LetterWise: Prefix-based Disambiguation for Mobile Text Input

I. Scott MacKenzie[1], Hedy Kober[2], Derek Smith[3], Terry Jones[3], and Eugene Skepner[3]

[1]Dept of Computer Science
York University
Toronto, ON, Canada
+1 416 736-2100
mack@yorku.ca

[2]Dept. of Psychology
Columbia University
New York, NY
+1 212-854-3608
hk384@columbia.edu

[3]Eatoni Ergonomics, Inc.
171 Madison Ave.
New York, NY
+1 212 725 9766
{dsmith,terry,eu}@eatoni.com

ABSTRACT

A new technique to enter text using a mobile phone keypad is described. For text input, the traditional touch-tone phone keypad is ambiguous because each key encodes three or four letters. Instead of using a stored dictionary to guess the intended word, our technique uses probabilities of letter sequences — "prefixes" — to guess the intended letter. Compared to dictionary-based methods, this technique, called *LetterWise*, takes significantly less memory and allows entry of non-dictionary words without switching to a special input mode. We conducted a longitudinal study to compare *LetterWise* to *Multitap*, the conventional text entry method for mobile phones. The experiment included 20 participants (10 *LetterWise*, 10 *Multitap*), and each entered phrases of text for 20 sessions of about 30 minutes each. Error rates were similar between the techniques; however, by the end of the experiment the mean entry speed was 36% faster with *LetterWise* than with *Multitap*.

Keywords

Text entry, mobile phones, language modeling

INTRODUCTION

In December 2000, fifteen billion text messages were sent using the standard 12-key mobile phone keypad. This number is provided by the GSM World Association (www.gsmworld.com) who also note that volumes are doubling every six months. This is particularly remarkable in view of the poor affordances of the mobile phone keypad. Fifteen billion messages translates into about one trillion keystrokes, assuming six words per message and input via the conventional multitap technique used on mobile phones. In this paper we present a new technique for entering text using a mobile phone keypad that approximately halves the number of keystrokes required.

Mobile Phone Keypad

Text entry on a mobile phone is based on the standard 12-key telephone keypad (see Figure 1).

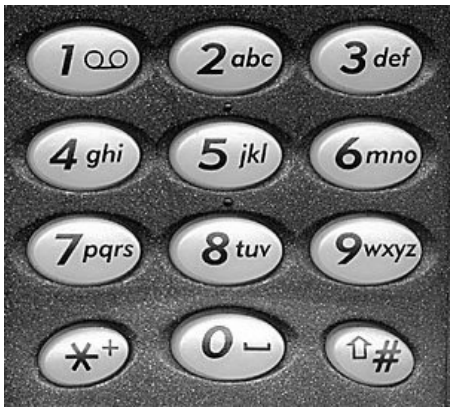

Figure 1. The standard 12-key telephone keypad

The 12-key keypad consists of number keys `0-9` and two additional keys (`*` and `#`). The letters `a-z` are spread over keys `2-9` in alphabetic order. The `SPACE` character is often assigned to the `0` key, but this varies depending on the phone. As alphabet size is typically at least 26 letters, three or four letters are grouped on each key, and, so, ambiguity arises.

In the following sections we describe three methods for working with this ambiguity. The first, *Multitap*, is the established method for entering names into a mobile phone's address book. As a general purpose text input method, however, it is slow, inefficient, and not well liked by users [4]. The second is a dictionary-based method with several commercial implementations. The third is a new method we call *LetterWise*. *LetterWise* is a linguistically optimized technique that is not dictionary-based.

Multitap

With *Multitap*, the user presses each key one or more times to specify the desired letter. For example, the `2` key

is pressed once for the letter a, twice for b, three times for c. Besides requiring multiple keystrokes for many letters, *Multitap* requires a mechanism to segment consecutive letters on the same key. An example is the word on, because both o and n are on the 6 key. To enter on the user presses 6 three times, waits for the system to timeout, and then presses 6 twice more to enter n. Another approach is to press a special key to skip the timeout ("timeout kill"), thus allowing direct entry of the next character on the same key. Some phones use a combination of the two solutions. For example, Nokia phones implement a 1.5 second timeout and a timeout-kill using the DOWN-ARROW key. The user decides which strategy to use.

Dictionary-based Disambiguation

Another way to overcome ambiguity is to add a dictionary to the system. One such technique is known as *dictionary-based disambiguation*. Commercial examples include *T9* by Tegic Communications (www.tegic.com), *eZiText* from Zi Corp. (www.zicorp.com), or *iTAP* from the Lexicus division of Motorola (www.motorola.com/lexicus).

With dictionary-based disambiguation, each key is pressed only once. For example, to enter the, the user enters 8-4-3-0. The 0 key, for SPACE, delimits words and terminates disambiguation of the preceding keys. The key sequence 8-4-3 has $3 \times 3 \times 3 = 27$ possible renderings (see Figure 1). The system compares the possibilities to a dictionary of words to guess the intended word.

Naturally, disambiguation is not perfect since multiple words may have the same key sequence. In these cases the most-probable word is the default. However, if the desired word is not the most-probable, overhead is incurred. For example, there are four words matching the key sequence 2-2-5-3. From most-to-least probable, the words and the required key sequences are

```
able 2-2-5-3-0
cake 2-2-5-3-N-0
bald 2-2-5-3-N-N-0
calf 2-2-5-3-N-N-N-0
```

If the user intends calf, then three presses of a special NEXT key are required to reach the correct response. Clearly, "one key per letter" is an over simplification of user interaction with dictionary-based entry methods.

Prefix-based Disambiguation

LetterWise was developed to avoid the problems just noted. It works with a stored database of probabilities of prefixes. A prefix is the letters preceding the current keystroke. For example, if the user presses 3 with prefix th, the most likely next letter is e because the in English is far more probable than either thd or thf.

The most significant departure is that *LetterWise* does not use a dictionary of stored words. Instead, a priori analysis of a dictionary is used to distill probability information about letter sequences in the language. This allows efficient entry of words and, unlike dictionary-based approaches, generalizes to non-words.

LetterWise occasionally guesses the wrong letter, and in these cases the user must press a special NEXT key to choose the next mostly likely letter for the given key and context. This behaviour is examined in detail shortly.

The performance of *LetterWise* improves with the number of preceding letters considered. In *LetterWise*, improved performance means fewer presses of the NEXT key. Increasing the number of preceding characters considered also increases the memory footprint of the implementation — an important consideration for mobile devices. Prefixes of length 3 were used in the experiment described in this paper.

Letterwise databases store information on a selected subset of prefixes. In practice, the memory requirements vary from about 500 bytes to 9000 bytes. See [5] for details.

Keystrokes Per Character (*KSPC*)

Keystrokes per character (*KSPC*) is a useful metric for characterising overall text entry behaviour. *KSPC* is the number of keystrokes, on average, required to produce each character using a given input method.

As a baseline, consider $KSPC = 1$. This is a reasonable measure for a Qwerty keyboard, because each letter has a dedicated key. $KSPC < 1$ is possible, for example, with word prediction techniques. $KSPC > 1$ is likely if the keyboard has fewer keys than symbols in the target language.

An extreme example of $KSPC > 1$ is text input using a 5-button two-way pager. With these devices, the cursor is maneuvered over letters using four arrow buttons and then a letter is selected using the ENTER button. If letters are presented alphabetically in two rows, the effect is $KSPC = 6.18$. Bellman and MacKenzie [1] describe a technique to reduce this to $KSPC = 4.03$ by fluctuating the layout after each keystroke to minimize the cursor distance to the next letter.

Multitap, *T9*, and *LetterWise* all have $KSPC > 1$. It is possible to compute the *KSPC* characteristic for a given entry technique using a language corpus (see [8] for details). For our investigations, we used the British National Corpus (ftp://ftp.itri.bton.ac.uk/bnc/). For simplicity, we reduced the 90 million word corpus to a list of approximately 65 thousand unique words and their frequencies.

Table 1 compares the *KSPC* characteristic for *Multitap*, *LetterWise*, and dictionary-based disambiguation techniques such *T9*. The measures were computed

considering only the letters `a-z` and the SPACE character. Punctuation and other symbols are excluded. Although important, such symbols, by and large, do not represent a "point of differentiation" among the entry techniques considered.

Table 1
Keystrokes Per Character
(*KSPC*) for Various Techniques

Technique	*KSPC*
Multitap	2.0342
Dictionary-based disambiguation (*T9*)	1.0072[a]
LetterWise	1.1500

[a] see text for important assumptions

At *KSPC* = 1.1500, *LetterWise* requires 43.5% fewer keystrokes per character than *Multitap*.

The figure for dictionary-based disambiguation is quite impressive at first glance. That it is so close to 1.0000 suggests that presses of NEXT are relatively rare with dictionary-based disambiguating methods. As noted by Silfverberg et al. [11], only about 5% of words require the NEXT function. The keystroke overhead reflected in the *KSPC* figure (1.0072) is much less than 5% however, since it is weighted by word frequency. Silfverberg et al.'s figure is unweighted: it is an absolute measure of the ratio of words requiring at least one press of NEXT. Importantly, their measure excludes the most probable word in any ambiguous set (e.g., "able", mentioned earlier) because it is the default and is entered directly.

The apparently impressive *KSPC* figure with *T9* is predicated on the rather generous assumption that users only enter dictionary words. It is well known that text messaging users employ a rich dialect of abbreviations, slang, etc. [3] When confronted with non-dictionary words, or when the user makes spelling or typing errors, dictionary-based disambiguation fails completely, and the user's only recourse is to switch to an alternate entry mode, such as *Multitap*. *LetterWise* bears no such assumptions, because it is not dictionary-based. We will describe the behaviour of *LetterWise* and *T9* on non-dictionary words later.

Presses of NEXT

In our implementation *LetterWise*, prefixes do not cross word boundaries. Thus, when entering the first letter of a word, the prefix is empty. For the second letter, the prefix has size one, and so on to the maximum prefix length. For this reason, keystroke overhead occurs primarily at the beginning of words. The probability of a letter appearing correctly increases sharply with position within a word.

Normalizing for word frequency, 50.1% of all words can be entered without ever pressing the NEXT key. Of the remaining 49.9%, most presses of NEXT occur on the first letter in a word, and, of these, usually just one press of NEXT is needed. Once the user successfully enters the first letter in a word, the need for presses of NEXT is greatly reduced (see Figure 2).

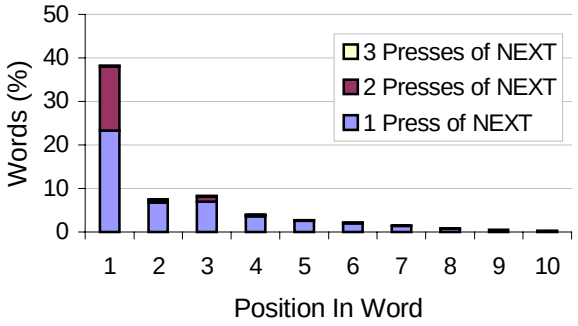

Figure 2. Press of NEXT vs. letter position in word

Non-Dictionary Words

Since *LetterWise* is prefix-based, not dictionary-based, it does not fail catastrophically when the user attempts to enter a non-dictionary word, such as a proper noun, abbreviation, or slang. The user can always succeed and, the more the word resembles English, the fewer presses of NEXT required. For example, the German word "haltestelle" is entered in LetterWise as follows:

```
h altes telle
4N25837N83553
```

Even if the word does not resemble English, users can always succeed in entering it. For example `hokkaido` is entered in *LetterWise* as follows:

```
h ok k aid o
4N65N5N243N6
```

There is no need to switch to *Multitap* mode to enter non-dictionary words, as is the case with *T9* and other dictionary-based methods.

Predicting Asymptotic Text Entry Rates

Soukoreff and MacKenzie [12] developed a model that combines Fitts' law and digram probabilities in a language to predict asymptotic text entry rates for tapping on a soft keyboard with a stylus. Silfverberg et al. [11] extended the model to finger input on a mobile phone keypad using various techniques. Table 2 reproduces Silverberg et al.'s figures and adds an additional entry for *LetterWise*.

Table 2
Predicted Asymptotic Text Entry Rates (wpm)

Method	Index finger	Thumb
Multitap		
- wait for timeout	22.5	20.8
- timeout kill	27.2	24.5
T9	45.7 [a]	40.6 [a]
LetterWise	38.1	33.7

[a] see text for important assumptions

LetterWise's position is not surprising, given the *KSPC* values in Table 1. At 33-38 wpm, the predicted entry rates for *LetterWise* rates are lower than those for *T9*, however, they do not carry similar assumptions with respect to ambiguous words or non-dictionary words.

Phrase Set

One of the first steps in designing an empirical evaluation is constructing a set of phrases to be entered. Our phrase set was created manually. We began with MacKenzie and Zhang's [9] set of 70 phrases and expanded it to 500 phrases. The goal was to construct phrases that were of moderate length, easy to remember, and with letter frequencies typical of English. The phrases included only letters and spaces. Figure 3 gives the main characteristics of the phrase set. The letter frequencies were tested against a standard reference [10]. The high correlation (r = .9541) indicates the phrase set was representative of English.

Number of phrases	500
Average phrase length (min / max)	28.6 (16 / 43)
Number of words (unique words)	2711 (1163)
Average word length (min / max)	4.45 (1 / 13)
Letter correlation with English	r = .9541

Figure 3. Characteristics of phrase set

With this background, we now present our empirical evaluation of *LetterWise*. We used *Multitap* as the point of comparison.

METHOD

Participants

Twenty participants volunteered for the experiment. They were recruited based on contacts within two university communities. Participants were paid an hourly rate, plus a bonus upon completion.

We used a between-subjects design and randomly assigned participants to either the *LetterWise* or *Multitap* condition, ten subjects per condition. A within-subjects design was considered, but not employed because of the potential for interference between the cognitive and motor skills needed for each technique.

Apparatus

Hardware

The experiment was conducted on computer systems running Mandrake's *GNU/Linux* version 7.2. Output was viewed on a 19" colour monitor. Text entry was performed using a PC Concepts *KB-5640* numeric keypad with standard 19 mm keys re-labeled to match the letter and number assignments typical of mobile phone keypads (see Figure 4). Participants pressed keys using a technique of their choosing, typically using the index finger of the right (preferred) hand. The keypad was either held in their left hand or positioned on the desk, as desired by each participant.

Figure 4. Keypad used in the experiment

Software

Our experimental software and analysis routines were developed in C, C++, Python, Perl, and Java. Figure 5 shows the interface.

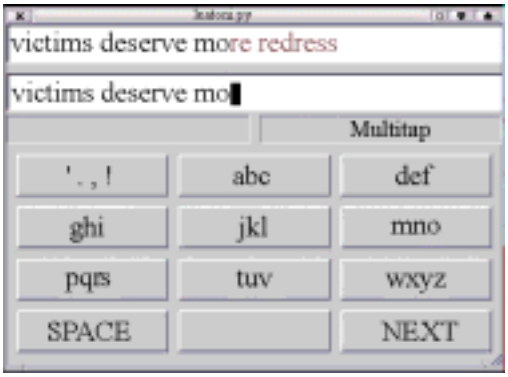

Figure 5. Screen shot of the experimental software

Procedure

Participants entered short phrases of text presented to them on the display. The instructions were brief, with the intent to simulate one or two screens of text on a mobile phone.

LetterWise instructions: When typing, press the key with the letter you want. Most probably, the letter you intend will appear. If it does not, press the NEXT key repeatedly until the right letter appears.

Multitap instructions: When typing, press the key with the letter you want. Press the key repeatedly until the letter appears. (Example: on the 2 key, press once for `a`, twice for `b`, three times for `c`.) If the same key is needed for two consecutive letters, such as `ba` in `bat`, then enter `b`, press the NEXT key, and then enter `a`.

Some additional instructions were given on the operation of the software, the treatment of errors, and the need to press the NEXT key at the end of a phrase to bring up the next phrase.

A beep was sounded if the software detected a keystroke error. In this case, participants had to adjust subsequent keystrokes to correct the error and regain synchronization with the presented text. With this procedure, the final product was error-free. Therefore, our error analyses are of keystroke errors, rather than character errors.

Participants were also told to rest at their discretion between phrases, but to proceed expeditiously through a phrase once the first character was entered.

Design

Participants performed twenty sessions of about 25-30 minutes each. Participants signed up for 1-hour appointments, and thus completed two sessions per appointment, with about a 5-minute rest in between. Appointments were booked on consecutive days (with occasional gaps of two days for weekends), with as many as two appointments per day, provided appointments were separated by at least one hour. This was done to ensure adequate rest.

The experiment was a 2×20 factorial design. "Entry method" was a between-subjects factor (*LetterWise* vs. *Multitap*), and "Session" was a within-subjects factor (1, 2, 3 … 20).

RESULTS

Data Summary

The files collected for 20 participants tested over 20 sessions of 25-30 minutes contained about 16 MB of raw data. These contained keystroke-level data for 23,709 phrases, totaling 1,076,676 keystrokes of input.

Entry Speed

The means for session one were 7.3 wpm and 7.2 wpm for *LetterWise* and *Multitap*, respectively. Improvement with practice was readily seen with both methods. On the 20th session entry speeds were 21.0 wpm and 15.5 wpm for *LetterWise* and *Multitap*, respectively. Thus, although *LetterWise* was only marginally faster initially

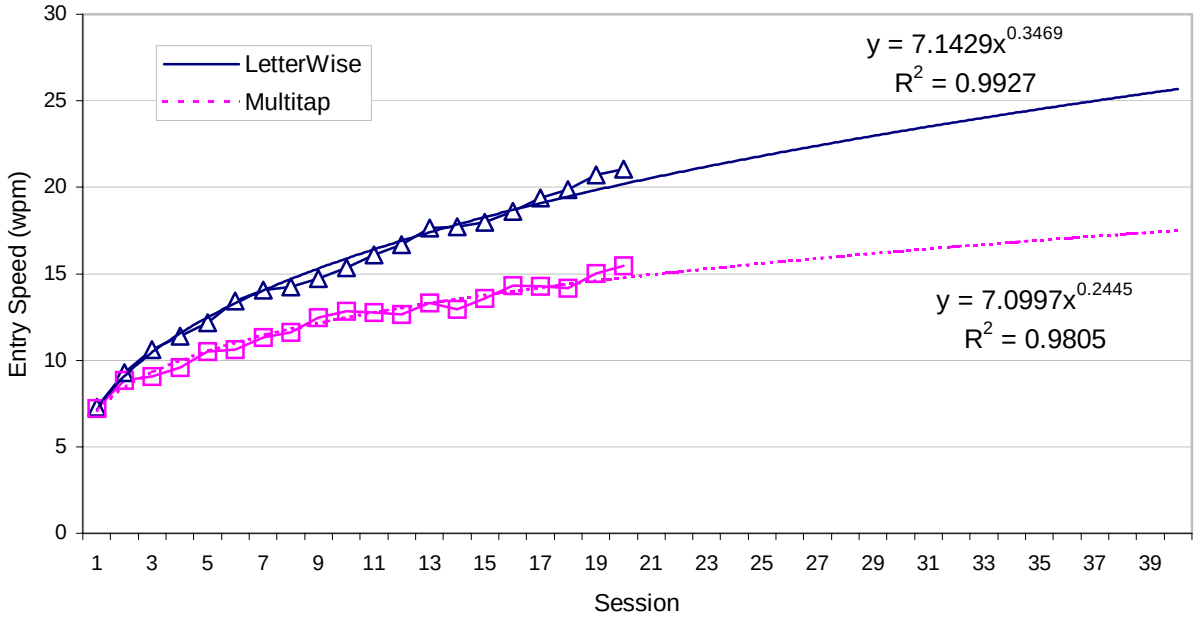

Figure 6. Entry speed (wpm) by entry method and session

(1.1%), the spread increased to 36.3% by the end of the experiment (see Figure 6).

An analysis of variance indicated significant main effects for entry method ($F_{1,18} = 4.33$, $p = .05$) and session ($F_{19,342} = 58.52$, $p < .0001$), and a significant entry method by session interaction ($F_{19,342} = 8.74$, $p = .0005$). These effects are seen in Figure 6.

The improvement with practice is further illustrated in the trend lines and prediction equations in Figure 6. These were computed through a least squares fit using the conventional power law of learning (see [1, 9] for examples). The following models resulted:

LetterWise: $y = 7.1429\ x^{-0.3469}$, $R^2 = .9927$

Multitap: $y = 7.0997\ x^{-0.2445}$, $R^2 = .9805$

where y is the predicted entry speed in "words per minute" and x is the number of 30-minute sessions. An extrapolation to the 40th session is shown in the figure. The high R^2 values imply that the fitted models provide a very good prediction of user behaviour. In both cases over 98% of the variance is accounted for in the models. For both entry methods the observed and predicted entry speeds are well below the speeds predicted in Table 2, suggesting there is plenty of room for improvement with practice.

Error Rates

The grand mean for error rate was 5.2% (see Figure 7). Overall, the error rates were slightly higher for *LetterWise* than for *Multitap*; however an ANOVA revealed that the differences by entry method were not statistically significant ($F_{1,18} = 0.384$, ns). There was also no statistical significance for the session main effect ($F_{19,342} = 1,692$, $p > .05$) or for the entry method by session interaction ($F_{16,342} = .653$, ns).

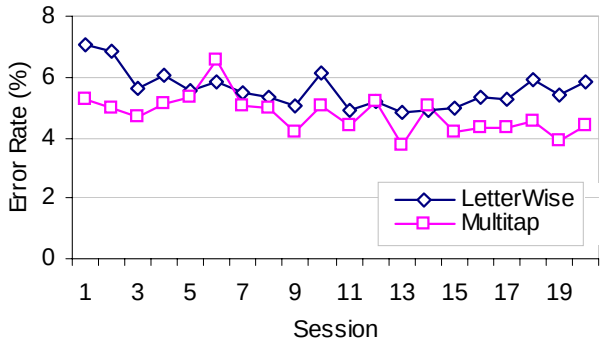

Figure 7. Errors rates (%) by entry method and session

DISCUSSION

Skill Acquisition

Our analyses and observations suggest that learning is divided into three phases:

Discovery phase. In this phase, speed of entry is dominated by users' familiarity with convention, such as alphabetic ordering. It appears that this phase lasts only a few hundred keystrokes.

Motor reflex acquisition phase. This phase begins after the discovery phase and lasts for thousands of keystrokes. During this phase, speed of input increases logarithmically. Participants in the present experiment performed about 50,000 keystrokes each over the 20 sessions of data entry. Learning was continuing, even at the end of the experiment (see Figure 6). On a log-log plot (not shown), it is more clearly seen that learning continues in the usual power-law fashion [1, 9].

Terminal (Fitts' law) phase. For advanced experts, all reflexes are learned, and entry speed is determined by keypad geometry and the frequency with which pairs of keys are operated in succession. Fitts' law pertains to this advanced stage of learning [11, 13]. At this stage, all functions of all keys are known perfectly well, and entry time is purely a function of motor constraints in the interface. Although such behaviour is unlikely to ever take hold fully, the approximations afforded from Fitts' law analyses represent a useful point in the interaction space — an asymptote toward which experts progress.

In comparison with the Fitts' law predictions in Table 1 for *Multitap* (index finger, timeout kill) and *LetterWise*, our participants are well short of reaching their expected asymptotic rates. For *Multitap*, the session 20 mean of 15.5 wpm is 56.8% of the expert prediction of 27.2 wpm. Simarily for *LetterWise*, the session 20 mean of 21.0 wpm is 55.1% of the predicted asymptotic rate of 38.1 wpm.

Components of Character Entry Time

In this section we present more-detailed analyses of users' interaction with *Multitap* and *LetterWise*.

To operate *Multitap* successfully, the user must discover the following processes for entering letters:

Find - locate the key for the desired letter, and press it.

Adjust - if the desired letter does not appear, press the key again until it does.

Timeout kill - if the same key is required for consecutive letters, press a timeout-kill button between the letters.

To operate *LetterWise*, the process is a bit simpler:

Find - locate the key for the desired letter, and press it.

Adjust - If the desired letter does not appear, press the NEXT key until it does.

Separate analyses of these components are presented below.

Finding a Key

The time to find a key, t_F, was defined operationally as the time from the last keystroke of one character to the first correct keystroke of the next character. One simple hypothesis for this component of character entry time is that novices find letters by visually scanning the keys sequentially. This should diminish with practice, with just motor constraints remaining.

There is no reason to suspect any difference in t_F between *Multitap* and *LetterWise*. In fact this was the case. t_F was essentially the same for both techniques, starting at about 1500 ms for session 1 and improving to about 550 ms on session 20.

Adjust Time and Timeout Kill

The time to adjust, t_A, was defined operationally as the time from the first correct keystroke for a character until the character was actually obtained through presses of the same key (*Multitap*), or presses of the NEXT key (*LetterWise*). In many cases t_A was zero as no adjustment was necessary.

Timeout kill time, t_K, is simply the time from the keystroke that produced the correct character to correctly pressing the timeout kill key, if needed. t_K is only required in *Multitap* mode.

The separate effects of t_A and t_K are shown in Figure 8 for *LetterWise* (bottom line) and *Multitap* (top two lines). It is seen that t_A for *LetterWise* (bottom line) decreases with practice, starting initially at about 250 ms and dropping steadily to 100 ms by session 20. Improvement continues to the end of the experiment. The values are small throughout, and this is because the figure is an average over all characters entered. With *LetterWise* 86% of characters are entered without an adjustment; i.e., $t_A = 0$ 86% of the time.

For *Multitap* (top two lines), the situation is different. Time to adjust, t_A, starts off at about 340 ms, improves to 260 ms by session 5, and then remains the same thereafter. Thus, participants quickly learn the requisite behaviour for *"multi-tapping"*, but the motor component remains and is fixed following this initial period of learning — about 2.5 hours in our experiment. The multi-tapping behaviour is required at least once for about 56% of all characters. Figure 8 suggests that multi-tapping adds on average 260 ms to character entry time.

Timeout kill time, t_K, with *Multitap* is shown as an additive component of character entry time in Figure 8. It adds about 150 ms to the character entry time initially, but drops to about 60 ms by session 7. Only slight improvement appears thereafter. Again, the value is small because it is an average over all characters entered. Timeout kill is only needed about 8% of the time with *Multitap*.

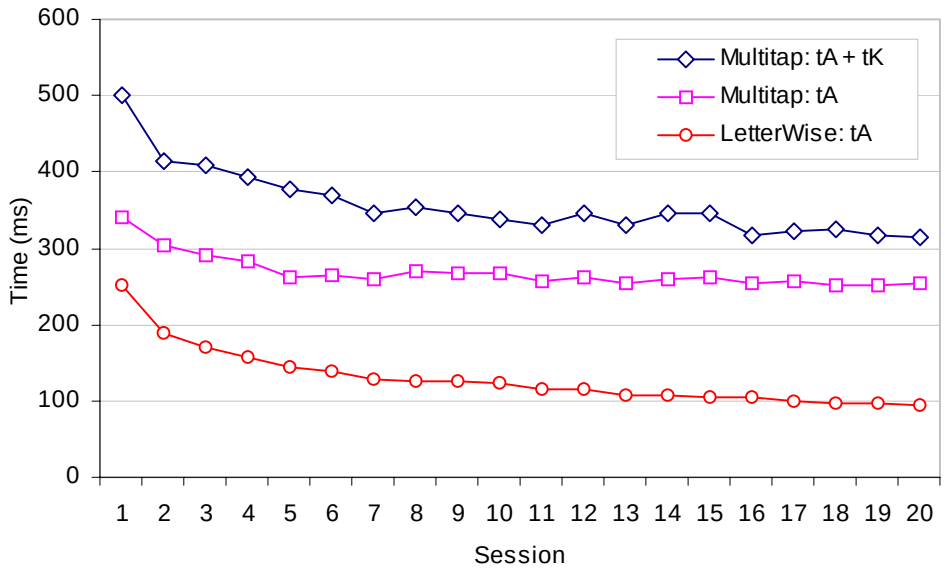

Figure 8. Time to adjust (t_A) and timeout kill (t_K) processes as a function of practice for *Multitap* (top two lines) and *LetterWise* (bottom line) (Note: Timeout kill is only required for *Multitap*)

A timeout kill in *Multitap* is in some sense similar to a press of the NEXT key in *LetterWise*. Both are required infrequently, and, in these experiments, both are accomplished using the same key — the # key on the standard telephone keypad.

Components of Character Entry Time for T9

To include dictionary-based entry methods in our analyses, we need to identify the components of character entry time, as just done for *Multitap* and *LetterWise*. Importantly, the behaviour for ambiguous words and non-dictionary words must be included. Our observations with several *T9*-equipped mobile phones suggest the following components:

Find - locate the key for the desired letter and press it; continue for each letter in the word.

Adjust - at the end of the word, if the intended word does not appear, then

(a) press the NEXT key until the intended word appears, or the originally displayed word re-appears, then

(b) if the intended word failed to appear, enter the word using *Multitap*.

For *T9*, the Adjust phase is complex and the strategy to adjust depends on many factors. For one, note that the adjustment occurs at the end of a word, rather than after each letter. Indeed, a "leap of faith" is expected during word entry because the display is unstable and often fluctuates unpredictably. As an example, consider the word "golf ", as entered in *T9*-mode on a Nokia 3210 mobile phone. Figure 9, reading top to bottom, illustrates the required keystrokes and the displayed output at each keystroke.

Keystroke	Display	Comment
4	i	wrong first letter
6	in	still wrong
5	ink	still wrong
3	hold	wrong word appears
*	hole	adjust - wrong word
*	gold	adjust - wrong word
*	golf	adjust - correct word
0	golf	accept word

Figure 9. Entering "golf " in *T9*-mode on a Nokia 3210 mobile phone ('0' is the SPACE key)

The interaction illustrated in Figure 9 is more complex than suggested by a simple keystroke count. Perceptual and cognitive processes are clearly at work as the user considers the system's response to each keystroke. At each keystroke where a response is considered, about 190-260 ms is added to visually perceive and process the choice [7]. If the intended word ultimately does not appear, then the interaction is even more complex.

To examine the performance cost of interaction in the presence of non-dictionary words, we undertook a parametric analysis based on our data. We first obtained the observed entry time for each word in our phrase set at each stage of learning. Recall that our phrase set had 2711 words, of which 1163 were unique (see Figure 3). We extracted the entry time for each word considering only the time to find each letter in the word, t_F. Time to adjust or timeout kill time was ignored; thus, the time should be a reasonable approximation of the *T9* entry time because it is based on only one keystroke per letter.

To accurately model typing and spelling errors, we took the typing and spelling errors from the *Multitap* user study, and mimicked the behaviour of the *T9* implementation in the Nokia 3210. This *T9* implementation attempts to notify the user when a typing or spelling error occurs by beeping when the input does not match any prefix in the dictionary. The beep generally occurs near the end of the word, regardless of where in the word the error occurred. Hearing the beep, the simulated user backspaced to the error (at expert speeds — again favorable to *T9*), corrected the error, and continued.

Our next assumption was that the underlying dictionary contained the vocabulary of our phrase set. Thus, we obtained novice-to-expert predictions under the "all words in dictionary" assumption.

We then removed words from the dictionary in stages, leaving .95, .90, .85, then .80 of the words in the dictionary. Then, we modeled user input with *T9* with the removed words entered as non-dictionary words. The time to enter these words was approximated as the *T9* time plus the *Multitap* time. This is reasonable, since users of *T9*-equipped phones do not have a priori knowledge of whether or not words are in the phone's dictionary. They must enter a word first, then, discovering that it is not in the dictionary, they must re-enter the word in *Multitap* mode.

Our analysis is generous to *T9* in several ways. First, words were removed at each stage systematically, starting with the least-probable entries. As well, we ignored the time for the user to consider and cycle through the alternatives in sets of ambiguous words. The results are shown in Figure 10.

The top line in the figure is the ideal situation where all words entered are in the dictionary. As a reality check,

note that the figures for session 1 (9.3 wpm) and session 20 (21.7 wpm) are very close to the figures cited by James and Reischel [6]. They reported 9.09 wpm for novice *T9* users and 20.4 wpm for expert *T9* users. In their study, all words entered were in the dictionary, so the appropriate comparison is with the top line in Figure 10.

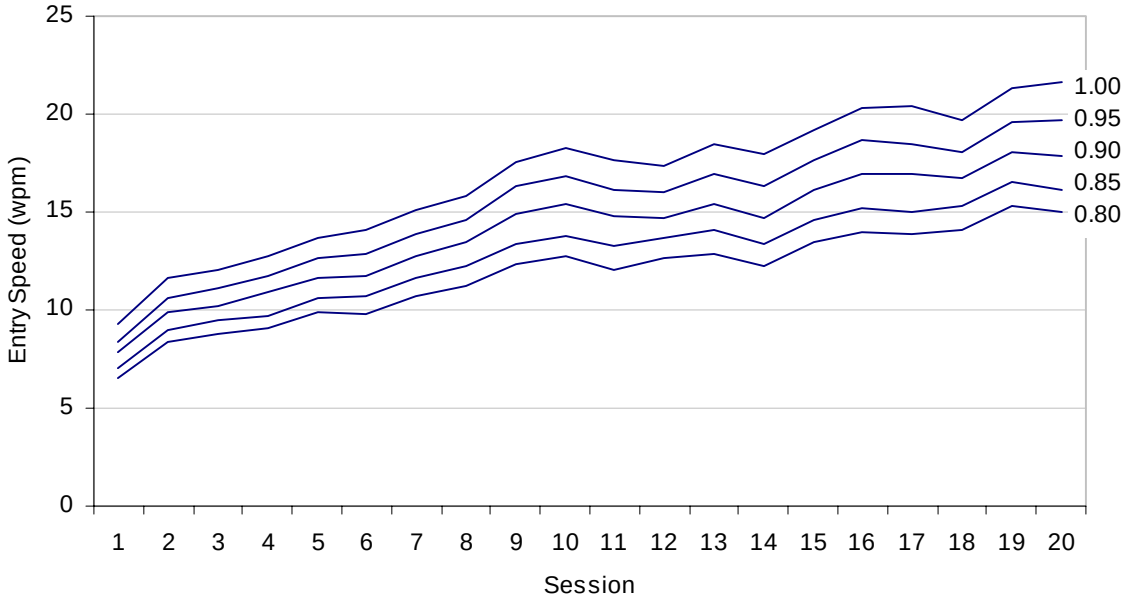

Figure 10. Simulated *T9* analysis for non-dictionary words. Lines show *T9* performance with decreasing ratios of words in dictionary

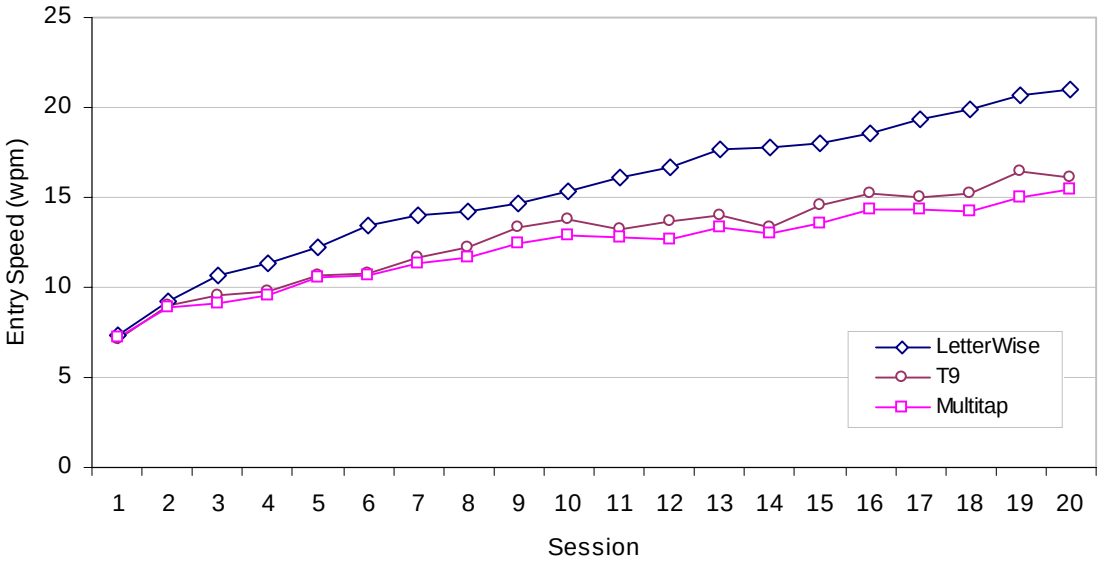

Figure 11. Comparison of entry rates (wpm) with practice for *LetterWise*, *T9*, and *Multitap*. (Note: *LetterWise* and *Multitap* figure are from Figure 6. Simulated T9 figures are from Figure 10 with 0.85 frequency of words in dictionary)

The other lines in Figure 10 represent various degradations in performance in the presence of non-dictionary words. The dashed line labeled "0.85" represents entry wherein 15% of the user's words are not

in the dictionary. In this case, the performance with *T9* is about the same as for *Multitap* (see Figure 11).

If the user is predisposed to use an even higher proportion of non-dictionary words, performance is further degraded, and is well below that for *Multitap*. Of course, at some point users will simply give-up in frustration, and work exclusively in the alternate entry mode. This was observed with at least some participants in Grinter and Eldridge's study with teenagers [3]. If the title of their study is any indication — *y do tngrs luv 2 txt msg?* — a high frequency of non-dictionary words is common, a phenomenon of text messaging they call "evolving language". As another example, a collection of text from the 1988 *Wall Street Journal* containing 20,691,239 words was found to contain not only 8,633,941 ambiguous words, but also 4,007,375 words which were not in Webster's seventh dictionary [2].

MULTILINGUAL INPUT

Languages throughout the world are currently supported in various forms in mobile computing, and this will continue. While the focus in the present paper is on English, the discussions apply to other languages, particularly those based on alphabets. Databases for *LetterWise* are presently available for 35 languages, with databases for other languages under development. See www.eatoni.com for details.

CONCLUSION

We have demonstrated prefix-based disambiguation to be an efficient means for text entry on keypad-based devices such as mobile phones. Keystroke count is reduced by close to 50% in comparison to *Multitap*, and entry rate is higher by about 36% after ten hours of use. Furthermore, the technique is not limited to the entry of words in a stored database, as with dictionary-based entry methods. A simulated comparison with *T9* shows that *LetterWise* and *T9* have similar entry speeds when all words are in *T9's* dictionary, but when as few as 15% of the least common words are missing, *T9*'s speed is similar to that of *Multitap*, and about 30% slower than *LetterWise*.

REFERENCES

1. Bellman, T., and MacKenzie, I. S. A probabilistic character layout strategy for mobile text entry, *Proceedings of Graphics Interface '98*. Toronto: Canadian Information Processing Society, 1998, 168-176.

2. Davis, J. R. Let your fingers do the spelling: Disambiguating words spelled with the telephone keypad, *Avios Journal 9* (1991), 57-66.

3. Grinter, R. E., and Eldridge, M. A. Y do tngrs luv 2 txt msg? To appear in *Proceedings of the European Conference on Computer Supported Cooperative Work - ECSCW 2001*. Amsterdam: Kluwer Academic Press, 2001.

4. Guernsey, L. Playing taps on the cell phone, *New York Times* (2000, October 12), D9.

5. Gutowitz, H. Patent No. 6,219,731, Method and apparatus for improved multi-tap text input. Eatoni Ergonomics, Inc. (2001).

6. James, C. L., and Reischel, K. M. Text input for mobile devices: Comparing model predictions to actual performance, *Proceedings of the ACM Conference on Human Factors in Computing Systems - CHI 2001*. New York: ACM, 2001, 365-371.

7. Keele, S. W., and Posner, M. I. Processing of visual feedback in rapid movements, *Journal of Experimental Psychology 77* (1968), 155-158.

8. MacKenzie, I. S. *KSPC (keystrokes per character) as a characteristic of text entry techniques*, Submitted for publication. 2001.

9. MacKenzie, I. S., and Zhang, S. X. The design and evaluation of a high-performance soft keyboard, *Proceedings of the ACM Conference on Human Factors in Computing Systems - CHI '99*. New York: ACM, 1999, 25-31.

10. Mayzner, M. S., and Tresselt, M. E. Table of single-letter and digram frequency counts for various word-length and letter-position combinations, *Psychonomic Monograph Supplements 1* (1965), 13-32.

11. Silfverberg, M., MacKenzie, I. S., and Korhonen, P. Predicting text entry speed on mobile phones, *Proceedings of the ACM Conference on Human Factors in Computing Systems - CHI 2000*. New York: ACM, 2000, 9-16.

12. Soukoreff, W., and MacKenzie, I. S. Theoretical upper and lower bounds on typing speeds using a stylus and soft keyboard, *Behaviour & Information Technology 14* (1995), 370-379.

13. Zhai, S., Hunter, M., and Smith, B. A. The Metropolis keyboard: An exploration of quantitative techniques for graphical keyboard design, *Proceedings of the ACM Symposium on User Interface Software and Technology - UIST 2000*. New York: ACM, 2000, 119-128.