Available online at www.sciencedirect.com

ScienceDirect

European Journal of Operational Research 179 (2007) 838–846

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

Multi-objective optimization of mobile phone keymaps for typing messages using a word list

Kenneth Sörensen *

University of Antwerp, Faculty of Applied Economics, Prinsstraat 13, B-2000 Antwerp, Belgium

Received 13 April 2004; accepted 7 March 2005
Available online 1 December 2005

Abstract

Most mobile phones today offer the option of using a word list to ease the typing of short messages (SMS). When a word list is used, a word is input as a sequence of digits by pressing the key corresponding to each letter once. The word list is used to look up the word(s) that correspond to this sequence of digits. This paper describes how a mobile phone keyboard layout can be obtained that is better suited for typing such messages. Two objectives are considered: the total cost of typing, and the total cost of word clashes that occur when a certain digit sequence corresponds to two or more words in the word list. A multi-start descent algorithm is developed to obtain a Pareto set of solutions.
© 2005 Elsevier B.V. All rights reserved.

Keywords: SMS; Mobile phone; Keyboard; Multi-objective optimization; Multi-start descent

1. Introduction

For many young people, SMS (short message service) has become one of the primary modes of communication. According to GSM Association, a consortium of mobile phone operators, the more than one billion GSM users in 205 countries worldwide sent 45 billion messages in February 2004 and are estimated to send over half a trillion messages in 2004 [2]. The SMS interface however, i.e., the mobile phone keyboard, was not originally conceived for typing text, but for entering telephone numbers. The distribution of the letters over the 10 numeric keys, shown in Fig. 1, was "optimized" for easy reference.[1] This alphabetic

* Tel.: +32 3 220 40 49.
E-mail address: kenneth.sorensen@ua.ac.be

[1] The fact that there are no characters on the 1 key has a historic reason: telephone numbers in the past would not be dialed, but verbally told to the operator. Such "numbers" would usually start with letters, e.g., the first letters of the town or street of the person called. When AT&T put the first dials into operation (in 1919), the equipment could not differentiate between the single pulse of a one being transmitted and the sound of a phone hanging up, unless the phone was already in "dialing mode". As a result, when the first number dialed was a 1, the equipment would hang up. Therefore, no letters could be assigned to the first key [1].

0377-2217/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2005.03.069

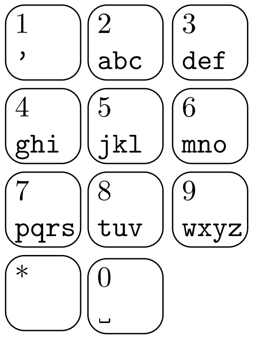

Fig. 1. The standard keyboard layout.

letter order is useful for typing American-style telephone numbers, such as `1-800 FLOWERS` or `1-800 SAVEAPET`, in which the number of characters typed is small. Each character corresponds to the number of the key it is on, e.g., "typing" `FLOWERS` corresponds to dialing `3569377`. On the other hand, the combination `3569377` might correspond to any other word that consists of a letter on each of the pressed keys in the same order. It is the responsibility of the telephone operator to ensure that no other "word" is issued that corresponds to the same telephone number.

Messages transmitted by SMS do not correspond to telephone numbers and ambiguities such as the ones described above have to be avoided. Two systems have been developed to form words on a mobile phone keyboard. In the first generation of message typing, the first letter on a key corresponds to a single press, the second letter corresponds to a double press, and so on. This requires the user to press key `7` four times if an *s* is required and is obviously a very inefficient way to type even the smallest of texts.

The next generation of mobile phone keyboards, and the main interest of this paper, uses a *word list* to distinguish between words. Words are formed by pressing the key corresponding to each letter of the word once, thereby transforming the word into a sequence of digits. The system then searches the word list for words that correspond to this digit sequence and—if at least one word is found—displays the most frequent word. If the user intends a different word than the one found by the system, a special key (indicated with $*$) can be used to switch between this word and the other words that correspond to the digit sequence. For example, to type the word "suppose", the user would type keys `7877673`. However, the word "purpose" also corresponds to this key combination, and is more frequent. Therefore, the system will show the latter word and the user is required to use the special key to switch words. An example of such a technology is AOL's T9® Text Input [9].

Given the fact that the letters on mobile phone keyboards are no longer just used as memory aids, i.e., to remember telephone numbers, but instead are used to type relatively long messages, the question arises whether the ergonomic qualities of the keyboard cannot be improved. Some attempts have been made in the past to improve the physical design of the keyboard, but these do not seem to be able to rival the standard design in terms of popularity. In this paper, we use the default physical design (i.e., the 4×3 matrix shown in Fig. 1), but develop a method to find a better placement of the letters on the keyboard (we will refer to such a placement as *keymap*). We will focus exclusively on the second-generation input systems, that use a word list.

The keymaps obtained in this paper are intended only for typing messages in the English language. For different languages, however, the same approach can be used with a different word list. The keymaps that result from the same analysis using, e.g., a French corpus are most likely not the same as those obtained for English. This does not constitute a serious problem as the "physical" keymap is usually a small piece of plastic that can be removed and replaced, i.e., nationalization of mobile phone keyboards is not an insurmountable problem. This does not mean that a user will have to switch keymaps when typing a message in a different language. The English language keymap can still be used to type—say—French messages using a French word list, it will only be suboptimal for this language.

A second note is that an "optimized" keymap can still be used to type American-style telephone numbers. In this case, the input system can translate the word typed into a telephone number by remembering which letters are on which keys in the standard keymap. That is, letter 'A' should be transformed into digit 2, whatever key it is on.

2. Literature review

To our knowledge, this is the first paper that addresses this issue. Related research has focused on finding a better typewriter keyboard layout. It is widely believed that the standard *QWERTY* layout—introduced by Christopher Sholes in 1873 for the typewriters produced by E. Remington & Son—was originally designed for slow typing as this would prevent the mechanic parts of the typing machine from getting stuck. It is similarly claimed that the *Dvorak* keyboard is superior. Without wishing to enter this discussion, we just mention that both claims are disputed (see, e.g., [12]). The assignment of letters to the keys of a keyboard has been modeled as a quadratic assignment problem [3,11]. A recent approach uses an ant colony optimization method to optimize a keyboard with respect to several ergonomic objectives [5].

A more closely related problem [13] is—given a finite word list—to find a mapping of a set of letters to a set of integers to minimize the number of ambiguities (i.e., words of integers that correspond to more than one word in the word list). The authors prove the NP hardness of this problem and develop a fast learning automaton solution method. The problem solved in this paper is fundamentally different from our problem (and therefore considerably less useful for the optimization of SMS keymaps) in two important ways. First, the problem solved in this paper considers an ambiguity to be more important if the words involved appear more frequently in the language. Secondly, [13] only minimize ambiguity, whereas we simultaneously minimize the total cost of typing.

The problem discussed in this paper is a bi-objective one and should be solved using a multi-objective optimization technique. Given the difficult mathematical structure of the objective functions, it is not very likely that an exact approach will be able to determine the set of non-dominated solutions. We therefore use a metaheuristic approach and develop a multi-start descent algorithm that uses a weight factor to determine the direction of the search. Many multi-objective metaheuristics have been proposed, most of them based on a single-objective metaheuristic such as simulated annealing [4,14,15] and tabu search [7,8]. For a recent tutorial on *evolutionary* multi-objective metaheuristics, we refer to Zitzler et al. [16].

3. Problem description

This paper discusses how a better keyboard layout can be found when using an SMS input system that uses a *word list*. Essentially, the word list contains a set of words, ordered by the frequency in which these words appear in the language. As the discussion in the introduction points out, typing words on such an input system corresponds to transforming these words into a sequence of digits (integers in the range $[2,9]$) (remember that key 1 does not contain any characters and that key 0 is used as the space bar). As a given sequence of integers corresponds to many potential sequences of characters, a word list is used to look up which character sequences correspond to an existing word. If at least one word corresponds to a given integer sequence, the most frequent word is shown on the screen. If this word is not the same as the original word intended by the user, a special key allows him or her to cycle between all words that correspond to the same sequence of integers.

A good keymap should have two properties. First, the effort required to type an average message should be as small as possible. A message is usually typed using only one finger (usually the thumb) and therefore this effort is determined by the amount of thumb movement that is required. Since the number of key-presses for a given word is a constant regardless of the keymap, the amount of movement is determined only by the way the characters are laid out onto the keyboard. Characters that frequently occur in close proximity (such as "th") should appear close together, preferably on the same key. It is important that words that occur frequently are easy to type, whereas this is less important for very infrequent words. Secondly, as little as possible word *clashes* should occur. A clash is defined as the fact that two or more words in the word list correspond to the same integer sequence. The absence of clashes is more important for frequent words than for infrequent ones, but the frequency to take into account is that of the second frequent word (and the third frequent, fourth frequent, etc.) corresponding to some integer sequence. Indeed, as the most frequent word corresponding to a certain key combination is displayed, the clash goes unnoticed if this was the intended word. On the default keymap, in use on most mobile phones, some of the most frequent words clash: "am" and "an", "if" and "he", "no" and "on".

The two objectives described in the previous paragraph (typing effort and absence of clashes) are contradicting. Whereas the first objective forces as many letters as possible on the same key, the second objective favors letters that often occur together to be on different keys.

One could argue that cycling between words that correspond to the same integer sequence is done by pressing a key and can therefore be considered to be typing. However, we believe this argument is flawed because typing and cycling are fundamentally different operations. This can be demonstrated by noting that typing words can be done without looking at the screen, whereas cycling between words cannot (unless the entire word list is memorized). Also, the fact that the input system does not come up with the intended word is generally conceived as very annoying.

We now formalize these objectives by making a number of assumptions and developing a mathematical model of the problem.

3.1. Assumptions

1. The keyboard consists of eleven keys. Ten keys are labeled 0 to 9. One key is labeled $*$.
2. 26 letters (a–z) need to be distributed over the nine keys labeled 1 to 9. Each key can contain any number of letters (even zero), but each letter can occur only once.
3. The space symbol ? is on the key labeled 0 and may not move.
4. One of the nine keys labeled 1 to 9 should remain reserved for the special symbol (′). This is necessary to create words that contain an apostrophe (*don't*, *shouldn't*, *you're*, etc.). It is necessary that this symbol is on a separate key because this key is also used for punctuation marks (comma, full stop, colon, etc.).
5. A word is typed by pressing the key corresponding to each of the letters of a word once, in the order the letters appear in the word. This transforms the word into a sequence of integers.
6. The key labeled $*$ is used to cycle between words that correspond to the same key sequence. The position of this key is irrelevant.
7. A *clash* occurs when two or more words in the word list correspond to the same integer sequence. The severity of a clash is determined by the frequency of the words that are not the most frequent word corresponding to an integer sequence.
8. Costs are associated with moving from key i to key j. These costs are assumed to be constant for a given pair (i,j).
9. To determine the cost of a word, it is assumed that all words are preceded with and followed by a space (?).

3.2. Mathematical formulation

Given the finite set of characters L, x_i is the subset of characters belonging to a set i. Without loss of generality, we can assume that $i \in [1, m]$, where m is the number of keys on the keyboard. A *keymap* $X = \{x_i\}$ is a partition of the characters of L over the keys. We require of X that

$$\bigcup_{i=1}^{m} x_i = L \quad \wedge \quad x_i \cap x_j = \emptyset \iff i \neq j.$$

We define the (finite) word list W as the set of words w_j $(j \in [1, \ldots, |W|])$. The frequency of a word w_j is denoted as $f(w_j)$.

When a given word w consisting of the letter sequence $l_1 l_2 \ldots l_n$ is typed on the keyboard, the result $D_X(w)$ is a sequence of digits $d_1 d_2 \ldots d_n$ with $l_i \in x_{d_i}$. Of course, this result depends on the keymap X.

For this specific case, $L = \{a, \ldots, z, '\}$, $m = 9$. We additionally require that $x_i = \{'\} \iff ' \in x_i$, i.e., the apostrophe should be alone on a key.

To calculate the total effort required to type an average message, we introduce a matrix $C = c(i,j)$ with $i,j \in [0, m]$ that represents the costs of moving between key i and key j. Key 0 is a "neutral" key that represents the space bar. We define the cost of typing word w on an input device with keymap X as

$$b_X(w) = c(d_0, d_1) + \sum_{i=1}^{n-1} c(d_i, d_{i+1}) + c(d_n, d_0), \quad (1)$$

with $w = l_1 l_2 \ldots l_n$ and $l_i \in x_{d_i}$.

And the total typing cost for a given keymap is determined by the cost of typing every word in the word list, weighted by word frequency:

$$f_t(X) = \sum_{w \in W} b_X(w) f(w). \quad (2)$$

To calculate the "cost" of word clashes, we introduce the function δ_X. $\delta_X(w_i)$ is a binary function that indicates whether, for a given word w_i, a more frequent word exists that corresponds to the same key combination. Of course, this depends on the keymap X:

$$\delta_X(w_i) = \begin{cases} 1 & \text{if } \exists w_j \in W \mid w_j \neq w_i \\ & \quad \wedge f(w_j) > f(w_i) \\ & \quad \wedge D_X(w_j) = D_X(w_i), \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

For a given keymap X, the total cost of all clashes now can be easily written as

$$f_c(X) = \sum_{w \in W} \delta_X(w) f(w). \quad (4)$$

This is the sum of the frequencies of all words that are not the most frequent word corresponding to their integer sequence.

3.3. Practical calculations

To calculate both objective function values, a list of the 2000 most common words from the Brown corpus [6,10][2] is used. The Brown corpus is a rather large (about one million words) collection of texts taken from several non-academic domains. The absolute frequency (number of occurrences) of the words in the Brown corpus is also given. For example, the most frequent word ("the") occurs 69970 times.

[2] Retrieved from http://www.edict.com.hk/textanalyser/.

The cost f_t of typing can be easily computed by calculating the value of formula (1) for each word in the word list and multiplying the typing cost with the frequency of each word. For large word lists, this may require a significant amount of time. It should be noted that the problem of minimizing the typing cost alone corresponds to a quadratic assignment problem. Using this result, a faster way to calculate f_t is to pre-process the word list and create a square matrix that contains the frequencies of all character pairs, i.e., the total number of times each character pair occurs in all words combined, weighted with the frequency of the words. Combined with the cost matrix C and the keymap X, this matrix can be used to quickly calculate the total typing cost.

The cost matrix $C = c(i,j)$ used in the experiments is given in Table 1.

The cost of word clashes (f_c) is more difficult to calculate and involves building a tree. In this tree, the nodes represent keys. Nodes on the first level represent keys that are used as the first letter of a word, nodes on the second level represent keys that are used at the second level of a word, etc. A flag is added to each node that indicates whether this node represents the final letter of a word. The tree is built by adding each word in the word list to it, in order of decreasing frequency. This is done by first translating a word to its key combination and then adding nodes on each required level if they do not yet exist. The final node added for a given word has its flag set to *true*. When the last letter of the word corresponds to an existing node with the final flag set to *true*, a word already exists

Table 1
Costs of moving from key in the column to key in the row

	0	1	2	3	4	5	6	7	8	9
0	1	8	7	8	6	5	6	4	3	4
1	8	1	3	5	3	4	6	5	6	7
2	7	3	1	3	4	3	4	6	5	6
3	8	5	3	1	6	4	3	7	6	5
4	6	3	4	6	1	3	5	3	4	6
5	5	4	3	4	3	1	3	4	3	4
6	6	5	4	3	5	3	1	6	4	3
7	4	5	6	7	3	4	6	4	3	4
8	3	6	5	6	4	3	4	3	1	3
9	4	7	6	5	6	4	3	4	3	1

that corresponds to the same key combination. Since the words are added in order of decreasing frequency, the word that was already added is more frequent and is shown first. Therefore, the frequency of the new (less frequent) word is added to the clash cost.

Fig. 2 shows a tree built using the standard keymap (Fig. 1) with words "the" (843), "to" (86), "free" (3733), "vote" (8683) and "does" (3637). Nodes with the flag set to *true* are depicted as a rhombus, other nodes as a circle. When the word "end" (363) is added to the tree, this does not give rise to a clash, since the node 3 (below the 6) is not a final node. Instead, adding this word changes the status of this node to final. When adding the word "ends" (3637) however, a clash is detected as the status of the leftmost 7 in the tree is final. Therefore, the frequency of the word "ends" is added to the clash cost.

Even when using a corpus of limited size (2000 words), the standard keymap features a large number of word clashes, including some of the most frequently used words (e.g., if–he, no–on, me–of). In some cases, three or more words correspond to the same keyboard combination (e.g., box–boy–any). The number and severity of clashes is likely to increase when larger word lists are used. Total typing cost of the standard keymap is 17,910,627. Clash cost of this keymap is 21,129.

We should note that the corpus used is most likely not representative of the average SMS conversation. For example, the word "book" appears more frequently than the word it clashes with ("cool"). However, the analysis can easily be repeated using a list compiled from a more representative corpus. Moreover, this does not invalidate the result that the standard keymap is not optimal as words like "to", "on", "am", "an", "if", "he", etc. most probably appear frequently in SMS conversation too.

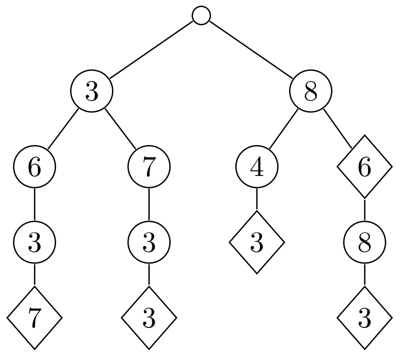

Fig. 2. Detecting clashes.

4. A multi-start descent algorithm for the SMS keymap optimization problem

A simple local search algorithm that is used to solve the problem described in Section 3, is shown in Algorithm 1. The algorithm starts from a random solution and attempts to iteratively improve this solution by putting a character on a different key. Given the fact that some characters appear much more frequently than others, it can be argued that the position of the most frequent letters is more important than that of less frequent letters. The algorithm therefore uses a list of characters, sorted in order of decreasing frequency. For the Brown corpus, this list corresponds to *etaoinsrhldcumfpgwybvkxjqz*.[3] Let $\phi(i)$ denote the *i*th most frequent character.

The algorithm restarts from $n_{\max}$ random solutions and improves them using a local search procedure. The apostrophe is not moved during a local search iteration and therefore remains on the key it was in the initial solution.

The local search procedure proceeds by first attempting to find a better position for the most frequent character (e). This is done by temporarily putting this character on each of the seven remaining keys and calculating the objective function value f. If the solution cannot be improved by moving the most frequent character to a better position, the search continues with the second-most frequent character, then the third-most frequent, etc. When an improved solution is found by moving a character to another key, this move is made and the search goes back to the most frequent character, then the second-most frequent, etc. The local search ends when the keymap cannot be improved by moving the least-frequent letter.

[3] http://www.bckelk.uklinux.net/words/etaoin.html

The objective function value f for a keymap X is determined by a weighted average of normalized typing and clash costs. Normalization of both costs is done by dividing the typing and clash costs of a given keymap by the typing and clash costs of the standard keymap X_s. Normalization is necessary because typing and clash costs are of different orders of magnitude.

$$f(X) = \alpha \frac{f_t(X)}{f_t(X_s)} + (1-\alpha)\frac{f_c(X)}{f_c(X_s)}. \quad (5)$$

The descent method is reinitialized $n_{\max}$ times with different random initial solutions. At the ith restart, the value of α is set to $i/n_{\max}$. Experiments show that this procedure generates a good approximation of the efficient frontier.

Algorithm 1. Multi-start descent for the SMS keymap optimization problem

```
for $i \leftarrow 0$ to $n_{\max}$ do
    Generate a random solution $X$;
    $\alpha \leftarrow i/n_{\max}$;
    $j \leftarrow 1$;
    repeat
        Put $\phi(j)$ on the key that yields the highest $f(X)$;
        If improvement found then
            $j \leftarrow 1$;
        $j \leftarrow j+1$;
    until $j = 26$;
```

5. Results

All programming was done in pascal, and compiled using the freepascal[4] compiler. The code is available from the author upon request.

We set the value of $n_{\max}$ to 10,000, hence 10,001 solutions are generated with values of α equally distributed between 0 and 1. Total running time was about 300 minutes on an AMD Athlon 1100 processor running Linux.

If $\alpha \approx 1$, solutions tend to "degenerate", i.e., only the typing cost is taken into account and all letters end up on one or two keys, resulting in very large clash cost. The reverse is not true, i.e., if $\alpha \approx 0$, solutions are still reasonable. The reason for this is that the clash cost objective has the tendency to distribute the characters evenly across the keys.

Fig. 3 shows 1000 solutions generated with the optimization procedure. The solutions that have a clash cost $f_c > 30{,}000$ can be regarded as "degenerated" solutions because most characters are on a single key.

Fig. 4 shows only the efficient solutions from a set of 10,000 solutions generated using the multi-start descent procedure. Only the solutions having $f_c < 30{,}000$ are shown. The standard keymap is also depicted. As can be seen, many solutions can be found that *dominate* the standard keymap. These solutions are better both in terms of clash costs and typing costs and should therefore be considerably easier to use than the standard keyboard.

5.1. Interpretation of the results

Analysis of the set of efficient solutions with $f_c < 30{,}000$ reveals some interesting properties and guidelines for the development of SMS keymaps.

- All vowels should preferably be on different keys. This is a characteristic of 82% of all efficient solutions.

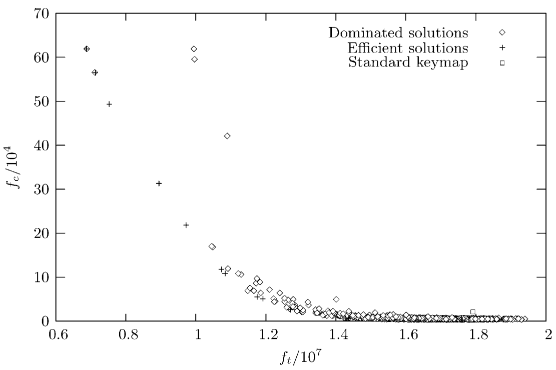

Fig. 3. Typing and clash cost of 1000 solutions and the standard keymap.

[4] http://www.freepascal.org

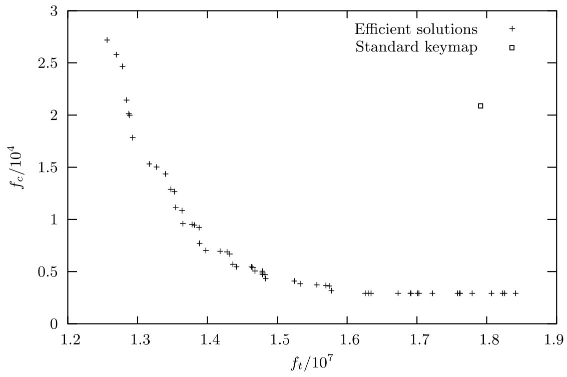

Fig. 4. Typing and clash cost of the efficient solutions and the standard keymap.

- Some characters should be grouped. For example, characters *d* and *f* appear on the same key in 51% of the nondominated solutions. Other good combinations include *os*, *egw*, *ijkpr*, *ht*, *bnx*, *clquvy* and *amz*.
- Over 90% of all efficient solutions use all keys. Some solutions (less than 10%) have one empty key, but these are all solutions with very low typing costs and relatively high clash costs.
- The number of characters per key ranges between 0 and 9. About 25% of all keys in all nondominated solutions have 2 characters, another 25% has three characters. Keys with 1 and 4 characters constitute 17% each. Keys with more than five characters take up 16%. We can conclude that there is no tendency to assign an equal number of characters to each key, but there is no tendency to put an extremely large number of characters on a single key either.

5.2. Some examples

As an example, we show in Fig. 5 the keymap that has the lowest clash cost. Clash cost for this keymap is 2920, a factor 7 improvement over the 21,129 of the standard keymap. It is interesting to note that this keymap has a key with 6 and a key with 5 characters on it.

An example of an efficient keymap that balances clash costs and typing costs is given in Fig. 6. Typing cost for this keymap is 14,787,342, clash cost is 5043.

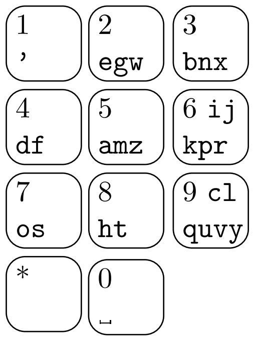

Fig. 5. The keymap with the lowest clash cost.

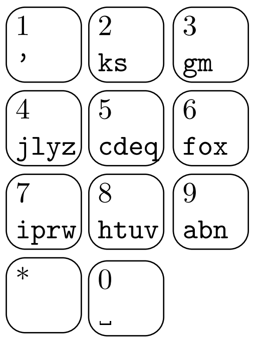

Fig. 6. An efficient keymap.

6. Conclusions and future research

In this paper, we have discussed the need for efficient keymaps to type short messages on a mobile phone. We have formulated this as a combinatorial optimization problem with two objectives: typing costs and clash costs. For a given keymap, typing costs were defined as the total effort to type all words in the word list, weighted by the frequencies of the words in the corpus. Clash costs were defined as the sum of the frequencies of all words that are not the most frequent word corresponding to their integer sequence. We implemented a multi-start local search algorithm for this multi-objective problem and showed how solutions could be found that dominate (i.e., perform better on both objectives than) the standard keymap, in use on almost all mobile phones.

Further research should focus on testing the new keymaps, actually implementing them on a mobile phone keyboard and finding out to which extent they improve the speed and accuracy of typing short messages on a mobile phone keyboard.

References

[1] The Almanac, Why do letters as well as numbers appear on telephone buttons—and why do the letters usually begin on the second button? Available from: <http://www.theatlantic.com/issues/96sep/9609am/9609am.htm>.

[2] GSM Association, GSM Statistics, 2004. Available from: <http://www.gsmworld.com/news/statistics/>.

[3] R.E. Burkard, J. Offermann, Entwurf von schreibmaschinentastaturen mittels quadratischer zuordnungsprobleme, Zeitschrift fur Operations Research 21 (1977) B121–B132.

[4] P. Czyzak, A. Jaszkiewicz, Pareto simulated annealing—A metaheuristic technique for multiple objective combinatorial optimization, Journal of Multicriteria Decision Analysis 7 (1998) 34–37.

[5] J. Eggers, D. Feillet, S. Kehl, M.O. Wagner, B. Yannou, Optimization of the keyboard arrangement problem using an Ant Colony Algorithm, European Journal of Operational Research 148 (3) (2003) 672–686.

[6] W.N. Francis, H. Kucera, Frequency Analysis of English Usage, Houghton Mifflin, Boston, 1982.

[7] X. Gandibleux, N. Mezdaoui, A. Fréville, A multiobjective tabu search procedure to solve combinatorial optimization problems, in: R. Caballero, F. Ruiz, R. Steuer (Eds.), Advances in Multiple Objective and Goal Programming, Lecture Notes in Economics and Mathematical Systems, vol. 455, Springer, Berlin, 1997, pp. 291–300.

[8] M.P. Hansen, Tabu search for multiobjective optimization: Mots, in: 13th International Conference on Multiple Criteria Decision-Making, Cape Town, South Africa, 1997, pp. 6–10.

[9] AOL T9® Text Input. Available from: <http://www.t9.com>.

[10] H. Kucera, W.N. Francis, Computational Analysis of Present-day American English, Brown University Press, Providence, 1967.

[11] S.P. Ladany, A model for optimal design of keyboards, Computers & Operations Research 2 (1) (1975) 55–59.

[12] S.J. Liebowitz, S.E. Margolis, The fable of the keys, Journal of Law and Economics 23 (1990) 1–26.

[13] B.J. Oommen, R.S. Valiveti, J. Zgierski, A fast learning automaton solution to the keyboard optimization problem, in: Proceedings of the Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, ACM Press, New York, USA, 1990, pp. 981–990.

[14] P. Serafini, Simulated annealing for multi objective optimization problems, in: 10th International Conference on MCDM Proceedings, Taipei, Taiwan, 1992, pp. 87–96.

[15] E.L.B. Ulungu, J. Teghem, Ph. Fortemps, Heuristics for multi-objective combinatorial optimization problems by simulated annealing, in: J. Gu, G.C.Q. Wei, Sh. Wang (Eds.), MCDM: Theory and Applications, SCI-TECH Information Services, 1995, pp. 228–238.

[16] E. Zitzler, M. Laumanns, S. Bleuler, A tutorial on evolutionary multiobjective optimization, in: X. Gandibleux, M. Sevaux, K. Sörensen, V. T'kindt (Eds.), Metaheuristics for Multiobjective Optimisation, Lecture Notes in Economics and Mathematical Systems, vol. 535, Springer, Berlin, 2004, pp. 3–37.