A Comparison of Consecutive and Concurrent Input Text Entry Techniques for Mobile Phones

Daniel Wigdor, Ravin Balakrishnan
Department of Computer Science
University of Toronto
www.dgp.toronto.edu
dwigdor | ravin @dgp.toronto.edu

ABSTRACT
The numeric keypads on mobile phones generally consist of 12 keys (0-9, *, #). Ambiguity arises when the 36-character alpha-numeric English alphabet is mapped onto this smaller number of keys. In this paper, we first present a taxonomy of the various techniques for resolving this ambiguity, dividing them into techniques that use consecutive actions to first select a character grouping and then a character from within that grouping, and those that use concurrent actions to achieve the same end. We then present the design and implementation of a chording approach to text entry that uses concurrent key presses. We conducted a controlled experiment that compared this chording technique to one-handed and two-handed versions of the commonly used *MultiTap* technique. The results show that the concurrent chording technique significantly outperforms both versions of the consecutive action *MultiTap* technique.

Categories and Subject Descriptors: H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Input devices and strategies, Interaction styles;* H.1.2 [**Models and Principles**]: User/Machine Systems – Human Factors.

General Terms: Experimentation, Human Factors, Design.

Keywords: text input, mobile phones, chording.

INTRODUCTION

Entering text from the 26 character English alphabet using the standard 12-key (`0-9`,*,#) mobile phone keypad forces a mapping of more than one character per key. The typical mapping has keys `2-9` representing either three or four alphabetic characters in addition to the numerals. All text input techniques that use this standard keypad have to somehow resolve the ambiguity that arises from this multiplexed mapping. Mackenzie et al. [7] describe this problem as involving two main tasks necessary for entering a character: *between-group* selection of the appropriate group of characters, and *within-group* selection of the appropriate character within the previously chosen group.

Most text input techniques to date can generally be divided into two categories: those that require multiple presses of a single key to make the between-group followed by within-group selections, and those that require a single press of multiple keys to make these selections. Because both categories require consecutive key presses, the research focus has been on reducing the average number of key strokes per character "KSPC" required to enter text. Advances in the area generally make language specific assumptions to "guess" the desired within-group character, thus reducing or eliminating the key presses required for the within-group selection. The success of these techniques, however, is based almost entirely on how closely the text entered conforms to the underlying language model. Given that text entered on mobile phones often involves significant abbreviations and even evolving new "languages" by frequent users of SMS messaging, making language assumptions may not be the best approach to solving the text input problem.

Recently, the *TiltType* [10] and *TiltText* [13] techniques demonstrated using a second physical action – tilting the device – to make within-group selection concurrent to the between-group key press selection. This effectively shifted the research focus from reducing KSPC in consecutive key press techniques to finding new, language independent, concurrent techniques. The improvements in entry speeds demonstrated by these techniques indicate that developing new concurrent input methods may be a promising avenue for further research.

In this paper, we explore the design space of consecutive and concurrent input techniques for text entry. We first review the literature, and develop a taxonomy of current techniques. We then present the design and implemention of a concurrent chording text input technique using both hands – one to make the between-group selection, and the other to concurrently make the within-group selection. We present a controlled experiment that compares this concurrent chording technique to one and two handed versions of the most common consecutive selection technique – *MultiTap*, and discuss implications for future designs.

CURRENT MOBILE PHONE TEXT INPUT TECHNIQUES

A few mobile phones on the market today use QWERTY style keypads that allow for text entry with techniques similar to typing on a regular keyboard, albeit on a much smaller physical scale (e.g., Nokia 5510 www.nokia.com). More recently, hybrid devices that combine PDAs with phones, such as the Handspring Treo (www.handspring.com) and PocketPC Phone (www.microsoft.com), use pen-based text input techniques such as Graffiti. While these devices are making small inroads into the mobile phone market, the vast majority of mobile phones are equipped with the standard keypad (Figure 1) which has 12 keys: 0-9, *, and #.

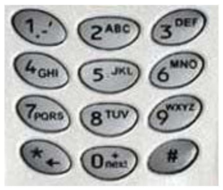

Figure 1. Standard 12-key mobile phone keypad

We now briefly review current techniques for entering text with this standard keypad, and refer the reader to Soukoreff and MacKenzie [12] for a more comprehensive review that is beyond the scope of the present paper.

MultiTap

The most common text entry technique for mobile phones is *MultiTap*, where users repeatedly press the key labelled with their desired character until it appears on the screen. For example, the characters abc traditionally appear on the 2 key. Pressing that key once yields a, twice b, and so on. In effect, multiple consecutive presses of the same key perform both between and within group selections. A problem arises when the user attempts to enter two letters from same key consecutively. For example, tapping the 7 key three times could result in either r or pq. To overcome this, *MultiTap* employs a time-out on the key presses, usually 1-2 seconds, such that no key presses during the timeout indicates completion of the current letter. Entering pq under this scheme has the user press the 7 key once for p, wait for the timeout, then press 7 twice more to enter q. To mitigate the time penalty this incurs, some versions add a “timeout kill” button that allows the user to explicitly skip the timeout. If we assume that 0 is the timeout kill button, this makes the sequence of key presses to enter pq: 7,0,7,7. *MultiTap* is simple and unambiguous, but can be slow, and has a KSPC rate of approximately 2.03 [5].

Two-key Disambiguation

In the two-key technique, users press one key to make the between-group selection, and a second key to select from within the group. For example, to enter the letter c, the 2 key is pressed to select the group abc, followed by the 3 key since c is the third letter in the group. This simple technique has failed to gain popularity for roman alphabets. It has a KSPC of 2, since all letters require two consecutive key presses.

Linguistic Disambiguation

Language based disambiguation techniques use predictive models to automate the within-group selection, but there is generally a “next” key that allows the user to choose from among the possible combinations of characters. If the user enters text that is perfectly predicted by the language model, then pressing of the “next” key is rarely required.

An example of these techniques is T9 (www.tegic.com) which computes all possible combinations of a sequence of key presses and looks them up in a dictionary. For example, the key sequence 5,3,8 results in 27 possible combinations (3x3x3 letters on each of those keys). A dictionary lookup indicates that the only valid combination in English is jet and is therefore entered as the result, with the other combinations rejected outright. Ideally, the user need only make the between-group selection, by pressing the key that is labelled with the desired character, and the system will make a perfect, automatic, within-group selection. Unfortunately, ambiguity can arise if there is more than one valid combination for the given language. Typically, the most common result is presented first. For example, the sequence 6,6 could indicate either on or no. If the algorithm suggests the wrong word, the user can manually cycle through the possible options by pressing a “next” key. An analysis of this technique for entering English text found a KSPC close to 1, indicating that the “next” key was rarely used [5]. Newer linguistic disambiguation techniques such as *LetterWise* [6] and *WordWise* (www.eatoni.com) perform similarly, with subtle advantages over earlier techniques. While these techniques all have low KSPC rates, their success relies on users entering “English like” text. As Mackenzie et al. [6] note, frequent users of text messaging system often resort to abbreviations, acronyms, or combinations of letters and numbers (e.g., b4 for before). It is also impossible to enter numerals using these techniques, so messages including numbers must be composed with a different technique. Another problem is that users have to visually monitor the display to resolve ambiguities, unlike the *MultiTap* and two-key techniques which expert users can operate “eyes free”.

TiltText

TiltText [13] requires the user to simultaneously tilt the phone in one of four directions and press a key to enter text. The key press selects the group of characters, and the tilt selects the character within that group. For example, to enter the character e, the user presses the 3 key to select the group def while simultaneously tilting the phone forward to indicate that the desired letter is the second in the group. This technique was demonstrated to be significantly faster than *MultiTap*, primarily because the two tasks (between and within group selections) are done concurrently. The KSPC is 1, but this does not reflect tilting.

A TAXONOMY OF MOBILE PHONE TEXT INPUT

In any design exercise, it is helpful to identify design dimensions and to organize existing techniques into a taxonomy that can help suggest future design possibilities. For text input techniques for mobile phones, important dimensions include input concurrency, and the number of keys and key presses needed to enter a single letter. Table 1 presents a taxonomy based on these dimensions:

	Single Key		*Multiple Keys*	
	Single Press	*Multi Press*	*Single Press*	*Multi Press*
Consecutive	#1	*MultiTap*	Linguistic, Two-key	#4
Concurrent	*TiltText* (with tilt sensor)	#2	#3	#5

Table 1. Taxonomy of mobile phone text-entry methods. The columns indicate how many keys (single or multiple) and subsequently how many presses of those keys are required to enter a single character.

From this taxonomy, it is clear that existing techniques fall within three cells. Of the unpopulated cells, #1, and #2 represent impossible situations and #4 is the regressive case since consecutive multiple presses on multiple keys will be obviously worse than the single key, multi press, *MultiTap* technique. Techniques that fit in cell #5 could be viable but likely difficult to accomplish in practice. Cell #3 suggests a technique that has not been explored for mobile-phone text entry: concurrent chording, where multiple keys are pressed once concurrently to input an unambiguous letter.

CHORDING KEYBOARDS

A chording keyboard is a one where characters are entered using combinations of key presses. Reported as early as 1942 [2] chording keyboards have been explored in various dimensions and configurations, and we now briefly review this literature.

Performance of Chording Keyboards

If characters are mapped to all possible key press combinations, a simple one-handed five key chord keyboard can enter 31 ($2^5 - 1$) distinct characters. For many text applications, this is sufficient. Adding the second hand increases this to 1023 ($2^{10} - 1$) possible unique characters.

Conrad and Longman [2] found that chording keyboards are faster and easier to learn than traditional keyboards. Gopher and Koenig [3] examined how best to determine the optimal mapping of chordings to characters of text. Gopher and Raij [4] examined whether the two-handed chording keyboard had any advantage over a one-handed implementation. They found that while both significantly outpaced a QWERTY keyboard, there was no significant difference in performance between their one and two-handed chording keyboards in the early stages of learning. As average user speed started to approach 32 wpm, the two-handed keyboard started to outperform its one-handed counterpart, and this spread in performance continued to grow as users gained more experience.

Current Chording Keyboards

Two-handed chorded keyboards have been used by the US postal service for mail sorting [11], and are still used today by stenographers. The *Twiddler* (www.handykey.com) and the *Septambic Keyer* (wearcam.org/septambic/) are examples of modern-day one-handed chording keyboards. Designed to be held in the hand while text is being entered, both are commonly used as part of a wearable computer [1], but are not used for mobile phones. The *Twiddler* is equipped with 6 keys to be used with the thumb, and 12 for the fingers, while the traditional *Septambic Keyer* has just 3 thumb and 4 finger switches. The Septambic Keyer allows for 47 different combinations of key presses, while the *Twiddler* allows over 80,000, though not all keys are used for text entry.

Another interesting chording keyboard is the the half-QWERTY developed by Matias et al. [9]. The system used half the usual number of keys of a QWERTY keypad, and required the user to press the space-bar prior to entering those keys that are normally located on one half of the keyboard. The results of their controlled experiment showed quick adaptation by expert users.

CHORDING INPUT FOR MOBILE PHONES

We have developed a new text input technique for mobile phones, called *ChordTap*, based on the principles of a chording keyboard. The mobile phone is augmented with three additional "chording" keys on the back of the phone (Figure 2). Users press a key with their dominant hand on the standard mobile phone keypad to make the between-group selection, and concurrently use their other hand to press the chording keys to make the within-group selection. This technique is similar in theory to the consecutive press *two key* method discussed previously. *ChordTap* improves upon this by adding dedicated "chord" keys for making the within-group selection. With these extra keys, users can concurrently make between and within group selections, potentially improving entry speed.

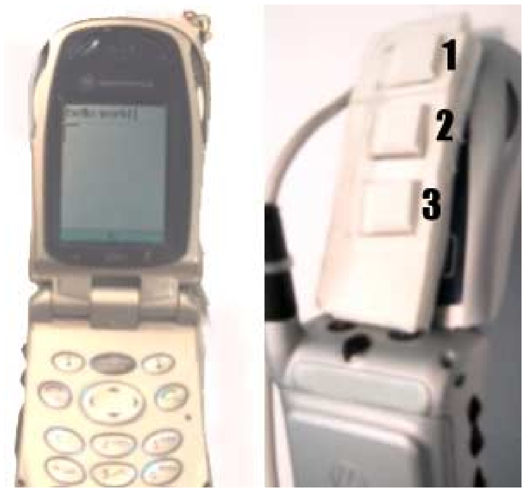

Figure 2. *ChordTap* prototype. The right image shows the chord keys mounted on the back of the phone.

There are two major design issues to consider in implementing *ChordTap*: which chord combinations indicate which letter, and which key presses to consider as "events" for text entry.

Mapping Chords States to Within-Group Selection

Each key on a mobile phone has mapped onto it one, four, or five characters: some have only the numeral, most have three letters and one numeral, while the 7 and 9 keys have four letters and one numeral (Table 2). When designing *ChordTap*, we had to decide how many chording keys to have, and how to assign combinations of chords to particular character selections. We use simple binary state switches for the chording keys. The need to map five possible within-group character selections onto the chord states dictated that we would need at least 3 chording keys to ensure unambiguous selection. The chords' states can be viewed as 3-digit binary numbers, where the i^{th} digit indicates whether that key is depressed ("1") or released ("0"). Table 2 illustrates.

Chord States	*Character selected*	*Example*
000	Numeral	7
001	First letter	p
010	Second letter	q
100	Third letter	r
011	Fourth letter	s
101	Fourth letter	s
110	Fourth letter	s
111	Fourth letter	s

Table 2. Mapping of chord state to within-group characters. Example selection shown based on pressing the 7 key.

This mapping was chosen with the intent that it be as simple as possible for the user. We believe that pressing the first chord for the first letter, second chord for the second letter, and third chord for the third letter would be a fairly intuitive mapping. The choice to use all remaining chordings for the fourth letter was made because we felt that since this mapping was used least frequently, and it was not in keeping with the more frequently used i^{th} chord to i^{th} letter mapping, it would reduce errors & learning time to simply map them all to the fourth letter. One could alternatively envision using these remaining mappings for additional characters in a non-English alphabet.

Event Handling

To enter each character, the user must input precisely two pieces of information: the between-group selection using the standard keypad, and the within-group selection using the chords. Since both the within and between group selections are explicit but separate key presses, a number of options are available when determining exactly *when* a character should be generated.

Treating Only Chord Presses as Events

In this implementation, chord presses trigger new text, but keypad presses do not. The keypad states are read only when an event is triggered by a chord press. As shown in Table 3, this approach saves work when two subsequent characters are present in the same letter group (i.e., on the same key). This savings is achieved because the user can hold down the same key while consecutively pressing the appropriate chords to generate the desired characters.

Key Held	*User Action*	*Key Held*	*Output Text*
-	depress "6"	6	
6	depress and release 3rd chord	6	o
6	depress and release 2nd chord	6	n
6	release "6"	-	
	depress "5"	5	
5	depress and release 3rd chord	5	l
5	release "5"	-	
-	depress "9"	9	
9	depress and release 3rd chord	9	y

Table 3. Sequence of actions required to enter the string "only" in a *ChordTap* implementation that treats only chord presses as events. Some consecutive actions are combined because they either generate no text, or the same text is generated with either ordering.

Of the 36^2 possible pairs of consecutive characters, there are 112 ($6 \times {}^4P_2 + 2 \times {}^5P_2$) sequences that come from the same key. This means that for 9% of all pairings the user would not need to move their finger between character entries. Though these sequences are not uniformly probable when entering text in a particular language, this still represents some savings in just about any language.

Treating Only Keypad Presses as Events

In this implementation, keypad presses trigger new text to be entered into the phone, but chord presses do not. The chords' states are read only when an event is triggered by a keypad press. As demonstrated in Table 4, this approach to text entry gives a savings of work whenever two subsequent characters appear on different keys, but share the same chord.

Chord State	*User Action*	*Chord State*	*Output Text*
000	depress 3rd chord	100	
100	depress and release "6"	100	o
100	release 3rd chord	000	
000	depress 2nd chord	010	
010	depress and release "6"	010	n
010	release 2nd chord	000	
000	depress 3rd chord	100	
100	depress and release "5"	100	l
100	depress and release "9"	100	y

Table 4. Sequence of user actions required to enter the string "only" in a *ChordTap* implementation that treats only keypad presses as events. Some consecutive actions are combined because they either generate no text, or the same text is generated with either ordering.

Of the 36^2 possible pairs of sequential characters, there are 262 (${}^{10}P_2 + 3 \times {}^8P_2 + 2 \times {}^2P_2$) sequences that share the same chording for both characters. This means that for 20% of all pairings the user would not need to change the chording between key presses, thus saving time.

Treating Both Chord & Keypad Presses as Events

In this implementation, either a chord or keypad press results in new text being entered. The advantage of this implementation is that because every state change generates a new character, expert users would benefit from the savings illustrated in both the previous event handlers. In order for this implementation to work, we must assign no character mapping to the "000" (all un-pressed) state of the chords. Table 5 demonstrates how fewer distinct actions are required to generate text in this configuration.

Chord State / Key Held	*User Action*	*Chord State / Key Held*	*Output Text*
000 / -	depress "6"	000 / 6	
000 / 6	depress and release 3rd chord	000 / 6	o
000 / 6	depress and release 2nd chord	000 / 6	n
000 / 6	release "6"	000 / -	
000 / -	depress 3rd chord	100 / -	
100 / -	depress and release "5"	100 / -	l
100 / -	depress and release "9"	100 / -	y

Table 5. Sequence of actions required to enter the string "only" in a *ChordTap* implementation that treats both chord and keypad presses as events. Note that in some cases ordering of events required to enter text is not unique.

This approach gives some savings for approximately 29% of all the possible sequences of two characters. However, this is likely harder to learn. As such, we used the "keypad presses as events" approach for our prototype, since it had the greater savings of the single-event approaches.

EVALUATION

Goals

We wished to compare the performance of *ChordTap* to existing techniques for entering text into mobile phones. For this experiment, we chose *MultiTap* as the comparison technique, because it has served as a baseline in almost every other evaluation of text entry reported to date, and because it is the most common of the consecutive action techniques. In previous experiments reported in the literature [13], *MultiTap* users were usually instructed to use only the thumb on the dominant hand to press keys. However, informal observation of *MultiTap* users indicates that many use two thumbs to enter text. Since *ChordTap* is also a two-handed technique, we tested both one and two-handed *MultiTap* use. The one-handed case served as a common baseline for comparison with previous studies.

Apparatus

Hardware

A Motorola i95cl phone was used, with chording facilitated by attaching momentary switches to the phone's back, connected via custom circuitry to the phone's serial port.

Software

The software to read chords' states and render text, as well as conduct the experiment, was written in Java 2 Micro-Edition using classes from the Mobile Devices Information Profile (MIDP 1.0) and proprietary i95cl specific classes.

The experiment was conducted entirely on the phone rather than simulating a mobile phone keypad on some other device. All software, including those implementing the text entry techniques, and data presentation and collection software, ran on the phone. No connection to an external computing device was used.

Our *MultiTap* implementation used the i95cl's built-in *MultiTap* engine, with a 2 second timeout and timeout kill. We only considered lowercase text entry in this evaluation. As such, the *MultiTap* engine was modified slightly to remove characters from the key mapping that were not on the face of the key, so that the options available were only the lower case letters and numeral on the key. This matches the traditional *MultiTap* implementation in past experiments, such as *LetterWise* [6] .

Participants

Fifteen participants recruited from the university community volunteered for the experiment. There were 5 women and 10 men of whom 2 were left-handed and 13 were right-handed. All participants had little prior experience in entering text into mobile phones, and did not receive any tangible compensation for their participation.

Procedure

Participants entered short phrases of text selected from MacKenzie and Soukoreff's corpus [8]. These phrases were selected because they have been used in previous text entry studies involving *MultiTap* [6, 13], allowing comparisons with this previous work. This corpus' high correlation of frequencies of letters to the English language is an asset, although it does not take into account abbreviations commonly used in mobile text input.

Timing began when participants entered the first character of the phrase, and ended when the phrase was entered completely and correctly. If an erroneous character was entered, the phone alerted the user by vibrating, and the user was required to correct their error. With this procedure, the end result is error-free in the sense that the correct phrase is captured. Also, the phrase completion time incorporates the time taken to correct for errors.

Phrases were shown to participants on the phone's display. Before beginning each treatment, participants were told to read and understand the displayed phrase before entering it, and were given instructions for that treatment as follows:

One-handed *MultiTap* instructions: to enter a character using the *MultiTap* technique, first find the key that is labeled with that character. Press that key repeatedly until the desired character is reached. Press once for the first character, twice for the second, three times for the third, and, if present, four times for the fourth. Once you have found the correct letter, and are ready for the next one, you simply repeat the process. If the letter you wish to enter next is on the same key, you must first either press the "right" arrow on the phone or wait two seconds for the cursor to advance. Please use only the thumb of the hand with which you hold the phone, and do not change hands during the experiment.

Two-handed *MultiTap* instructions (the same instructions were given as for the one-handed technique, with the following addendum): in this experiment, we are interested in seeing how people use MultiTap with two thumbs simultaneously. Please hold the phone with two hands so that you are able to reach all of the keys with either thumb comfortably. As you enter text, use whichever thumb you wish to press the appropriate key – do whatever feels best for you. Feel free to change how you press keys as you get more comfortable with the technique, but please be sure to press only with your thumbs.

ChordTap instructions: to enter a character using the *ChordTap* technique, first find the key that is labeled with that character, then hold it down. Next, press the chord on the back of the display that corresponds to the position of the letter on the key. For the first letter, press the top chord, for the second letter, the 2nd chord from the top, for the 3rd letter, the 3rd chord from the top. To enter the 4th letter on a key, press any two of the chords. *ChordTap* works by detecting the state of the chords at the time you release a key. Because of this, you can continue to hold down a chord if two keys in a row require the same chord. It's also not important whether you press the chords before or after the key, just so long as the correct chord is being held when you release the keys.

The experimenter then demonstrated the relevant technique. To ensure that participants understood how the technique worked, they were asked to enter a single phrase that would require the use of all chord combination for *ChordTap*, or two successive letters on the same key for *MultiTap.*

Instructions were also given to describe space and delete keys, as well as to enter an extra space at the end of the phrase to indicate completion. The process for error correction was also explained. Participants were directed to rest as required between phrases, but to continue as quickly as possible once they had started entering a phrase.

Design

Data was collected for both one and two-handed *MultiTap* and *ChordTap*. To prevent the transfer effects between techniques inherent in within-subjects designs, a between-subjects design was used. Participants were randomly assigned to three groups of five. The first group performed the experiment with the one-handed *MultiTap* technique, the second group used the two-handed *MultiTap* technique, and the third group used the *ChordTap* technique.

Participants were asked to complete two sessions of 8 blocks of trials each. Each block required the entry of 2 identical practice phrases, followed by 20 different phrases selected randomly from the corpus. Phrase selection for each of the 16 blocks were done before the experiment, and presented in the same order to each participant. Phrases were selected such that all blocks had similar average phrase lengths. The same set of phrases and blocks were used for all three techniques. In other words, all participants entered identical phrases in the same order, the only difference being which technique they used. Participants were asked to rest for at least 5 minutes between each block, and each session of 8 blocks was conducted on separate days. In summary, the design was as follows:

3 techniques x
5 participants per technique x
2 sessions per participant x
8 blocks per session x
20 phrases per block (excluding practice phrases)
= 4800 phrases entered in total.

Results

Data Summary

The data collected from 15 participants took an average of 9.9 minutes per block. A total of 109020 correct characters of input were entered for the 4800 phrases.

Physical Comfort

Some participants reported that their thumb became sore while using the one-handed *MultiTap* technique. When this was reported, the participants were encouraged to rest until they felt comfortable to proceed. No participant reported pain or discomfort in their wrist or arms.

Overall Entry Speed

The standard wpm (words-per-minute) measure was used to quantify text entry speed. Traditionally, this is calculated as (characters per second)*60/5. Because timing in our experiment started only after entering the first character, that character should not be included in entry speed calculations. Thus, the phrase length is n-1 characters in our computations. Although users entered an extra space at the end of each phrase to signify completion, the entry of the last real character of the phrase denotes the end time.

The average text entry speeds for all blocks were 13.59 wpm for *ChordTap*, 10.11 wpm for one-handed *MultiTap*, and 10.33 wpm for two-handed *MultiTap* (Figure 3). Analysis of variance showed a significant main effect for technique ($F_{2,12} = 615.8$, $p < .0001$). Pairwise means comparisons showed that *ChordTap* was significantly faster than both *MultiTap* techniques, with no significant difference between the two *MultiTap* techniques. Overall, *ChordTap* was 32% faster than two-handed *MultiTap*, which in turn was 2% faster than one-handed *MultiTap*.

Interestingly, we see in Figure 3 that while the progress of average speed per block for one-handed *MultiTap* fits the power law of learning with a high correlation (R^2 of .9032), this correlation for two-handed *MultiTap* is not as strong ($R^2 = .7964$). We attribute the difference to users' changing their use of the non-dominant hand throughout the experiment in the two-handed case. Since hand use was not prescribed, users were free to change how it was used over the course of the experiment.

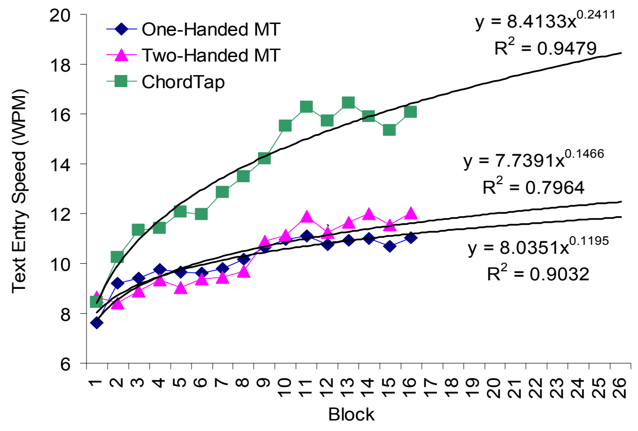

Figure 3. Entry speed (wpm) by technique and block for entire experiment. Best-fit power law of learning curve shows projected progress beyond the 16 blocks of measured data.

Learning

As Figure 3 shows, all three techniques began with roughly the same performance (average speeds of 7.62 wpm for one-handed *MultiTap*, 8.67 wpm for two-handed *MultiTap*, and 8.46 wpm for *ChordTap)*, but improved at different rates. *ChordTap* users had an overall improvement of 90% between the first and last blocks, vs. 45% and 39% for one and two-handed *MultiTap* respectively. Two-handed *MultiTap* users spent most of the first day (first 8 blocks) with lower average speeds than the one-handed users. The one-day break seemed to benefit them more, as they spent all of the second day with higher speeds than the one-handed users. By the end of the experiment, average speeds were 11.05 wpm for one-handed *MultiTap*, 12.04 wpm for two-handed MultiTap, and 16.06 wpm for *ChordTap*.

Error Rates

Recall that that our experimental procedure required participants to make corrections as they proceeded, with an end result of a completely correctly entered phrase. As such, the entry speed results discussed previously incorporate the cost of error correction. However, it is still helpful to look at a more explicit error rate. We calculate percentage error rate as the number of characters entered that did not match the expected character, divided by the length of the phrase. In this case, we used the actual length of the phrase, and not (n-1) as in the wpm rate.

Overall, there was a significant main effect for error rate ($F_{2,12}$ = 79.91, p < .0001). The error rate for one-handed *MultiTap* was 2.6%, two-handed *MultiTap* was 4.6%, and *ChordTap* was 3.3% (Figure 4).

With *ChordTap* an incorrect character can be generated in two ways: by pressing an incorrect key (key error) or incorrect chord (chord error). An examination of error rates (Figure 5) on individual letters shows that the key error rate is fairly consistent across letters (average rate of 1.8 key errors per 100 entries, standard deviation of 1.4). The chord error rate, however, varies more widely (average rate of 2.6 per 100 entries, standard deviation of 3.3). Pairwise means comparisons revealed that the chord error rate was significantly higher (*p* < .0001) for characters that required multiple-chord chording (s,z). We attribute this higher rate to the less obvious chording scheme (others are first chord=first letter, second chord = second letter, etc), and to the requirement to press two chords simultaneously. While the higher error rate for 'z' could also be attributed to its lower frequency and thus fewer opportunities for user practice, the same cannot be said for 's' which appears as frequently as characters (a,i,n,r) with lower error rates.

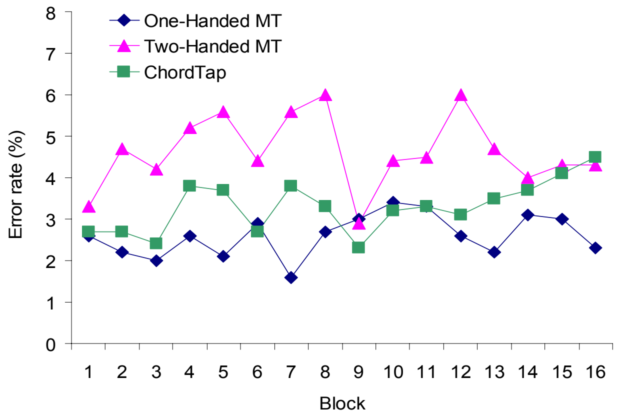

Figure 4. Error rate per 100 attempted character entries by block for all three techniques

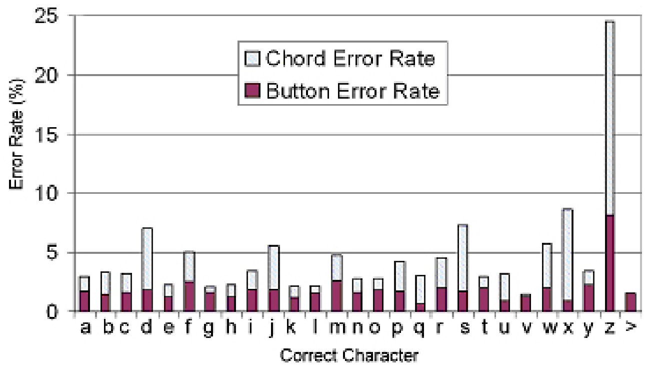

Figure 5. Key and Chord error rates per 100 attempted entries for each character in the experiment (space shown as ">")

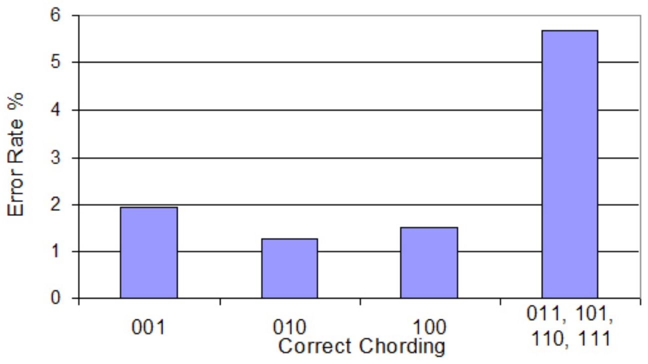

Figure 6. Chord error rate by required chord. Since all multiple-chords (011,101,110,111) produced the same letter in our prototype, they are combined in this graph.

DISCUSSION & FUTURE WORK

This is a proof of concept experiment that indicates concurrent chording to be a viable text input technique for mobile phones. Note that these results were achieved despite a fairly crude prototype of switches for entering chords. As such, it is highly likely that with better industrial design of the chord switches and their integration with the phone, even greater performance benefits could be realized. It is also plausible that an appropriately designed layout could enable chording and keypad entry to be performed using the fingers of one hand. It will be the topic of future work to examine how this technique could be adapted for the use with one hand, and how varying the placement of the chords impacts speed of entry. We will also examine applying the research of Gopher & Koenig [3] to alter the mappings of chords to characters to optimize entry speed.

One of the reasons why we chose to compare *ChordTap* to *MultiTap* was because *MultiTap* is used as a baseline technique in most studies of text entry performance. We are able to make direct comparisons with our own previous technique, *TiltText*, since our experiment in [13] used a nearly identical design. We are able to overcome the only difference between the present and previous work by looking only at the data from the between-subject portion of the earlier experiment [13]. At the end of the experiment for *TiltText*, users had achieved speeds of 13.5 wpm, with an error rate of 8.6%, as compared with *ChordTap*'s 16.06 wpm and 4.5% in the present study.

As was done in [13], we are also able to approximate a comparison of the performance of *ChordTap* to MacKenzie's *Letterwise*, by comparing our results to that of [6]. We see that the 16th block of our experiment is roughly equivalent to the 6th or 7th session in MacKenzie et al.'s experiment. At this point, their data for *MultiTap* is roughly in the 11 wpm range, which is very close to ours. At the same point in time, entry rates for *LetterWise* are about 14 wpm, which is in the same range as our experiment's rate for *ChordTap* of 16.06 wpm. While the different experimental designs, number of phrases per block, and other factors necessarily imply that these cross-experiment comparisons are not precise, this rough analysis does give us a ballpark sense of how *ChordTap* compares to techniques other than *MultiTap.*

CONCLUSION

We have described a taxonomy of mobile phone text-entry research, with particular focus on the differences between consecutive and concurrent approaches. Our experiment has demonstrated the effectiveness of the classic concurrent technique, chording, when applied to mobile phone text entry. The performance advantages seen over the consecutive action *MultiTap* technique, and consequently the inherent advantages over linguistic disambiguation techniques, indicates that concurrent text input could be a viable alternative to current techniques.

ACKNOWLEDGMENTS

We thank Tovi Grossman, Maya Przybylski, our experiment participants, members of the Dynamic Graphics Project lab, and Microsoft Research.

REFERENCES

1. Barfield, W., & Caudell, T., eds. (2001). Fundamentals of wearable computers and augmented reality. Lawrence Erlbaum Associates: Mahwah, New Jersey.
2. Conrad, R., & Longman, D. (1965). Standard typewriter versus chord keyboard: An experimental comparison. *Ergonomics*. 8. p. 77-88.
3. Gopher, D., & Koenig, W. (1983). Hands coordination in data entry with a two-hand chord typewriter.Technical Report CPL 83-3. Cognitive Psychology Laboratory, Dept. of Psychology, University of Illinois, Champaign, ILL 61820.
4. Gopher, D., & Raij, D. (1988). Typiing with a two-handed chord keyboard: Will QWERTY become obsolete. *IEEE Transactions on Systems, Man, and Cybernetics*. 18(4). p. 601-609.
5. MacKenzie, I.S. (2002). KSPC (keystrokes per character) as a characteristic of text entry techniques. *Fourth International Symposium on Human-Computer Interaction with Mobile Devices*. p. 195-210.
6. MacKenzie, I.S., Kober, H., Smith, D., Jones, T., & Skepner, E. (2001). LetterWise: Prefix-based disambiguation for mobile text input. *ACM UIST Symposium*. p. 111-120.
7. MacKenzie, S., & Soukoreff, W. (2002). Text entry for mobile computing: Models and methods, theory and practice. *Human-Computer Interaction*. 17. p. 147-198.
8. MacKenzie, S., & Soukoreff, W. (2003). Phrase sets for evaluating text entry techniques. *Extended Abstracts of the ACM CHI Conference on Human Factors in Computing Systems*. p. 754-755.
9. Matias, E., MacKenzie, I., & Buxton, W. (1996). One-handed touch typing on a QWERTY keyboard. *Human-Computer Interaction*. 11. p. 1-27.
10. Partridge, K., Chatterjee, S., Sazawal, V., Borriello, G., & Want, R. (2002). TiltType: accelerometer-supported text entry for very small devices. *ACM UIST Symposium*. p. 201-204.
11. Rosenberg, B. (1994). Chord keyboards. Queen Mary & Westfield College. London, UK.
12. Soukoreff, W., & MacKenzie, I.S. (2002). Text entry for mobile computing: Models and methods, theory and practice. *Human-Computer Interaction*. 17. p. 147-198.
13. Wigdor, D., & Balakrishnan, R. (2003). TiltText: Using tilt for text input to mobile phones. *ACM UIST Symposium*. p. 81-90.