Optimizing a Personalized Cellphone Keypad

Joonseok Lee[1] and Bob McKay[2]

[1] SMLV Lab, College of Computing, Georgia Tech, Atlanta, GA, USA
[2] SC Lab, Comp. Sci. and Eng., Seoul National University, Seoul Korea
{joonseok2010,rimsnucse}@gmail.com

Abstract. Current layouts for alphabetic input on mobile phone keypads are inefficient. We propose a genetic algorithm (GA) to find a suitable keypad layout for each user, based on their personal text history. It incorporates codes for frequent multigrams.
We optimize for two-thumb use, minimizing the number of strokes, and consecutive use of the same key or the same hand. Using these criteria, the algorithm re-arranges the characters on a 10-key pad. We demonstrate that this arrangement can generate a more effective layout, especially for SMS-style messages. Substantial savings are verified by computational analysis.

1 Introduction

Hardware cell-phone keypad layouts are of two kinds: unambiguous and ambiguous. Unambiguous layouts have enough keys to uniquely map the intended alphabet to the keys – the QWERTY keyboard is an example. It is straightforward to learn, but the small keys can cause problems. Thus ambiguous layouts, with fewer keys than alphabetic characters, have been widely used. Two mechanisms are used to resolve ambiguity: multi-tap and contextual disambiguation.

Multi-tap uses repetition: letters on the same key are distinguished by the number of strokes. Although easy to learn due to the natural ordering, it is inefficient to use, because of the high average number of key strokes. It performs poorly for single-thumb text entry because of the high average thumb movement distance [6]. Equally important, it uses a unigram layout. English (like most languages) has many common digrams and trigrams, so a pure unigram encoding misses important opportunities for efficiency [3, 5–7].

In disambiguation algorithms, the user types only a single stroke per letter. The disambiguation algorithm guesses the intended word statistically. For example, when the user types '63', the algorithm guesses the intended word as 'of', based on English word frequency. If the user rejects this, the algorithm shows 'me' instead. However disambiguation methods increase cognitive loads [7]: users must check whether the intended word has been found. It is even worse if the intended words are not in the dictionary – the user must change to direct input mode and re-type. This is quite frequent, especially in SMS. Nesbat also notes that users may mix languages, – 'Hello Amigo' – so that predictive methods based on standard dictionaries may perform poorly.

We aim to overcome these limitations through personalization and multigrams. Previous proposals have assumed a standard layout and frequency table. However, letter and bigram frequencies differ by user, culture, and personal taste. For instance, 'q' is more frequently used in Quebec, because of its French culture. A personally-optimized layout will allow this generally-rare letter to fall on a better position. We also use multigrams: frequently-used bigrams or trigrams are directly represented. The multigrams are also personalized to the user.

We restrict consideration to English 12-key layout, though the methods can be extended to other single-byte character systems like Spanish. Only 10 keys are used for typing, with two reserved for special purposes, such as mode change.

In the rest of the paper, we provide some background on keyboard layout optimization, followed by a detailed description of our method. We detail a computational evaluation of the design, concluding with a summary of the contribution and planned future work.

2 Background

2.1 Optimizing Mobile Keypads

Disambiguation is widely used in cell-phones. Lesher et al. [5] optimized the disambiguation rate, dropping the keystrokes per character (KSPC) rate to 1.09. Gong and Tarasewich [3] further constrained the assignment to alphabetical order, generating KSPCs from 1.05 (written article) to 1.25 (text messages), based on a disambiguation rate of 98.13% (text) to 95.13% (SMS).

Genetic Algorithms (GA) have often been applied to layout optimization. In predictive systems, How and Kan [4] remapped key layout with a simple GA. They have also been used to optimise multi-tap layouts. Moradi and Nickabadi [6] based their optimization on a frequency sample. The GA minimized average strokes, delay due to consecutive use of the same key, and movement distance, for one-thumb typing. In their metric, typing cost dropped from 2.90 (ABC layout) to 2.25 (optimized). They argued that the theoretical minimum is 1.70. We use their work as a basis for comparison for our multigram approach.

2.2 Personalized Keyboards

All these methods assume that keyboard design should be standardized. Nguyen et al. [8] argue that universal keyboards are unnecessary due to the rise of standard interfaces. Instead, personalized key layouts can be readily downloaded into input devices. They proposed a GA to generate a user-optimal key layout for 10-finger pure-chording devices.

2.3 Genetic Algorithms

We use GAs in two places: choosing multigrams for the layout, and choosing the arrangement of keys. We use steady-state GAs, in which one (mutation) or

two (crossover) parents are chosen randomly from the current population. The relevant operator is applied, producing a child, which competes deterministically (truncation selection) with the parent(s), the least fit being eliminated from the population. We refer readers to a standard text such as De Jong's [2] for a detailed background on GAs.

3 Motivation

3.1 Criteria for Keypad Optimization

We focus on speed-conscious users, and therefore assume two-thumb typing, leading to a variant of the Moradi and Nickabadi fitness function [6]. Three criteria determine layout efficiency: strokes per character, delay due to consecutive use of the same thumb, or of the same key (because a delay is needed to distinguish the two characters). We combine these criteria, defining a three-part function:

$$f_1 = \frac{\sum_l st(l)}{C} \tag{1}$$

$$f_2 = \frac{\sum_l sk(l)}{C} \tag{2}$$

$$f_3 = \frac{\sum_l sh(l)}{C} \tag{3}$$

$st(l)$ is the number of strokes required to type character l, C is the total number of characters in the corpus. $sk(l)$ is 1 if consecutive characters use the same key, else 0, and $sh(l)$ is 1 if consecutive characters use the same hand, 0 otherwise. However, there is a complication. We assume that keys on the left column use the left hand, and vice versa. But the middle column may be typed with either hand – whichever was not used last. In this case, we use parity counting to determine whether to impose a penalty. The overall adaptive fitness function is

$$F_A = f_1 + \beta f_2 + \gamma f_3 \tag{4}$$

f_2 penalizes repeated use of a key. Speed-conscious typists will use the cursor key, so the cost is one stroke; but the cursor key is located inconveniently, so we set $\beta = 1.5$. For f_3, the effect is relatively minor, so we use $\gamma = 0.25$.

In the comparisons , we use Moradi and Nickabadi's distance term and cost function:

$$f_4 = \frac{\sum_l d(l, prev(l))}{C} \tag{5}$$

$$F_M = \alpha_M f_1 + \beta_M f_2 + \gamma_M f_4 \tag{6}$$

with $d(a, b)$ defined as $\sqrt{(r_a - r_b)^2 + (c_a - c_b)^2}$, r and c denoting row and column, and $\alpha_M = 0.7$, $\beta_M = 3$ and $\gamma_M = 1$.

3.2 Multigrams

To improve typing speed, we introduce multigrams. We allow at most four strokes for each key. With ten keys, we have 40 slots. The 26 alphabet keys leave 14 slots for multigrams. We use a two-stage process, optimizing the choice of multigrams, and then allocating positions.

It might seem that we could simply select the 14 most frequent multigrams. However frequent bigrams and trigrams may overlap, excluding this simple approach. Instead, we search for the best combination among the 50 commonest bigrams and trigrams using a first GA featuring eager initialization and operators. The initial 14 multigrams are selected by roulette sampling based on their sample frequency. The crossover operator preserves all multigrams that occur in both parents, then selects other multigrams from either parent in order of frequency. The mutation operator randomly, but at a low rate (1%) replaces the 14 multigrams with others from outside the top 50 (in case our initial assumption was wrong). For the fitness function, we use the sum of the changes in rank of the unigrams, since this tells us how often the multigrams will be located on high-priority key positions, and hence serves as a proxy for the key strokes they will save. We use truncation selection between a child and its parent(s). We run the GA for 1000 repetitions, with a population of 50 (these values being determined through trial and error).

Once we have determined the 14 multigrams, we treat them exactly like ordinary letters, letting them compete for fewer strokes and better positions. If the frequency of a multigram is low, it may be allocated a bad position such as 4 strokes. If this results in more strokes than direct input would require, the multigram is deprecated.

4 The Core Genetic Algorithm

We use a steady-state GA to optimize the keypad layout.

Chromosome Expression 40 genes, each giving the assigned location for a letter: row (1..4), column (1..3) and strokes (1..4). Because we don't use the '$*$' and '#' keys, genes such as (4, 1, 1) and (4, 3, 3) are forbidden.

Initialization Each individual is a permutation of the letters on the keypad, sampled uniformly randomly.

Crossover A gene g_1 is *between* genes g_2 and g_3 if each component lies in the closed interval defined by the corresponding values in g_2 and g_3. A chromosome is *between* two others if all corresponding genes satisfy this property. To construct children lying *between* parents, we use a greedy search of the legal possibilities. In rare cases, we may not be able to satisfy the *between* condition. In this case, we allocate it randomly among the vacant positions.

Mutation We use five exchange operators: swapping columns, rows, keys, and pairs of letters, and swapping strokes on a key.

Selection For crossover, we randomly select two parents, create a child, then use a tournament between parents and child, the loser dropping out of the population. A parent for mutation is chosen randomly, the child always replacing the parent.

5 Computational Analysis

In this section, we compare the theoretical improvement due to personalised multigrams (PM) over ABC layout using Moradi and Nickabadi's cost metric.

5.1 Method

Type	Words	Letters inc. spaces	Description
SMS	554	2766	Recent SMS messages by the first author
FCB	725	3358	Instant messages gathered from Facebook
ART	378	2340	An article relating to a social issue [1]

Table 1. Profile of Archives used in Experiments

We used three kinds of text (Table 1). Each archive came from s single author. The first uses recent Short Message Service (SMS) messages of the first author. The second uses Facebook (FCB) postings. The last is an article (ART) from TIME magazine [1], which should reflect general usage, but also include characteristics of the author in word choice, voice, and use of pronouns; but it does not contain the abbreviations and emoticons typical of SMS.

Parameter	Value	Parameter	Value
Number of Trials	50	Population size	50
Algorithm type	steady-state	Number of Evaluations	50,050
Selection	Parent-child tournament	Mutation rate (5 types)	0.01
Elite Size	1	Crossover rate	1

Table 2. Experimental Parameters

Parameter Settings We used 50 trials for each treatment, and the same number of evaluations (in our case, keyboard configurations) in all experiments. Parameter Settings are shown in Table 2.

5.2 Result and Analysis

Text	Multi-gram	ABC pad	Random Keypad		Optimized Keypad	
			Best	Average	Best	Average
SMS	With	–	1.90	2.03 ± 0.05	1.54	1.64 ± 0.03
	W/O	2.41	2.40	2.50 ± 0.04	1.98	2.02 ± 0.01
FCB	With	–	2.07	2.14 ± 0.03	1.71	1.78 ± 0.02
	W/O	2.37	2.40	2.54 ± 0.04	2.01	2.04 ± 0.01
ART	With	–	2.12	2.24 ± 0.04	1.77	1.82 ± 0.02
	W/O	2.66	2.53	2.65 ± 0.05	2.07	2.10 ± 0.01

Table 3. Layout Comparisons, based on Moradi metric

Comparison with Moradi and Nickabadi We compare our approach with Moradi and Nickabadi. Table 3 (top) compares five layouts using their cost function: ABC, random (with/without multigrams), and optimized (with/without multigrams). Optimized layouts with multigrams always perform best. The ABC layout has similar (slightly worse) efficiency than random layouts. Optimized layouts show better efficiency in every setting, as Moradi suggests. Efficiency is improved by multigrams, in both random and optimized layouts. These results are consistent with those of Moradi and Nickabadi [6] (standard layout 2.90, optimized layout 2.25), based on part of "Harry Potter 5 - The order of the Phoenix." Moradi mentions 1.70 as the theoretical optimum for unigram layouts. Table 3 shows multigram layouts can exceed this limit.

We statistically evaluated the effectiveness of multigrams using the Wilcoxon rank sum test, with $m = n = 50$. In every case, multigrams made a difference significant at the 10^{-15} level.

Influence of Text Archive In all cases, the algorithm found better fitness values for SMS than for FCB or ART.Thus the algorithm is especially relevant to text such as SMS on mobile phones. Gong and Tarasewicz [3] also make such a comparison, but their design shows lower efficiency on SMS than written or spoken language. Thus this approach seems more relevant to mobile phone applications.

Keypad Layouts Figure 1 shows the resulting layouts. In this figure, light-colored multigrams are deprecated, as they do not improve typing speed compared to use of single characters.

6 Discussion

Contributions The PM layout promotes higher typing speed through two personalized features: optimized letter arrangement and multigrams.

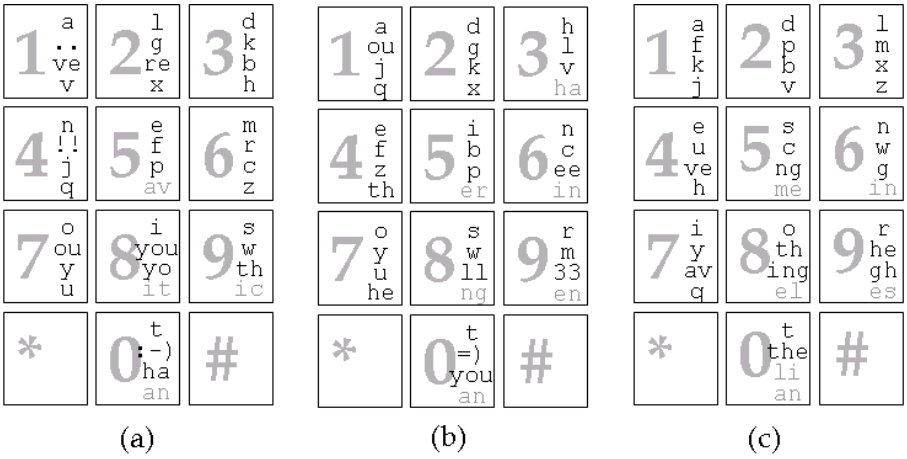

Fig. 1. Optimized Keypad Layouts from (a) SMS (b) FCB (c) ART

Further Work Changing patterns of use may change the optimal layout. But complete re-optimization is likely to generate very different layouts, imposing high learning loads on the user. Alternatively, the amount of change from the previous layout could be incorporated in the fitness function, to minimize the cognitive load while maintaining high typing speed. In general, the issue of how to maintain high typing speed in the face of changing use patterns, while also minimizing cognitive load, is a challenging research topic (not only for mobile phone keypads), worthy of a substantial research investment.

We would like to extend our experiments to a larger number of people with more varied backgrounds. We need to test with more people with a wider range of characteristics, but the current protocol imposes a substantial effort on subjects, and over a long time. To overcome this, we have designed a new experimental protocol. We will be running detailed user study with a larger and more varied pool of participants in the near future.

7 Conclusion

We have introduced an optimization algorithm for personalized mobile phone key layout using multigrams. Genetic algorithms are used to find both the best combination of multigrams, and the best key arrangement. The fitness function takes into account the average number of strokes required to type text, consecutive strokes on the same key, and consecutive use of the same hand. We thus found efficient multigram layouts for three kinds of archives: instant messages as used in cell phones; social network services; and general articles. Fitness values for these best layouts are significantly smaller than for layouts without multigrams, and for unoptimized layouts – including the current alphabetically-ordered layout.

The general method is almost independent of language or character set – it can be applied to any single-level alphabetic language. With minor adaptations, it can handle multi-level alphabetic languages such as Korean, and unambiguous representations of ideographic languages (e.g. wu-bi method for Chinese), but not hgihly ambiguous representations (pinyin for Chinese).

In comparison with previous work, we show that our design substantially improves efficiency. The advantage is particularly large for SMS texts.

References

1. Altman, A.: Why air travel is about to get worse. Time Magazine (October 2009), http://www.time.com/time/business/article/0,8599,1929324,00.html
2. De Jong, K.A.: Evolutionary Computation. MIT Press (2006)
3. Gong, J., Tarasewich, P.: Alphabetically constrained keypad designs for text entry on mobile devices. In: CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems. pp. 211–220. ACM, New York, NY, USA (2005)
4. How, Y., yen Kan, M.: Optimizing predictive text entry for short message service on mobile phones. In: in Human Computer Interfaces International (HCII 05). 2005: Las Vegas (2005), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.96.638
5. Lesher, G., Moulton, B., Higginbotham, D.: Optimal character arrangements for ambiguous keyboards. Rehabilitation Engineering, IEEE Transactions on 6(4), 415–423 (Dec 1998)
6. Moradi, S., Nickabadi, A.: Optimization of mobile phone keypad layout via genetic algorithm. In: Information and Communication Technologies, 2006. ICTTA '06. 2nd. vol. 1, pp. 1676–1681 (2006)
7. Nesbat, S.B.: A system for fast, full-text entry for small electronic devices. In: ICMI '03: Proceedings of the 5th international conference on Multimodal interfaces. pp. 4–11. ACM, New York, NY, USA (2003)
8. Pham, T., Kim, K., McKay, B., Nguyen, X.H.: An adaptive, personalised chording keyboard. In: 2009 Korean Human-Computer Interface Conference. pp. 245–252. The HCI Society of Korea (Feb 2009)