An Adaptive, Personalised Chording Keyboard

Tuan Pham*, Kangil Kim**, Bob McKay**, Xuan Hoai Nguyen ***

Abstract We present a design for personalisation of a chording keyboard. There are two primary design goals. Firstly, the keyboard layout should be easy to learn, and easy to use, taking into account the background and physical constraints of the user. Secondly, the keyboard layout should be readily extensible, based on the previous behaviour of the keyboard user. The design proposal accomplishes these goals, and can be simply implemented on cost-effective hardware. In addition, we present preliminary experimental results on optimising the initial keyboard layout.

핵심어: Keyboard, Chording, Text Entry, Adaptive, Learning

This research was partially supported by Korea Research Foundation Grant Number 2007-KRF-331-D00415, and by the Brain Korea 21 Project. Seoul National University Institute for Computer Science and Technology provided facilities for this research. The work benefited especially from discussions with Dr Kim Dong-Kyun of Seoul National University Structural Complexity Laboratory..

*주저자 : 서울대학교 컴퓨터공학과 석사과정생 Minh Tuan Pham e-mail: tuanpm84@gmail.com

**공동저자 : 서울대학교 컴퓨터공학과 박사과정생 Kangil Kim e-mail: kangil.kim.01@gmail.com

서울대학교 컴퓨터공학과 교수 Bob McKay; e-mail: rimsnucse@gmail.com

***교신저자 : 서울대학교 컴퓨터공학과 교수 Xuan Hoai Nguyen; e-mail: nxhoai@gmail.com

1. Introduction

This research is motivated by four primary considerations:

1. That current high-speed text input modes are insufficiently portable and adaptable for use in active situations (for example qwerty [1]and dvorak [2] keyboards)

2. That current portable input devices are both inefficient for sustained input, and require too much attention for many portable requirements (alpha-numeric keypads, haptic screen keypads)

3. That standard input interfaces (USB, W-USB, bluetooth) remove the need for input device formats to be standardised (there is no need to use someone else's input device if one's own portable input device can be readily connected)

4. That personalised optimisation and machine learning can generate layouts better suited to the individual than can a one-size-fits-all approach

Thus our emphasis is on optimisation and learning methods, to adapt the coding of a particular physical text input device to the user, rather than vice versa. In this paper, we will report an optimisation and learning framework suitable for this task.

Text input systems generally fall into three classes:

1. Positional input systems (such as qwerty or dvorak including haptic versions of these), in which one position is used for each character thus requiring a large number of keys to generate the ~100 characters required for most natural languages

2. Chording devices, where each finger corresponds to only one key, and multiple simultaneous key presses are used to generate specific characters (thus requiring at least 7 keys to generate 100 characters) such as the stenotype machine [3] traditionally used by court reporters

3. Repetition-based devices, in which characters are distinguished by length and/or repetition of key presses (Morse keying [4] is the most widely recognised of these).

Of course combinations are possible. Thus most current-generation phones use a a combination of positional and repetition modes, while the twiddler 2 keyboard [5] combines positional and chording modes, to allow up to 255 characters from four fingers (or in more recent versions, 1023 from five fingers). Unfortunately the twiddler is no longer in production.

Most current-generation text input devices rely on a fixed coding, and require users to adapt to that coding. The twiddler keyboard is a partial exception: while a standard key-mapping is supplied, the user is free to adapt the key-mapping to suit personal requirements; however no software support for this process is provided.

In the present work, the discussion is framed around a ten-finger pure-chording device (we have specific implementations in mind, but for patent reasons cannot discuss these in detail). However the techniques described here can be readily extended to alternative device structures such as the twiddler. The discussion is also framed around ASCII-based character sets; however preliminary investigation of extensions to other alphabetic character sets (Korean) and to ideographic character sets (Chinese) have also been conducted.

2. Design

We divide the adaptation/personalisation process into two phases:

1. A pre-use phase, in which the user supplies information about themselves and their requirements, which is used to generate an initial, relatively simple and redundant keyboard mapping, optimised to those requirements

2. An in-use phase, in which on-line learning methods are used to suggest potential mapping improvements to the user, and to negotiate installation of those improvements

This paper concentrates on the pre-use phase, though the in-use phase is briefly discussed.

2.1 Pre-use Phase

The task of the Genetic Algorithm (GA) is to optimise the objective which in our case measures the ease of use and learning - over the search space of possible key mappings. In this initial work, we used a relatively simple GA representation, which we describe below.

2.1.1 Representation

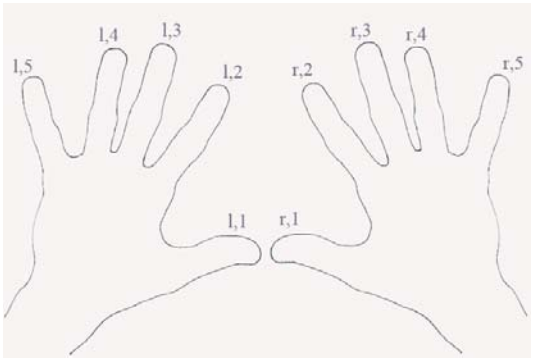

Figure 1: Finger Numbering

A coding for a character is just a string of ten bits, representing the state of the ten fingers (l,1)···(l,5),(r,1)···(r,5). Thus a full coding for a set of k characters may be represented by a k*10 bit array, representing the coding for each of the characters.

2.1.2 Objective Function

The optimisation process uses a genetic algorithm [6] to optimise an objective function. Thus the primary task, for the user, is to specify that objective function. In the current version of the system, we divide the objective into two parts:

1. Ease of learning
2. Ease (speed) of use

That is, for any coding *code*, we define the fitness *fit* to be

$$fit(code) = \mu * mem(code) + \tau * typ(code) \qquad (1)$$

where *mem* and *typ* are functions respectively representing the difficulty of memorising and of typing with *code*, and μ and τ are tuning parameters, intended to reflect the importance, to the particular user, of these two aspects. In this initial version of the work, they were both set to 1.0.

The difficulty of learning, *mem*, was defined as:

$$mem(code) = \sum_{c \in chars} f(c) * D_L(code[c]) \qquad (2)$$

where the sum ranges over the whole character set, and f is the frequency of a character in the given language (English in all these experiments).

The difficulty of learning a particular character depends on two factors. The first factor simply reflects the complexity of the code which must be remembered. In the normal case, it is the sum of the cost of learning each hand (which we define later). However in the special case where the two hands are symmetric, we assume that this is easier to remember; in that case, the cost is simply that of one hand. The second factor gives a similarity reward in the case that the two hands are similar but different. For instance, if the left hand is 00000 and the right 00100, there is a reward of 1. In other words, the reward is equal to the number of bits differing left and right.

Thus, the difficulty of learning a particular character is:

$$D_L(c) = D_C(left[c]) - \varepsilon * R(c) = D_C(right[c]) - \varepsilon * R(c) \qquad (3)$$

if *left[c] = right[c]*. Otherwise (*left[c]* ≠ *right[c]*),:

$$D_L(c) = D_C(left[c]) + D_C(right[c]) - \varepsilon * R(c) \qquad (4)$$

In this formula, ε is a tuning parameter for weighting the importance of the two factors.

The difficulty of typing, *typ*, in turn has two components: the intrinsic difficulty of typing a particular character code, and the difficulty of transitioning between pairs of character codes. Thus we computed it as:

$$typ(code) = diff(code) + \beta * trans(code) \qquad (5)$$

Again, β is a tuning parameter for weighting the importance of these two aspects, initially set to 1.0.

The difficulty of typing a code naturally reflects the difficulty of typing the individual characters in the code, but should be weighted by the expected frequency of typing that character (thus in English, it is far more important for the character 'e' to be easy to type than the character 'q'). We thus used the definition:

$$diff(code) = \sum_{c \in chars} D_C(code[c]) * f(c) \qquad (6)$$

where the sum ranges over all characters c in the character set, and f is the frequency of that character in the given language (in these experiments, English, but more generally, reflecting the choice of the user).

We assume that the difficulties of typing the left-hand and right-hand chords of a character are independent, and that the overall difficulty is just the max of the two, weighted by the handedness of the typist (that is, a right-hander can more readily type right-hand chords, conversely for a left-hander. For simplicity, we assume that the difficulty of hands is symmetric (that is, the same chord is equally difficult for either hand, weighted only by the ease of use of that hand). Obviously, this would not be correct for users with specific difficulties; the representation is readily extended for these cases, but in this paper we only consider the simple case.

The difficulty of typing a particular character is thus:

$$D_C(code[c]) = Max\{D_C(left[c]), \alpha * D_C(right[c])\} \quad (7)$$

where the coefficient alpha is less than 1.0 for right-handers, and greater for left-handers (in these experiments, we used alpha=0.8)

We assume that the difficulty of forming a particular chord relates to the difficulty of using particular fingers, and the difficulty of having adjacent fingers in opposite states. For simplicity in expressing the adjacency condition, we assume that there are two 'extra' fingers, 0 and 6, which are always in the 'unpressed' state:

$$D_C(h) = \sum_{k=1}^{5} h(k) * d[k] * (\delta + \lambda * (ch_h[k-1,k] + ch_h[k,k+1])) \quad (8)$$

Where h is $left[c]$ or $right[c]$ as appropriate, $h(k)$ is the corresponding bit value of h (i.e. 1 for pressed and 0 for unpressed), $d[k]$ is the difficulty of pressing finger k, $ch[i,j]$ is one if fingers i and j in opposite states in h, zero if they are in the same state, and δ and λ are tuning constants. In these experiments, δ and λ were set to 1.0, and d had the values [1.0, 1.0, 1.2, 1.5, 1.7]. These values will be experimentally tuned in future work.

There are two "illegal" exceptions to the above. When both hands are all zero (that is, no keys are pressed), it is impossible for the system to distinguish this from the user simply not having formed that character yet. Secondly, if two characters are allocated the same binary code, it will be impossible for the system to know which character is intended. These two cases are regarded as illegal, and given a very large penalty fitness of 100,000.

2.2 In-use Phase

In the in-use phase (not yet implemented), the system builds a frequency model of the user's typing:

Mis-typings of single characters

Multigram contexts of mistypings

Commonly-typed multigrams (bigrams, trigrams…)

Of course, as the user's typing facility increases, mis-typings become rarer, and the system's ability to use multigram context to predict the correct character increases, so the need for error coding reduces. On the other hand, as users of older chording devices such as the stenotype discovered, single chords for commonly-typed multigrams (such as 'ing', 'ion', 'er' in English) can greatly increase typing speed.

Hence the system, in use, is designed to provide the user with suggestions of error codings that are no longer needed, and may be re-used for multigram sequences. New multigram codings are optimised, as far as possible, to be mnemonic.

3. Experiments

We implemented a Genetic Algorithm based on the GALib package [7], to optimise the objective function previously defined. For these experiments, we limited the character set to the 26 English letters plus the space character, both to simplify the optimisation problem for the experiments, and to avoid complex issues of the relative frequencies of alphabetic and other characters in different types of text. We used character frequency distributions for English from Data-Compression.com [8], We used the simple bit-array representation previously described.

3.1 Evolutionary Algorithm

We used GALib in close to its default settings. The primary aim of these experiments was to determine the roughness of our fitness landscape, and hence the exploration/exploitation trade-off required. Thus we simply used the two default evolutionary operators, point mutation and single-point crossover, at their default rates; that is, the probability of mutation of any given bit is 0.001, and the probability of an individual participating in a crossover is 0.9. However

to give closer control of the stringency of selection, we used tournament selection rather than the default roulette selection.

Preliminary experiments showed that there was a high risk of losing good solutions once found, hence we added elitism to the default GALib settings, with an elite of one.

The experimental settings are shown in detail in table 1.

Trials/Setting	30
Representation	27 * 10 bit array
Population Size	1,000; 10,000
Number of Generations	1,000; 100
Number of evaluations	1,000,000
Mutation operator	Point mutation
Mutation rate	0.001 per bit
Crossover operator	One-point
Crossover rate	0.9
Selection operator	Tournament
Tournament size	3; 5
Elite Size	1

Table 1: Evolutionary Settings

3.2 Experimental Trials

Since the primary aim of this work was to investigate the difficulty of the fitness landscape, we set a fixed budget of evaluations (1,000,000 evaluations) and conducted bi-factorial analysis of the effect of population size and of tournament size. In detail, we used two population sizes (1,000 and 10,000, implying 1,000 and 100 generations respectively), and tournament sizes of 3 and 5. Each treatment was used in 30 trials, using the same set of random-number-generator seeds in each trial.

4. Results

For each trial, we recorded the best fitness achieved over the whole run (because we used elitism, it was also the best in the final generation). The mean and standard deviation, over each treatment, of the best fitness in each trial, is shown in table 2.

Tournament Population	3	5
1,000	106.4±2.8	105.4±2.8
p-value	0.006	0.15
10,000	124.1±1.7	104.9±1.4
p-value	$1.5*10^{-47}$	N/A

Table 2: Mean Best Fitness by Treatment

From the table, it is apparent that the tournament size 5, population 10,000 treatment is the most effective: that is, reasonably stringent selection, combined with a large population, works well for this problem.

To confirm this, we used Student's t-test (one-tailed heteroscedastic) to test significance of the differences; while we did not test for normality (and indeed, the distributions are unlikely to be normal), the p-values for the differences in tournament size are fairly clear; it is not so clear that population size is significant.

The best fitness achieved, over all runs, was 99.66, The corresponding coding is shown in table 3.

A	L: 01000	R: 10100
B	L: 01000	R: 01000
C	L: 11000	R: 01000
D	L: 01000	R: 11100
E	L: 01000	R: 00000
F	L: 00100	R: 01000
G	L: 10000	R: 00000
H	L: 10000	R: 10000
I	L: 00000	R: 10100
J	L: 11000	R: 11100
K	L: 10000	R: 11000
L	L: 00000	R: 01100
M	L: 01000	R: 00100
N	L: 00000	R: 11100
O	L: 01000	R: 11000
P	L: 11000	R: 00100
Q	L: 11000	R: 01100
R	L: 00000	R: 00100
S	L: 00000	R: 10000
T	L: 00000	R: 01000
U	L: 01000	R: 01100
V	L: 01100	R: 10000
W	L: 11000	R: 10000
X	L: 10000	R: 01100
Y	L: 01000	R: 10000
Z	L: 01100	R: 00100
Space	L: 00000	R: 11000

Table 3: Best Character Coding Found

A more detailed understanding of the evolutionary process may be gained from figures 2 and 3, showing the mean (over all trials in a treatment) of respectively the best, and the mean, fitness in a population after a specific number of evaluations.

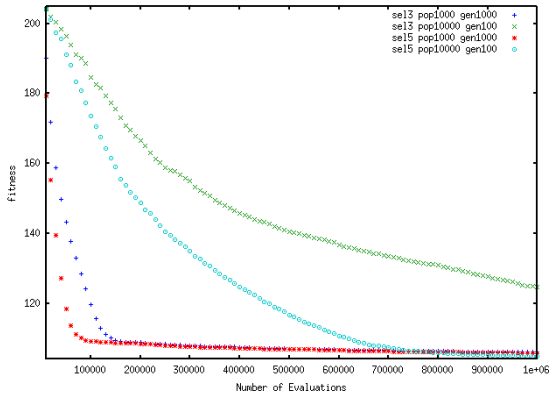

Figure 2: Mean Best Fitnesses (over all trials) by # of Evaluations

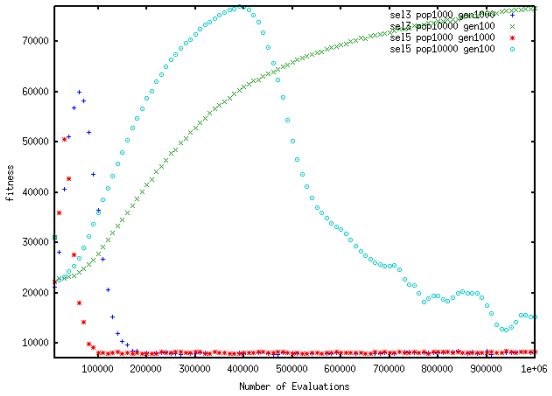

Figure 3: Mean Average Fitnesses (over all trials) by # of Evaluations

These figures have some interesting implications. Firstly, it is clear that diversity is an issue for the

smaller-population runs, and that in these runs, search has essentially stagnated by about 100,000 evaluations (100 generations). Secondly, it seems that the most successful setting (population 10,000; tournament size 5) also generates very large numbers of illegal codings (the mean fitness is too high to have been generated in any other way).

5. Discussion

In the first generation, most runs have an average fitness value of around 30,000, implying that around 30% of the individuals generated correspond to illegal codes. In that same first generation, most runs have a best fitness in the range 200-220. Thus the improvement in fitness achieved by the evolutionary algorithm is quite substantial. It seems clear that substantially improved codings can be obtained through evolution. It also seems clear that further progress can be made through improving the evolutionary settings.

The large number of illegal codings arising in the most successful runs leads to a strong suspicion that the global optimum is difficult to find at least partially because it is closely surrounded by illegal codings. This is not particularly surprising, because the ease-of-use component of the fitness function favours the codes for characters that often occur in digrams being close together (ideally, one bit apart), so that a single bit change can then change the coding into an illegal one. In the short term, this problem should be greatly ameliorated by incorporating local search into the algorithm. In the longer term, once we incorporate error handling into the fitness function (see below), this is likely to move global optima further away from illegal (double coding) codes, reducing this aspect of the difficulty of search.

6. Conclusions

6.1 Assumptions and Limitations

The primary limitations of this work lie in the fitness functions used. These are essentially subjective, derived from our own experience and introspection. There has, so far, been no experimental verification of their validity. Of course, such validation is essential for the future of this project; but experimental validation requires physical realisations, which we do not yet have. Thus this project was a first stage, intended to demonstrate the viability of the approach, and provide the justification for implementing physical realisations. Thus from another perspective, our key assumption is that, even if these particular fitness functions require amendment, they are representative of the kind of fitness function, and the likely optimisation difficulty, required for this project.

A second key limitation lies in the representation and genetic operators used. Applying a GA is not merely a matter of naively choosing a representation in a GA package, and then blindly applying the evolutionary operators it supplies. Rather, the representation and operators must be tuned based on

the implementors' knowledge of the problem. We are very aware that improved representations and operators are required. Within the current encoding, mutation operators that permute the use of specific fingers in a group of neighbouring character codes would be desirable, as would a simple exchange mutation, permitting two characters to exchange their encoding. Similarly, recombination operators based on grouping characters together based on their distance, then exchanging them as a group, are more likely to be effective than the random exchange given by uniform-random single-point crossover.

6.2 Further Work

This paper reports on the pilot stage of what we intend to be a much larger future project. Future work will concentrate on a number of different aspects:

- Further optimising the GA settings and representation. It is unlikely that the current GA representation is optimal, and we will investigate different encodings. It seems likely that GA performance could be substantially improved by:
    - additional, more problem-specific mutation and recombination operators
    - further tuning of algorithm parameters
    - diversity mechanisms in particular, fitness sharing based on Manhattan distance
    - local search, since the fitness landscape is rough on a coarse scale, but may be relatively smooth on a local scale
- Extending the fitness function to incorporate error correction/detection capability. This is readily incorporated, since one-bit error detection simply requires that the Manhattan distance between characters be at least two bits, while error correction requires a three-bit distance. Unfortunately 10 bits does not give sufficient code space for full 1-bit error correction on the whole ASCII keyspace, but we believe it will be feasible to achieve 1-bit error correction for common alphabetic characters, combined with 1-bit error detection for less common characters. However there is an important further issue: the error correction objective is likely to be in substantial conflict with the ease-of-typing objective, hence the fitness landscape is likely to deteriorate with the inclusion of this objective.
- Evaluating and tuning the fitness functions in real-world application, based on purpose-built hardware. We already have designs for realisations of this hardware, and expect to be able to build prototypes within the next few months.
- Implementing the In-Use phase of the project

6.3 Summary

In this paper, we have described a project aiming to design and build an adaptive, personalisable chording keyboard, intended for portable and mobile use. The ultimate aim is to build a keyboard mechanism that will be highly portable, easy to learn, and easy and efficient to use. We have presented experimental results suggesting that evolutionary algorithms will be capable of performing the optimisation required, and thus that the ultimate aim of a personalised keyboard layout, taking into account the background knowledge, physical abilities and intended applications of the user, is realistic.

Acknowledgements

This research was partially supported by Korea Research Foundation Grant Number 2007-KRF-331-D00415, and by the Brain Korea 21 Project. Seoul National University Institute for Computer Science and Technology provided facilities for this research. The work benefited especially from discussions with Dr Kim Dong-Kyun of Seoul National University Structural Complexity Laboratory.

References

[1] E. Clarkson , J. Clawson , K. Lyons , T. Starner, An empirical study of typing rates on mini-QWERTY keyboards, CHI '05 extended abstracts on Human factors in computing systems, April 02-07, 2005, Portland, OR, USA

[2] Cassingham, R.C. The Dvorak Keyboard. Freelance Communications, 1986

[3] W. S. Ireland, "Stenotype Reporter," 3d ed., The Stenotype Company, Indianapolis (1914).

[4] International Morse Code, International Telegraphic Union Radiocommunication M. 1677, 2004,

[5]K. Lyons, T. Starner, D. Plaisted, J. Fusia, A. Lyons, A. Drew, E. W. Looney Twiddler typing: one-handed chording text entry for mobile phones, Proceedings of the SIGCHI conference on Human factors in computing systems 671 678, 2004

[6] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989

[7] GALib website:
http://lancet.mit.edu/galib-2.4/GAlib.html
(retrieved 2009-01-04)

[8] Data-Compression.com website:
http://www.data-compression.com/english.html

(retrieved 2009-01-04)