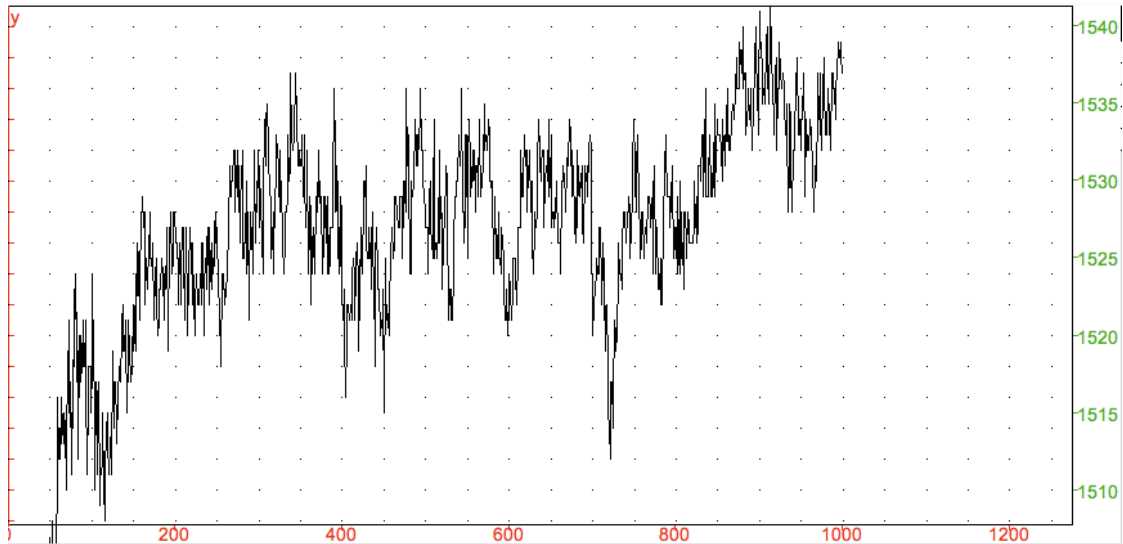


Figure 12. Best individual of each generation for 1000 generations using our own crossover.

MUTATION

Mutating the layouts was a very simple operation. Using a low probability, we swap a certain amount of letters. Sometimes there are more swaps; sometimes less depending on the pseudo-random numbers being generated.

6. FITNESS FUNCTIONS

For our problem, we defined two fitness functions. The first function scores and assigns a weight to each individual letter based on its position within the layout. The second function uses the weights generated by the first function, and “races” the layout to give the layout a total score.

6.1 INDIVIDUAL LETTER FITNESS

The individual letter fitness function uses several smaller functions to help build the weights for each of the letters. The base weight for each letter is its frequency in the English language. We wrote a simple program to analyze the frequency of all the characters in a series of documents. The relevant letters and punctuation were then isolated and normalized to produce the base weights. All the others weights produced by the other functions are then multiplied to the base to produce the final weights which are also normalized. The criteria for the other functions were as follows:

1. Position - the letters that are on keys closer to the left edge should have higher weights. People shouldn't have to reach to the center of the keyboard to access more common letters.

2. Fitt's Law Distance - the distance from one key to another as a weight based on Fitt's Law. For example, since T and H is a common letter pair, if the letter T is on one key, and the letter H is on another key, multiply both by the weight generated by Fitt's Law. We want to promote closeness between common letter pairs.

3. Letter Position Within Key - if the letter is in the center of the key, it is in the most accessible position. As a result it should receive a higher weight. And if it's in a corner, it should receive a lower weight.

4. Letter Pair in Same Corner - if there is a common letter pair on two different keys, but on the same corner, they should both be given higher weights because it's easier to lift one finger than it is two lift both.

5. Letter Pair in Same Key on Corner - same as (4), except within the same key.

All the scores are normalized and then saved so they can be used by the next fitness function or the crossover algorithm.

6.2 “RACE” LAYOUT FITNESS

Using the weights for the letters generated by the previous fitness functions, the layout is “raced” to produce a single number to rank a given layout. What we mean by “racing” is that the layout is given a passage to type. The letters are looked up and the weights added up for each of the letters in the passage. The dominant letters should have the highest weights, and therefore the better their locations, the higher their weights will be. This in turn increases the race score.

7. HUMAN TESTING AND RESULTS

For our human testing, we developed a handout as shown in Appendix A that contains fifty of the most common words in the English language, directions, and a small questionnaire. We used thirty people to test our three layouts, with ten people per layout. They were each given at least ten minutes to practice with the keyboard, and then were asked to type a small passage. We used a Two-Factor ANOVA with repetition to analyze the results. The raw results are in Appendix B. Layout 1 corresponds to ANEVS, Layout 2 corresponds to MITWS, and Layout 3 corresponds to QWERT. Based on the results, we concluded with a 90% level of confidence that we can accept the alternate hypothesis, which says that there is enough variance between ANEVS, MITWS, and QWERT. Amongst the ANEVS users, the results were on average and across the board more positive than any of the other layouts, which suggest to us that the Genetic Algorithm was successful in finding viable layouts for our keyboard.

8. FUTURE WORK

We would be interested in doing another study to test these layouts with the same body of people for a much more extended period of time, where all of the people can become familiar with all three of the layouts. We could track their progress over several months and find out which layouts they prefer, and if their perspective about mobile keyboards has changed.

9. BIBLIOGRAPHY

- [1] M. Raynal, "Claviers GAG: claviers logiciels optimisés pour la saisie de texte au stylet," *Proceedings of the 18th International Conference of the Association Francophone d'Interaction Homme-Machine*, Montreal, Canada: ACM, 2006, pp. 3-10.
- [2] S. Shanbhag, D. Rao, R. K. Joshi, "An intelligent multi-layered input scheme for phonetic scripts," *Proceedings of the 2nd International Symposium on Smart Graphics*, Hawthorne, New York: ACM, 2002, pp. 35-38.
- [3] R. W. Soukoreff, I. S. MacKenzie, "Metrics for text entry research: an evaluation of MSD and KSPC, and a new unified error metric," *Proceedings of the SIGCHI conference on Human factors in computing systems*, Ft. Lauderdale, Florida, USA: ACM, 2003, pp. 113-120.
- [4] F. E. Sandnes, "Evaluating mobile text entry strategies with finite state automata," *Proceedings of the 7th international conference on Human computer interaction with mobile devices & services*, Salzburg, Austria: ACM, 2005, pp. 115-121.
- [5] M. Hunter, S. Zhai, B. A. Smith, "Physics-based graphical keyboard design," *CHI '00 extended abstracts on Human factors in computing systems*, The Hague, The Netherlands: ACM, 2000, pp. 157-158.
- [6] E. Clarkson, K. Lyons, J. Clawson, T. Starner, "Revisiting and validating a model of two-thumb text entry," *Proceedings of the SIGCHI conference on Human factors in computing systems*, San Jose, California, USA: ACM, 2007, pp. 163-166.
- [7] N. Godard, M. Raynal, B. Martin, J. Vinot, "Etude de l'impact d'une pré-visualisation des résultats d'un système de prédiction de caractères," *Proceedings of the 21st International Conference on Association Francophone d'Interaction Homme-Machine*, Grenoble, France: ACM, 2009, pp. 235-238.

- [8] S. Zhai, B. A. Smith, "Alphabetically biased virtual keyboards are easier to use: layout does matter," *CHI '01 extended abstracts on Human factors in computing systems*, Seattle, Washington: ACM, 2001, pp. 321-322.
- [9] J. Wobbrock, B. Myers, B. Rothrock, "Few-key text entry revisited: mnemonic gestures on four keys," *Proceedings of the SIGCHI conference on Human Factors in computing systems*, Montréal, Québec, Canada: ACM, 2006, pp. 489-492.
- [10] J. Gong, B. Haggerty, P. Tarasewich, "An enhanced Multi-tap text entry method with predictive next-letter highlighting," *CHI '05 extended abstracts on Human factors in computing systems*, Portland, OR, USA: ACM, 2005, pp. 1399-1402.
- [11] M. Raynal, N. Vigouroux, "Genetic algorithm to generate optimized soft keyboard," *CHI '05 extended abstracts on Human factors in computing systems*, Portland, OR, USA: ACM, 2005, pp. 1729-1732.
- [12] M. Cicconet, P. C. Carvalho, "Playing the QWERTY keyboard," *ACM SIGGRAPH 2010 Posters*, Los Angeles, California: ACM, 2010, pp. 95.
- [13] B. Martin, P. Isokoski, F. Jayet, T. Schang, "Performance of finger-operated soft keyboard with and without offset zoom on the pressed key," *Proceedings of the 6th International Conference on Mobile Technology, Application & Systems*, Nice, France: ACM, 2009, pp. 59.
- [14] T. Felzer, R. Nordmann, "Alternative text entry using different input methods," *Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility*, Portland, Oregon, USA: ACM, 2006, pp. 10-17.
- [15] M. Goldstein, R. Book, G. Alsiö, S. Tessa, "Non-keyboard QWERTY touch typing: a portable input interface for the mobile user," *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, Pittsburgh, Pennsylvania, United States: ACM, 1999, pp.

32-39.

- [16] I. S. MacKenzie, J. C. Read, "Using paper mockups for evaluating soft keyboard layouts," *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, New York, NY, USA: ACM, 2007, pp. 98-108.
- [17] S. Mizobuchi, M. Chignell, D. Newton, "Mobile text entry: relationship between walking speed and text input task difficulty," *Proceedings of the 7th international conference on Human computer interaction with mobile devices & services*, Salzburg, Austria: ACM, 2005, pp. 122-128.
- [18] G. Badr, M. Raynal, "WordTree: nouvelle technique d'interaction avec une liste de prédiction," *Proceedings of the 21st International Conference on Association Francophone d'Interaction Homme-Machine*, Grenoble, France: ACM, 2009, pp. 227-233.
- [19] J. Gong, "Semantic & syntactic context-aware text entry methods," *Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility*, Tempe, Arizona, USA: ACM, 2007, pp. 261-262.
- [20] F. Kaplan, "Are gesture-based interfaces the future of human computer interaction?," *Proceedings of the 2009 international conference on Multimodal interfaces*, Cambridge, Massachusetts, USA: ACM, 2009, pp. 239-240.
- [21] M. Belatar, F. Poirier, "Text entry for mobile devices and users with severe motor impairments: handiglyph, a primitive shapes based onscreen keyboard," *Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility*, Halifax, Nova Scotia, Canada: ACM, 2008, pp. 209-216.
- [22] C. H. Morimoto, A. Amir, "Context switching for fast key selection in text entry applications," *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications*, Austin, Texas: ACM, 2010, pp. 271-274.

- [23] F. Vella, M. Raynal, P. Boissière, N. Vigouroux, "Techniques d'optimisation de la saisie de texte sur clavier virtuel," *Proceedings of the 17th international conference on Francophone sur l'Interaction Homme-Machine*, Toulouse, France: ACM, 2005, pp. 349-350.
- [24] D. R. Rashid, N. A. Smith, "Relative keyboard input system," *Proceedings of the 13th international conference on Intelligent user interfaces*, Gran Canaria, Spain: ACM, 2008, pp. 397-400.
- [25] N. Green, J. Kruger, C. Faldu, R. St. Amant, "A reduced QWERTY keyboard for mobile text entry," *CHI '04 extended abstracts on Human factors in computing systems*, Vienna, Austria: ACM, 2004, pp. 1429-1432.
- [26] P. O. Kristensson, L. Denby, "Text entry performance of state of the art unconstrained handwriting recognition: a longitudinal user study," *Proceedings of the 27th international conference on Human factors in computing systems*, Boston, MA, USA: ACM, 2009, pp. 567-570.
- [27] K. Go, L. Tsurumi, "Arranging touch screen software keyboard split-keys based on contact surface," *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, Atlanta, Georgia, USA: ACM, 2010, pp. 3805-3810.
- [28] T. Niikura, Y. Hirobe, A. Cassinelli, Y. Watanabe, T. Komuro, M. Ishikawa, "In-air typing interface for mobile devices with vibration feedback," *ACM SIGGRAPH 2010 Emerging Technologies*, Los Angeles, California: ACM, 2010, pp. 15.
- [29] F. Block, H. Gellersen, N. Villar, "Touch-display keyboards: transforming keyboards into interactive surfaces," *Proceedings of the 28th international conference on Human factors in computing systems*, Atlanta, Georgia, USA: ACM, 2010, pp. 1145-1154.

- [30] J. Gong, P. Tarasewich, I. S. MacKenzie, "Improved word list ordering for text entry on ambiguous keypads," *Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges*, Lund, Sweden: ACM, 2008, pp. 152-161.
- [31] J. Goodman, G. Venolia, K. Steury, C. Parker, "Language modeling for soft keyboards," *Proceedings of the 7th international conference on Intelligent user interfaces*, San Francisco, California, USA: ACM, 2002, pp. 194-195.
- [32] I. S. MacKenzie, " Fitt's Law as a Research and Design Tool in Human-Computer Interaction," *Human-Computer Interaction*, vol. 7, no. 1, pp. 91-139, 1992.
- [33] X. Bi, B. A. Smith, S. Zhai, "Quasi-qwerty soft keyboard optimization," *Proceedings of the 28th international conference on Human factors in computing systems*, Atlanta, Georgia, USA: ACM, 2010, pp.283-286.
- [34] A. Oniszczak, I. S. MacKenzie, "A comparison of two input methods for keypads on mobile devices," *Proceedings of the third Nordic conference on Human-computer interaction*, Tampere, Finland: ACM, 2004, pp.101-104.
- [35] "8pen," [Online Document], 2010, [cited 2010 Nov. 22], Available: <http://www.the8pen.com>
- [36] K. Lyons, T. Starner, D. Plaisted, J. Fusia, A. Lyons, A. Drew, E. W. Looney, "Twiddler typing: one-handed chording text entry for mobile phones," *Proceedings of the SIGCHI conference on Human factors in computing systems*, Vienna, Austria: ACM, 2004, pp. 671-678.
- [37] D. Goldberg, *Genetic algorithms in search, optimization & machine learning*, Edition of book, Massachusetts: Addison-Wesley, 1989, p. 170 - 175.

APPENDIX A

CHORDELINA HUMAN TEST SHEET

date _____

end time _____

Practice Words

the	be	to	of	and
a	in	that	have	I
it	for	not	on	with
he	as	you	do	at
this	but	his	by	from
they	we	say	her	she
or	an	will	my	one
all	would	there	their	what
so	up	out	if	about
who	get	which	go	me

Directions:

- 1) Familiarize yourself with the layout by practicing with the words above until either you've typed all the words, or 10 minutes have elapsed.
- 2) Press the back button on the phone a couple of times, and when it asks if you want to send data, press NO.
- 3) Now open a new document, and type the passage below.
- 4) Press the back button on the phone a couple of times to activate the popup that asks whether or not you want to send the data, and this time press YES.

Passage:

Every time I think of you I feel shot right through with a bolt of blue.

Strongly Disagree = 1; Disagree = 2; Neutral = 3; Agree = 4; Strongly Agree = 5

- 1 2 3 4 5 "The keyboard layout was intuitive and easy to use."
 1 2 3 4 5 "I could see myself using something like this in the future."
 1 2 3 4 5 "I like this better than current input methods."
 1 2 3 4 5 "I'm happy with the way I currently type on mobile phones."

Comments _____

APPENDIX B – Raw Data Analysis

Layout	Q1	Q2	Q3	Q4
1	2	4	5	1
1	5	4	3	4
1	4	5	3	4
1	5	4	3	4
1	3	5	2	3
1	5	5	5	3
1	4	5	3	4
1	4	5	5	2
1	4	3	3	2
1	4	4	2	3
2	3	5	2	5
2	5	5	5	3
2	4	4	4	3
2	3	2	1	4
2	1	3	2	5
2	3	2	1	5
2	4	5	3	1
2	3	2	1	3
2	4	3	2	5
2	2	3	2	3
3	4	4	3	4
3	5	4	2	4
3	3	4	5	2
3	4	4	3	4
3	5	4	4	4
3	3	4	2	2
3	4	4	4	2
3	5	4	4	3
3	4	5	3	4
3	2	3	2	5

Anova: Two-Factor With Replication

SUMMARY	Q1	Q2	Q3	Q4	Total
<i>1</i>					
Count	10	10	10	10	40
Sum	40	44	34	30	148
Average	4	4.4	3.4	3	3.7
Variance	0.888889	0.488889	1.377778	1.111111	1.189744
<i>2</i>					
Count	10	10	10	10	40
Sum	32	34	23	37	126
Average	3.2	3.4	2.3	3.7	3.15
Variance	1.288889	1.6	1.788889	1.788889	1.771795
<i>3</i>					
Count	10	10	10	10	40
Sum	39	40	32	34	145
Average	3.9	4	3.2	3.4	3.625
Variance	0.988889	0.222222	1.066667	1.155556	0.907051
<i>Total</i>					
Count	30	30	30	30	
Sum	111	118	89	101	
Average	3.7	3.933333	2.966667	3.366667	
Variance	1.113793	0.891954	1.550575	1.343678	

ANOVA

Source of Variation	SS	df	MS	F	P-value	F crit
Sample	7.116667	2	3.558333	3.101695	0.049002	2.3524
Columns	15.89167	3	5.297222	4.617433	0.00444	2.1352
Interaction	11.08333	6	1.847222	1.610169	0.151191	1.8294
Within	123.9	108	1.147222			
Total	157.9917	119				