Cluster keyboard optimisation through genetic programming

Tobias Hjelm
ss06th3@student.lth.se

14 January, 2007

Abstract

A cluster keyboard is a keyboard for a computer or other electronic device where the number of available characters (usually letters) is larger than the number of keys. Each key therefore must contain more than one character (on average), and which character has been entered must be determined somehow. This can be done manually or automatically. Word disambiguation is very popular, where the user does not need to specify each character explicitly, but software determines which word was intended. T9, developped by Tegic, is probably the most widely used method for this purpose, present in many mobile phones where a small keyboard is desirable. By November 2006, two billion devices had been shipped with T9 software (T9 Solutions, 2006).

However, ever so often two or more words "collide", i.e. they correspond to the same key sequence, which makes manual disambiguation (and hence more key presses) necessary. The letters are, generally, grouped alphabetically. This project aims to challenge this convention through re-grouping letters in order to minimise the number of erroneous word propositions, which makes text entry quicker. The regrouping is made using a genetic algorithm which can easily be run on different languages, different alphabets and different corpora. All testing and evaluation has been done using Swedish corpora and the Swedish alphabet. The best result reduces incorrect word propositions by nearly 90%, compared to alphabetical grouping of the letters.

1 Introduction

One of the most common and most popular mobile phone services of today is SMS, that is, short text messages instantly sent from one mobile phone to another. Due to the restricted size of mobile phones, it is desirable that the unit contains as few keys as possible, while providing the possibility to enter any desired word into these messages. Hence each number key on a phone usually doubles as a letter key, each number (except 1 and 0) containing three or four characters from the English alphabet – which has long been seen on regular telephone keypads. In the USA and some other countries, it is common to display phone numbers as words, where each letter maps to a digit, thus making it easier to remember a number (i.e. PHONE-ME is easier to remember than 74663-63).

1.1 Disambiguation

When it comes to entering text using this grouping of letters, there is a need to distinguish which character was intended. For example, the *2* key contains the three letters A, B, and C. The first approach to the problem of entering text into mobile phones was to simply include a time delay, within which the same key could be pressed again in order to cycle through the possible letters. In this case, an A would be generated through pressing *2*, B through pressing *2-2* and C through pressing *2-2-2*. It is easily seen that entering regular text will require more than one keypress per character on average. This is especially inefficient in e.g. the case of S, which is situated fourth on the key *7*: *PQRS*, where S is more than three times as frequent as P in English text, not to mention the infrequent Q. (Letter frequencies, 2006) In addition to this, there is the time delay that has to be waited for each time two consecutive characters in a word are on the same key.

The efficiency of text entry using a cluster keyboard can be measured through the average number of keystrokes per character, the KSPC (MacKenzie, 2002). On a keyboard with only one character per key, such as a regular computer keyboard, the KSPC is exactly 1, except for shifted, accented or otherwise combined characters. Had each key on a mobile phone contained exactly three characters, and each character was equally frequent, it is easily verified that the KSPC would be 2. This is not the case, but it can be considered a rough estimate for regular text.

1.2 Present methods

To get around the large number of keystrokes, several methods have been developped to disambiguate which character has been intended. One of the most common methods is T9, developped by Tegic, which captures a key sequence, where each key is pressed only once, and instantly matches this sequence to a predefined wordlist. In the best case, this can reduce the KSPC to 1. However, the standard of grouping letters alphabetically gives rise to unnecessarily many collisions, where two or more words correspond to the same key sequence. In this case, a certain key must be pressed to toggle between the possible words. This of course increases the KSPC.

An example: In Swedish, the key sequence 7-2 corresponds to the words "på", "så", "sa" and "rå" (as Å has been located on the same key as A although it is not the same letter, but the resemblance is obvious). Both "på" and "så" are very common words. One of them will always need 3 keystrokes, resulting in a KSPC of 1.5. "Rå", in the same manner, yields a KSPC of 2.5, but as it is a far more uncommon word, the impact on text entry efficiency as a whole is not as large. An even worse example is the key sequence *7-6-6*, which corresponds to the words "som", "son", "rom", "snö", "rön", "ron", "sno", "sön" (common abbreviation for "söndag"), and "söm". The last of these words yields a terrible KSPC of 3.67, not to mention the frustration of having to browse through all these words, resulting in a considerable delay. Luckily, the words seem to be ordered by decreasing frequency, which reduces the overall KSPC considerably.

2 Project goal

The object of this project is to investigate whether a different grouping of the letters of the alphabet can reduce the KSPC, and how well. The above examples suggest that perhaps if P and S were on different keys, and/or N, O and Ö were separated, an overall KSPC would be reduced. However, these moves will most likely give rise to new collisions which weren't there before. On a standard telephone keypad, the digits 2-9 are used for letters, that is, 8 keys. Keys *1*, *0*, ***, and *#* are either used for other functions such as punctuation and other characters, case mode, browsing through words, or play no role at all during text entry.

It is not reasonable to think that any grouping of the whole alphabet will be without collisions at all on such a small keypad, but nevertheless the problem can be seen as the problem of minimising the number of word collisions. After all, using a system similar to T9 where the possible words are presented in an order corresponding to frequency, a KSPC of more than 1 is only reached whenever an intended word corresponds to the same key sequence as another, more common, word.

2.1 Solution

Changing the placement of one character may reduce an overall KSPC, but how is the absolute minimum reached? The only way to know for sure is to test all possible groupings of letters on the number of keys, i.e. the problem is in NP. In the case of the Swedish alphabet, consisting of 29 letters, and 8 keys, the number of possible groupings is extremely large, and thus it is not feasible to go through all possibilities systematically. Therefore a solution using a genetic algorithm has been chosen. Genetic algorithms are named thus due to their resemblance to the theory of evolution through natural selection, as proposed by Charles Darwin.

2.2 Limitations

This project has been limited to words only, and hence only deals with alphabetical characters. Numbers, punctuation, other characters and capitalisation are usually achieved differently in different phone models anyway.

Neither has any care been taken to predict words with less keystrokes than the number of letters in a word.

Words not present in the vocabulary have not been considered, partially due to the way the experiment was planned to be carried out, but mainly because of the increased number of assumptions that have to be made and the extra parameters in statistical calculations it would bring along. The difference in efficiency that a new key grouping would make in this regard was also estimated to be negligible, as word prediction is always based on a given vocabulary and words not present in the same will have to be entered in a different way anyway.

3 Genetic algorithms

A genetic algorithm, in general, first randomly creates a large number of possible solutions to a given problem, called *individuals*. The set of individuals is called a *population*. Using a well-defined so-called *fitness function*, the suitability of each of the possible solutions is then evaluated, resulting in a scalar number. Doing this once is called a *generation*. The individuals yielding the best values of the fitness function ("the fittest") in the population are then kept for the next generation Another set of possible solutions is then generated using the first set of solutions, which are modified somewhat as to resemble the first ones. The modification of individuals can be done in one out of two ways:

- *Cross-breeding*, where two different solutions share parts, much in the same manner as the genes of two specimens of any sexually breeding species are combined in their offspring.
- *Mutation*, where an individual is slightly modified, regardless of the properties of any other individual.

By modifying individuals, one hopes that some of the new individuals will represent better solutions to the problem than the individuals in the previous generation. The whole procedure is repeated until a sufficiently good solution has been found. The analogy with Darwin's theory about natural selection should be obvious. (Mitchell, 1997)

As the interruption criterion is usually not a fixed number of generations, it is important that each generation consists of exactly the same number of individuals as the previous one, as otherwise the population will grow or shrink, losing control over memory usage and running time.

It is common to reach a local minimum. Therefore, measures are often taken to try to escape from these local minima, at least with some individuals in the population. Genetic algorithms usually do not aim to reach *the* best solution, but rather to reach a *sufficiently good* solution, when the set of possible solutions is so large that a trade-off between solution optimality and running time and/or memory usage is necessary.

4 Dictionaries

One goal throughout this project has been to develop an algorithm which can be used for any corpus and any (latin-based) alphabet, as well as for an arbitrary number of keys on a keypad. However, as the whole idea arose when examining the T9 function in a mobile phone, the main part of the testing has been done for 8 keys. Focus has also been on the Swedish language and the Swedish alphabet.

4.1 Corpora

Three corpora have been used in order to extract dictionaries:

- The first was a readily available corpus containing the entire bibliography of Selma Lagerlöf, one of Sweden's best-known book authors, who received the Nobel Prize in literature in 1909. These texts have been published between 1891 and 1933, and thus represent a somewhat outdated use of the Swedish language with respect to word usage, spelling etc. They also represent a certain kind of printed language, which is grammatically correct, and thoroughly corrected – thus most probably correctly spelt throughout. This corpus sports about 950,000 words.

- In order to better represent the kind of language which is most probably used in everyday (written) person-to-person communication, such as SMS, text from IRC (Internet Relay Chat) has been collected. A computer was running an IRC client, collecting logs from 20 IRC channels where Swedish was the main language, during a 30-day period. In each channel, a theoretically unlimited number of people can have a conversation at the same time. This text is in no sense corrected or gramatically checked, but rather represents the way that people actually write to each other when they communicate instantly through text. Therefore this corpus should most closely resemble the ways of writing text messages using mobile phones.

  This corpus had to be altered somewhat, as it is very easy for a participant to repeatedly enter the same line of text multiple times. Such doubles were expected to skew the statistics, and therefore doubles have been removed. For the same reason, no nicknames of the conversating people (which are also stored in the logs) have been kept. The size of this corpus is also close to 950,000 words.

- The third corpus resembles the IRC corpus somewhat, in that it contains logs from real, instant internet chat conversations, but this time through the popular internet chat clients ICQ and MSN Messenger. The main differences between this corpus and the former is that this one consists of one-on-one conversations only, and roughly half of the text has been typed by the same person, namely the author of this article, who is somewhat picky when it comes to using correct spelling and grammar. Nicknames have been removed in this corpus as well. This corpus contains just over 1,100,000 words.

4.2 Pre-processing

All three corpora have been pre-processed prior to further use as to contain only words, i.e., all punctuation, numbers, hyphenation, etc. has been removed before further use.

In order to evaluate fitnesses efficiently, each corpus was transformed into a wordlist, where each word is listed along with its number of occurrences.

It is important to note that all words occurring less than three times each were removed. Thus, some misspelt words (except for frequently misspelt ones) and many uncommon words were removed. Words occurring less than three times in a million were considered statistically insignificant enough to be able to be removed without affecting the overall suitability of the fitness function too much. This modification also resulted in roughly two-thirds of unique words being removed in each corpus, thus reducing the amount of work considerably! It showed, which makes good sense, that there are a large number of unique uncommon words, whereas the number of unique common words is relatively small.

4.3 Statistics

Table 1 shows how many words each original corpus consisted of, as well as the number of unique words (i.e. dictionary size) and the average word length in each corpus.

	Lagerlöf	IRC	ICQ/MSN
# words	945,467	952,994	1,102,662
# unique	14,763	15,321	13,169
chars/word	4.34	3.98	3.98

Table 1: Corpus statistics

4.4 Alphabet

The alphabet which is relevant is of course the Swedish alphabet, i.e. the 26 letters from A to Z, plus Å, Ä, and Ö. Four additional, accented letters, have also been given status as separate letters, although they are not usually treated as such in Swedish: À, É, È, and Ü. Neither is very common in Swedish, but they were all used in the used corpora and were added in order to avoid collisions such as “idé” (“idea”) and “ide” (“den”), which would otherwise necessarily have corresponded to the very same key sequence. The total length of the alphabet is thus 33. Words consisting of any alhphabetic characters not among these were simply ignored.

Key	Characters
2	A,B,C,Å,Ä,À
3	D,E,F,É,È
4	G,H,I
5	J,K,L
6	M,N,O,Ö
7	P,Q,R,S
8	T,U,V,Ü
9	W,X,Y,Z

Table 2: Swedish standard grouping of letters on an 8-button keypad

5 Implementation

5.1 Fitness

The fitness function used to assess the possible solutions during the running of the genetic algorithm takes into account the frequency of different words, as the frequency of occurrence of different words varies greatly. (Not surprisingly, most of the most frequently occurring words in the used corpora were short.)

As the aim of the whole project is to minimise an overall KSPC when typing some regular text, the fitness function gives a penalty for each time two or more words “collide”, i.e. when they correspond to the same key sequence. For each key sequence, the list of possible words is ordered in descending order of occurrence, i.e. the most common word needs no toggling between words and is thus free of penalty. The second most common word corresponding to the same key sequence yields a penalty of 1, the third, a penalty of 2, and so on. For each sequence s_j, the contribution to the overall fitness function f is calculated thus:

$$f(s_j) = \sum_{i=1}^{n} occ(w_i)(i-1),$$

where w_i is the ith most common word corresponding to the key sequence, and n is the number of words corresponding to the key sequence.

$$\sum_{s_j \in S} f(s_j)$$

is then calculated, where S is the set of key sequences generating at least one word each.

5.2 Data structures

The program which was developped for the project was written in Java$^{\mathrm{TM}}$. It takes a wordlist, as described in section 4.2, as input. Key sequences are represented using a tree structure, where each node represents a certain key sequence and can contain branches to nodes representing longer key sequences, or a list of words corresponding to that particular key sequence, or both. Each wordlist is represented by a priority queue, sorted by frequency.

5.3 The genetic algorithm

All individuals in the first generation are created randomly by assigning a random key to each character in the alphabet. The number of keys, as well as the number of individuals in each generation, can be chosen arbitrarily at runtime. The value of the fitness function is then calculated for all individuals in the population. Here a choice has had to be made, as to how to generate future generations:

- The best 1/3 of the individuals are kept.
- The best 1/3 are each mutated by assigning two characters to keys different from where they were before, making up 1/3 of the new generation.
- The remaining 1/3 are completely random.
- Measures are taken to avoid doubles.

Note that no cross-breeding occurs. As even small changes in solutions were assumed to be able make big differences in the outcome, there was no need to come up with a way to crossbreed individuals.

After a few generations, many of the fittest individuals in a population tend to resemble each other, more or less closely. This is, in a sense, the whole point: a fit individual which is modified just a little bit very often becomes another, very well fit individual. Doubles must necessarily be removed, as otherwise there is a very large probability that the population will be filled with clones. This scenario is equal to having reached a local minimum, which must be escaped. It is best to modify the fitter individuals just enough, but not too much. By modifying an individual more, a local minimum can be escaped, but the result can just as well be a much worse solution, which is quickly thrown away. This is the hardest part of making a good genetic algorithm. Creating completely random individuals always gives the chance of getting an individual, not close to the current local minimum, which is even fitter, but as the generations evolve, most random individuals naturally are less fit compared to the rest. After much testing of different possibilities, the above scheme proved good.

Another matter which must be taken care of is when to interrupt the process of generating new generations. The current best result is taken to be the measure of how quickly the fitness converges over time, and in general, the results tend to improve rapidly at first, and then converge asymptotically. Plotting the best result over time, the graph usually resembles an $f(x) = \frac{k}{x}$ function (where k is a constant). The interruption criterion in this case was an integer parameter which could be chosen at runtime, and which determines how many consecutive generations must have the same best fitness value before the algorithm is interrupted.

Note: As a genetic algorithm usually produces different results during different runs, regardless of whether the input has been the same, the exact size of the population and the interruption criterion are not crucial parameters to which result is attained. Trial and error is often an easy way to determine which values are good, and, in this case, a population size of 150-200 with an interrupt criterion of 15-30 consecutive generations produced results within a close range for each corpus.

Running time for these results have been 37-65 minutes on a modern PC as of 2006, running Java$^{\text{TM}}$ under Debian GNU/Linux.

5.4 Running

The algorithm has been run at least ten times on each of the three different wordlists, described above. Thus the chance of being close to a global minimum is good, as each time a different solution with a similar value of the fitness function is found. At the same time as the best character grouping is determined, it is also tested on a certain wordlist. All serious testing has been done on an 8-key keyboard, but the program can handle any number of keys. 8 keys was chosen just because that is how many keys are used on a mobile phone, and hence has some connection to reality.

5.5 GUI

A very slim GUI (graphical user interface) was programmed to facilitate evaluation of the results. In this GUI, any grouping of characters can be entered, represented as a string of characters with spaces. Any wordlist can also be used to test the fitness of a certain character grouping on the corpus corresponding to that wordlist. Secondly, the GUI displays the custom keypad, allowing the user to input words by pressing virtual keys using the mouse. Text can also be entered using the computer keyboard, which is necessary for words not present in the dictionary. New words are added to the dictionary.

6 Results

The most interesting comparison in this case is to compare the evolved key groupings to the alphabetical grouping, so that has been done. Also, each of the three different best solutions, corresponding to the three different corpora, has been tested on both remaining corpora.

The resulting key groupings can be seen in Appendix A.

6.1 Similarities

The proposed solutions vary much between different runs of the program, but looking at the three different solution tables, some tendencies kan be discerned.

It is not surprising that the vowels are well-spread, as it is much less likely that a vowel and a consonant can replace each other in a word, than two vowels or two consonants.

Another similarity is that common letters also seem to spread among the different keys. This makes perfect sense, since common letters cause more collisions than uncommon ones. It seems as though the total frequency of each key (i.e., the total number of occurrences of the letters on the same key) may very well be evenly distributed, although this has not been confirmed.

Although such general patterns can be seen, no particular pairs of letters seem to group together more often than any other during different runs.

6.2 Efficiency measure

Evaluation of the efficiency of the different groupings has been made the same way as the fitness function is calculated (described in section 5.1). Each time the word toggling key has to be pressed in order to reach the desired word, that gives a penalty of 1. As the efficiency has been measured on certain wordlists throughout the project, the case of a word not present in the dictionary has been irrelevant, and hence not included in any calculations. In fact, the efficiency measured is only the efficiency of writing a particular text using a particular key grouping, given that all words are in the dictionary.

The efficiency measure presented below is shown in figures of percent, that is, how many percent of entered words are *not* the intended words, but require an extra key press. However, if one key press is not enough, this adds up to the total.

$$\text{error rate} = \frac{\#\text{ extra keypresses}}{\#\text{ words}}.$$

The resulting error rates can be seen in table 3.

	Lagerlöf	IRC	ICQ/MSN
α	3.06%	9.73%	5.00%
Lagerlöf	0.352%	10.4%	3.28%
IRC	1.05%	6.48%	2.24%
ICQ/MSN	0.496%	8.40%	1.58%

Table 3: Error rates.

The first row represents the alphabetical grouping of characters. Given a text such as the one in IRC chats, 9.73%, or every 10th word, on average, will require one extra keypress. However, using a grouping optimised for that kind of text, this figure decreases to 6.48%, an improvement by a moderate 33%.

The keyboard which has been optimised for the ICQ/MSN conversations reduces the error rate from 5.00% to 1.58%, or one erroneous word in 63, an improvement by a significant 68%.

Nonetheless, the Selma Lagerlöf keyboard must be considered the "winner" in its field, where an optimised keyboard improves the error rate from 3.06% to 0.352%, a stunning 88% improvement!

6.3 Error rate factors

A few properties of corpora, which can affect the reult, are hereby proposed:

- Using a few words many times, as opposed to using many words equally often, would in general generate more collisions, even if the total number of words was equal.

- A large number of different words would also increase the collision rate. However, this makes no significant difference among the three corpora used, as this parameter varies very little between them.

- Longer words, naturally, generate fewer collisions on average. With n keys on a keyboard, the number of possible key sequences of length l is l^n. Hence the collision rate should, at least theoretically, decrease exponentially with the average word length. As $4.34^8 \approx 126,000$ and $3.98^8 \approx 63,000$, this at least explains where half of the collisions have gone.

The Selma Lagerlöf corpus gets rather amazing results using all three evolved key groupings, whereas the IRC corpus only increases its results somewhat – and the Selma Lagerlöf grouping actually decreases its error rate!

6.4 Conclusions

A few conclusions can be drawn from table 3 above. It seems that optimising a cluster keyboard for a particular kind of text can dramatically improve the error rate given that the text entered resembles the one it has been trained on. For each of the three corpora used, it can be seen that the cluster keyboard optimised for that particular corpus gives the lowest error rate.

It is also interesting that the alphabetical grouping of characters is beaten by all three evolved groupings on all corpora, the only exception being the evolved grouping from the

Selma Lagerlöf corpus used on an IRC text, where the evolved key grouping is a mere 7% worse – 10.4% as opposed to 9.73%. These two texts are also the two that were expected to be least like each other. The IRC evolved key grouping is also the one out of the three evolved ones which presented the smallest improvement over alphabetical order, run on the Selma Lagerlöf corpus.

The improvements in error rate that the three evolved key groupings achieve, run on their respective corpora, vary largely. Although none of the results is probably a global minimum, they give a very good clue about the lower theoretical limit of the error rate. The error rate for typing the Selma Lagerlöf corpus could be cut down by nearly 9/10 using an adopted key grouping on only 8 keys, whereas the respective error rate reduction for the IRC conversations is a modest 1/3. This is not very surprising, as the published style of writing, and moreover written by only one author, very well could be more homogenous.

Last but not least, it is important to point out that not too much attention should be given to the actual evolved keyboard layouts, but rather to the fact that an improvement *is* possible to achieve through this method, and differences in performance between different layouts should be mainly referred to the properties of the respective corpora.

7 Suggestions for further study

This article has described a rather limited experiment, carried out during a limited period of time. No real-life testing has taken place, but only theoretical results have been considered.

Further studies within the area could include the actual, experienced effectiveness of switching layouts, as learning a new system will most likely produce a slower result to begin with. Also, an alphabetical layout is more intuitive, which makes it easier to find characters when the user has no previous experience. A study where time, instead of number of keypresses, is considered, may prove interesting.

While re-arranging characters on the keypad, one could just as well arrange the characters within each key according to frequency, thus facilitating entry of words not yet in the vocabulary.

Care could also be taken to investigate the effectiveness of including other characters than letters, such as apostrophe or hyphen, in words, and placing them among the letters.

Finally, it may very well be a good idea to combine an optimised key grouping with other word prediction methods, such as word bigram and/or grammatically based word prediction methods, or a method where a word is predicted by its initial letters.

References

Letter frequencies. http://en.wikipedia.org/wiki/Letter_frequencies

I. Scott MacKenzie. 2002. *KSPC (keystrokes per character) as a characteristic of text entry techniques.* http://www.yorku.ca/mack/hcimobile02.PDF

Tom M. Mitchell. 1997. *Machine learning, international edition. pp. 249-260.* McGraw-Hill, Singapore.

T9 Solutions. http://www.tegic.com/

A Resulting key groupings

Below are tables of the best key groupings found during the experiments.

A.1 Selma Lagerlöf

Key	Characters
2	A,G,M
3	À,É,N,U
4	B,Q,T,Ö
5	C,D,È,O,W
6	E,F,K,W
7	H,I,L,Ü,X,Z
8	J,P,R,Ä
9	S,Y,Å

A.2 IRC

Key	Characters
2	A,F,Q,X
3	À, M, O, Ü
4	B,D,T,Y
5	C,É,N,U,Ö
6	E,S
7	È,K,P,R,V,Ä
8	G,J,W,Z,Å
9	H,I,L

A.3 ICQ/MSN

Key	Characters
2	A,M
3	À,É,F,T,Ä
4	B,S,Z,Å
5	C,G,I,J,X
6	D,H,Q,U,Y
7	E,L,V,W
8	È,O,R
9	K,N,P,Ü,Ö