The single-finger keyboard layout problem

Mauro Dell'Amico[†1], José Carlos Díaz Díaz[†], Manuel Iori[†] and Roberto Montanari[†]

† DISMI, University of Modena and Reggio Emilia,
42100 Reggio Emilia, Italy

Revised version November 2008

Abstract

The problem of designing new keyboards layouts able to improve the typing speed of an average message has been widely considered in the literature of the Ergonomics domain. Empirical tests with users and simple optimization criteria have been used to propose new solutions. On the contrary, very few papers in Operations Research have addressed this optimization problem. In this paper we firstly resume the most relevant problems in keyboard design, enlightening the related Ergonomics aspects. Then we concentrate on keyboards that must be used with a single finger or stylus, like that of Portable Data Assistant, Smartphones and other small devices. We show that the underlying optimization problem is a generalization of the well known Quadratic Assignment Problem (QAP). We recall some of the most effective metaheuristic algorithms for QAP and we propose some non trivial extensions to the keyboard design problem. We compare the new algorithms through computational experiments with instances obtained from word lists of the English, French, Italian and Spanish languages. We provide on the web benchmark instances for each language and the best solutions we obtained.

KEYWORDS: Keyboard design, Quadratic Assignment Problem, Metaheuristic

1 Introduction

The keyboard represents one of the most popular and effective devices to insert, edit, delete and update long chunk of information. Keyboards used with more than one finger (*n-fingers* later) were firstly introduced more than 100 years ago to support the typists' task. The first keyboard, called Q-W-E-R-T-Y, derived its name from the keys layout. QWERTY layout is nowadays still used to insert users' data into personal computers. The most recent proliferation of Portable Data Assistant

[1]Corresponding author, email: mauro.dellamico@unimore.it, DISMI via Amendola, 2, 42100 Reggio Emilia, fax +39-0522 52 22 30

(PDA), Smartphone and phones have required a strong improvement in the design of ad-hoc input devices, able as the keyboard to allow the input and management of text, e.g., writing of e-mails or messages, allocation of dates in a Personal Information Manager (PIM). Typically, these keyboards could be used either with many fingers or a single finger (*s-finger*, hereinafter). While the n-fingers keyboard has not significantly changed the keys layout, and the major standards have survived despite many alternatives have been proposed, the keyboards for portable systems (both n- and s-finger) are a still open design domain. Many alternatives are available and none of them have definitely dominated the others, in terms of users' acceptance, usage effectiveness, large adoption by the devices developers. Moreover, these portable keyboards are typically used in multitasking conditions (e.g., while walking or driving). Namely, the task of searching and scanning a letter while composing a word could become a problem if it takes too much time. Therefore, to keep this task as short as possible represents a relevant design objective. Proposals for a keyboard layout that is expected to optimize criteria as typing speed, quick learning curve, typing error reduction have been often proposed (see, e.g., Norman and Fisher [34]). The major part of them is based on standard n-fingers keyboard, whereas only rarely the research pointed in the direction of s-finger keyboard, despite the strong interest that this domain could have both in theoretical and applied point of views. This paper find its motivation in the proposals of new s-finger keyboard layouts, through the solution of the underlying optimization problems.

In Section 2 we resume the main problems in keyboard design (both n- and s-finger), giving some hits of the corresponding Ergonomics aspects and discussing the related literature in Operations Research. Section 3 gives a formal description of the s-finger problem we address and shows that it generalizes the Quadratic Assignment Problem. New neighborhoods and properties to speed-up local search algorithms for the keyboard design problem are presented in Section 4. A brief survey of some of the most effective metaheuristic algorithms developed for QAP is presented in Section 5. The next Section 6 presents non-trivial adaptations of these metaheuristic to the keyboard design problem. In Section 7 we introduce the benchmark instances we have obtained starting from real-life word lists of English, French, Italian and Spanish languages. The algorithms performances are evaluated in Section 8. The final Section 9 resumes our work.

2 Study motivation from an Ergonomic point of view

As mentioned above, keyboards can be seen as a challenging design problem, either they are n- or s-finger. For the n-fingers case most of the works have been focused on the tentative to overtake the QWERTY standard. For the s-finger the major task has been trying to improve the typing accurateness, as well as the time spent to edit a text. Typical measures to evaluate n-fingers keyboards are posture structure, discomfort produced, keying force, user acceptance (see, e.g., Sanders and McCormick [37]). For s-finger, measures are referred to the time spent completing a given task, to the errors produced by the users, and to performances modification due to the multitasking conditions. In the rest of this Section, a review of major layouts and literature findings for what concerns both the n- and s-finger cases are reported.

2.1 The n-fingers keyboards: Types and literature

The design parameters for the original typewriter keyboard, namely QWERTY, are obsolete as stated in Hargreaves et al. [23]. The main design problems of this keyboard are the poor shape and the poor key allocation. Because the standard QWERTY was designed for two-fingers typists, it does not efficiently allocate keys to fingers as some fingers are request to perform much work than others. For instance, as demonstrated by Swanson et al. [40], the left hand and fingers (typically weaker than the right ones) are aimed at handling the most frequently used letters.

One of the most basic alternative keyboards is a split QWERTY keyboard, divided into two halves. This allows reduce forearm pronation, to keep low ulnar deviation, to reduce wrist extension. Even from the physical point of view benefits are evident as they revealed a short-term reduction in productivity. Other performance studies for *mini-QWERTY* keyboards (i.e., small versions of the QWERTY keyboard in use in many mobile devices) were proposed by Norman and Fisher [34].

The Dvorak keyboard (see Figure 1) is a U.S.A. keyboard layout patented in 1936 by the educational psychologist August Dvorak. It has been designed on the basis of how frequent is the use of different letters, including the frequency of two, three, four and five sequences of symbols. Although Dvorak layout is more effective than the QWERTY one, it was unfortunately never accepted by the general population. Other proposals came from Claude Marsan, based on the French language, in 1976, 1979 and 1987. Also these keyboards have not been accepted by the users.

Several works in Ergonomics have tried to assess the behavior of the above

mentioned keyboards (see, e.g., Card, Moran and Newell [9]). Recently Operations Research and Ergonomics started working together to design "optimal" keyboards by means of quantitative methods. Egger et al. [15] considered the problem of assigning characters to keys arranged in a pre-specified layout structure. They designed a weighting method based on six-performance indicators: (i) a distribution of the fingers load among all fingers; (ii) the hits number needed to compose a text; (iii) comfort and speed guaranteed when consecutive keys are not hit by the same hand and (iv) fingers; (v) avoidance of great steps among two different keys; and (vi) hits direction that should move from the little finger towards the thumb. At the end, a global score is computed by a weighted linear combination of the six scores. An Ant Colony Optimization algorithm was used to propose a solution for the English and German languages.

In this paper we propose a new study based on the integration of Operations Research techniques and Ergonomics concepts, focused on the optimization of the s-finger keyboards.

2.2 The s-finger keyboards: An open design domain

Both Ergonomics and Operations Research related topics have been applied to study small keyboards and s-finger keyboards. To apply quantitative methods the effort in typing an average phrase with a single finger (or a stylus) has to be defined. The major influencing factors in the movement evaluation are the distance among the keys and the space to be carried out by the single finger. The law introduced by Fitt [16] models the time and difficulties required to move to a target area as a logarithmic function of the ratio between the distance to be covered and the size of the target area. Fitts' law has been used widely as a major rationale to guide design solutions, both for physical and virtual pointing (i.e., with mouse and fingers) and it has been adopted in a huge series of experiments (see, e.g., MacKenzie [31]). In most of them, a correlation coefficient of 0.95 or higher has been found, highlighting that this is a very accurate model, sufficient to guarantee an implicit assess of the layout solutions identified.

Another choice in designing s-finger keyboards concerns complete or incomplete keyboards. A complete keyboard uses as much keys as characters in the alphabet of the language considered, whereas incomplete keyboards have fewer keys, hence more than one character may be associated with the same key. This paper presents optimization methods for the complete case, but, for sake of completeness, we shortly resume the main results for both choices.

2.2.1 Incomplete keyboards

A very common example of incomplete keyboard is the 12 keys ISO keyboard [25] used in most of the mobile phones, see Figure 2-(a). The design of an optimal layout of an incomplete keyboard is a combinatorial problem (called KEYBOARD by Cardinal and Langerman [10]), which asks for a minimum size partition of an alphabet, allowing the users to type any word of a given dictionary so that each word is recognized without ambiguity. In [10] the authors considered the complexity of such a problem, and showed that it is NP-hard even if we only wish to decide whether two keys are sufficient. In [10] several variants of the problem are considered, by: (a) taking into account the possibility that a word is recognized with a small ambiguity (i.e., a given sequence of keys corresponds to more than one word); (b) fixing the number of keys, and (c) imposing that the characters assigned to the same key are contiguous in the alphabet (CONTIGUOUS KEYBOARD problem). Conditions (a)-(c) hold in the cited 12 keys ISO keyboard. Figure 2-(b), taken from [10], reports solutions with minimum ambiguity for the continuous keyboard problem with fixed number of keys, and a dictionary of 885 words. These optimal solutions look quite different than the ISO keyboard. Sörensen [39] extended the 8 key KEYBOARD problem, by using: (i) a *word list* instead of a dictionary (i.e., a set of words ordered by the frequency in which they appear in the language), and (ii) a multi-objective function which considers both the degree of ambiguity and a measure of the effort required to type an average message in the language. A multi-start local search algorithm based on the movement of a single character from a key to another is developed and used to find the Pareto frontier.

2.2.2 Complete keyboards

The FITALY keyboard is an USA *s*-finger keyboard patented by Textware Solutions (see Figure 3). It has been designed for English on the basis of the corresponding words frequency. Others examples of complete keyboards are the ABC layout, OPTI, Metropolis, Hooke, Lewis, and many more presented in the complete review proposed by MacKenzie and Soukoreff [30].

Li, Chen and Goonetilleke [28] consider three layout structures where the position and size of the keys are fixed and the problem is to assign one character of the English language to each key so that the movement time is minimized. The movement is evaluated through the Fitts' law. A simple simulated annealing metaheuristic was used to define heuristic layouts. The authors compared their solutions with other standard layouts, by means of 20 instances taken from BBC and Times. They concluded that the performances of their solutions change a lot with the in-

stance. A FITALY structure with character rearranged in the so called "YLAROF" shape is the most robust solution.

In this paper we use the same objective function of [28], but our problem is more general since we do not adopt an a-priori layout structure. Therefore, we have to choose both the location of the keys and the assigned characters.

3 Problem Description

In this paper we consider the *s-finger keyboard* layout problem, defined by the following assumptions:

(a) all *keys* are identical and they are arranged in a grid (or square lattice) of unit squares, named *locations*. In the following we will indifferently use the word 'key' or 'location' to indicate a unit square;

(b) the *symbols* (or characters) placed in the locations are all different;

(c) each key contains exactly one symbol;

(d) each symbol is assigned to exactly one key.

Assumption (a) imposes that no keys of large size (e.g., the space bar in a QWERTY keyboard) will be used. Assumptions (b)-(d) imply that the keyboard cannot contain duplicate symbols. The problem is to assign the symbols to the locations while minimizing the average time required to write a statement in a given language.

In the following we show that this problem is a generalization of the well known *Quadratic Assignment Problem* (QAP). We identify hereinafter our problem with the acronym SK-QAP.

We can compute the average time to write a statement in a given language, by considering, for each ordered pair of symbols, the frequency in which this pair appears in the chosen language. To obtain the solution value, we multiply this frequency by the time needed to move the single finger between the locations accommodating the two symbols, and we sum up these values over all the ordered pairs of symbols. As seen in Section 2.2, the rationale behind this choice is modeled around the Fitts' law [16], which states that the effort of typing two symbols i and k, consecutively, is

$$\alpha + \beta \log_2 \left(\frac{D}{A} + 1 \right). \tag{1}$$

The parameters α and β have prefixed constant values, D is the distance of the keys which symbols i and k are assigned to and A is the size of the key which k is assigned to.

Note that the contribution to the overall writing time due to a repetition of the same symbol is independent of the assignment of the symbol to a location, hence it can be omitted from the objective function. Moreover the assumption that the keys have equal size implies that the time required to move the finger between two locations only depends on the distance of the two keys, still in accordance with the Fitt's law (the introduction of the size factor, that is significantly expected to affect the measure of the effort, will be considered in further investigations). It follows that in our problem the objective function only depends on the distance between the centers of two locations. For our computations we used the values of the two constants in (1) that have been experimentally determined by MacKenzie, Sellen and Buxton [29], namely $\alpha = 0$ and $\beta = 10/49$.

We can formally describe SK-QAP as follows. We are given an alphabet consisting of a set $N = \{1, 2, \ldots, n\}$ of symbols (or characters) and a set $M = \{1, 2, \ldots, m^2\}$ of locations, organized in an $m \times m$ grid, with m sufficiently large to accommodate all the symbols. We describe a solution through an injective function $\varphi : N \to M$, which maps the symbols $1, 2, \ldots, n$ to the locations $\varphi(1), \varphi(2), \ldots, \varphi(n)$. A solution φ is an assignment of the symbols to the locations and is called, for short, an *assignment*. The set of all possible solutions is denoted by $\mathcal{S}$. We use matrices $\overline{A}$ and B to denote frequencies and distances:

$\overline{A} = (\overline{a}_{ik})$, where $\overline{a}_{ik}$ is the frequency of the ordered symbol pair (i, k);

$B = (b_{jl})$, where b_{jl} is the value of the Fitts function (1), computed with respect to the Euclidean distance from the center of location j to the center of location l.

The problem can be stated as

$$z = \min_{\varphi \in \mathcal{S}} \sum_{i=1}^{n} \sum_{k=1}^{n} \overline{a}_{ik} b_{\varphi(i)\varphi(k)}. \tag{2}$$

Since matrix B is symmetric we can rewrite (2) as

$$z = \min_{\varphi \in \mathcal{S}} \sum_{i=1}^{n} \sum_{k=i+1}^{n} (\overline{a}_{ik} + \overline{a}_{ki}) b_{\varphi(i)\varphi(k)} = \frac{1}{2} \min_{\varphi \in \mathcal{S}} \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{\varphi(i)\varphi(k)}, \tag{3}$$

where $A = (a_{ik})$ is a symmetric matrix with $a_{ik} = \overline{a}_{ik} + \overline{a}_{ki}$. In the following we adopt formulation (3) to simplify the formulas used to evaluate the solutions generated by our algorithms. The value of a solution φ will be denoted by $z(\varphi)$.

3.1 SK-QAP vs QAP

We have already noted that SK-QAP is a generalization of the Quadratic Assignment Problem (QAP), one of the most classical and difficult combinatorial optimization problems. QAP is usually described as the problem of assigning a set of n facilities to n locations, with the cost being proportional to the flows between the facilities multiplied by the distances between the locations. If we assume that a_{ik} is the flow between facility i and facility k, and b_{jl} is the distance between location j and l, QAP is the special case of SK-QAP where φ is a mapping from $\{1, 2, \ldots, n\}$ to itself. In other words φ is a *permutation* of the first n integers. For a complete description of QAP and the relevant solution methods, the reader is addressed to the recent monograph by Burkard, Dell'Amico and Martello [6].

QAP was used by Pollatschek, Gershoni and Radday [36] and Burkard and Offermann [7] to model a keyboard design in a typewriter in which the layout of the keyboard is given, and the goal is to assign the n symbols to the n existing keys. In both studies results have been relevant. In [7], for instance, the heuristic proposed achieved an improvement of 7-10% compared with the standard typewriter keyboard. More recently Egger at al. [15] addressed the same problem, but using an objective function that sums up six Ergonomics indicators (see Section 2.1). Due to the nature of the objective function the resulting problem cannot be modeled as a QAP. The solution approach used in [15] is an Ant Colony Optimization method (see Section 5) and the optimized keyboards achieve a great improvement for all indicators, with respect to standard layouts.

The problem here proposed is more general than the above ones, since there are more locations than symbols and the layout is not known a priori. Note that this is a very relevant difference. Indeed, assuming that ten fingers are used, the time required to type two keys is not monotonously increasing with the distance of the two locations. As a matter of fact it is necessary a very small time to type two keys placed on the opposite sides of the keyboard (e.g., symbols Q and P in a QWERTY keyboard) since two different hands are working. In the case of fixed layout, it is known from typewriting theory how to associate fingers to keys, so one can easily determine the time requested to type two keys consecutively. If instead, the layout is not fixed, one cannot determine the typing time for ten-fingers without strong assumptions on the association of fingers to keys. In the s-finger case, instead, the time is easily determined by (1).

QAP is known to be strongly $\mathcal{NP}$-hard, hence SK-QAP is strongly $\mathcal{NP}$-hard too. As mentioned above QAP is a challenging optimization problem. Benchmark instances nug27, nug28 and nug30 (with size 27, 28 and 30, respectively), proposed

in the late Sixties by Nugent, Vollmann and Ruml [35], have been solved exactly only recently. The fastest algorithm published so far is the branch-and-bound code by Adams, Guignard, Hahn and Hightower [3] which solves the three instances in 20 CPU days, 5 CPU months, and 2.5 CPU years, respectively, on an HP9000 C3000 workstation.

QAP has been attacked with several constructive heuristics and tens of pure and hybrid local search methods. A survey of several metaheuristic approaches for solving QAP has been presented by Drezner, Hahn and Taillard [14]. Recently James, Rego and Glover [26] proposed a comparison of almost all the most recent metaheuristic algorithms for QAP, using a large number of benchmark instances and parallel implementations.

In the next sections we propose some adaptations of basic metaheuristic algorithms to SK-QAP, and we test their effectiveness on a set of benchmark instances.

4 Neighborhoods and Speed-ups

In this section we describe the neighborhoods that we have adopted or developed for this problem, and a set of speed-up techniques used to reduce the search space and to avoid useless computations.

4.1 Neighborhoods

In a local search algorithm we use a *neighborhood* function $\mathcal{N}$, which associates with any solution φ a portion $\mathcal{N}(\varphi)$ of the solution space containing all solutions that can be obtained from φ with a 'simple' transformation. We call *move* the process that transforms φ into a neighboring solution φ'. One of the simplest neighborhood for SK-QAP involves the movement of a single symbol to a new (empty) location. Let us define *contour* the set of empty locations having at least one symbol in one of their adjacent locations. Figure 4 provides a solution of a nine symbols layout, where the shaded squares enlighten the contour locations. Note that a contour location can be either in the exterior or in the interior of the layout of a solution. Given a SK-QAP solution, we define *border location* each of the external locations containing a symbol. In Figure 4 the locations containing symbols A, B and C define the *top* border, the location containing symbol F the *right* border, the location containing symbol I the *bottom* border and the locations containing symbols A and D the *left* border. We are now ready to describe the neighborhoods and the speed-ups we implemented.

Neighborhood $\mathcal{N}_1$ (contour filling) : Set of solutions obtained by moving each symbol to an empty contour location.

Note that this neighborhood make no sense for QAP, since it has no empty location. One can see that the contour locations are $O(n)$, so the neighborhood size is $O(n^2)$. To evaluate the entire neighborhood we use a technique adapted from those proposed by Heider [24] and Burkard and Rendl [8] for QAP. Let φ be the current solution and φ' the solution obtained from φ by moving symbol i to location j. The value $z(\varphi')$ can be evaluated in $O(n)$ by computing the difference $z(\varphi') - z(\varphi)$, which is affected only by the moved symbol:

$$\Delta(\varphi, i, j) = z(\varphi') - z(\varphi) = \sum_{\substack{k=1 \\ k \neq i}}^{n} a_{ki}(b_{\varphi(k)j} - b_{\varphi(k)\varphi(i)}). \tag{4}$$

The exploration from scratch of the entire neighborhood requires $O(n^3)$ time. For QAP Frieze, Yadegar, El-Horbaty and Parkinson [17] proposed an improvement that stores the values Δ to evaluate the next neighbor in $O(n^2)$. For SK-QAP a similar improvement can be obtained. Let φ'' be the solution obtained from φ' by moving a symbol h to an empty location $l(\neq j)$ which is being a contour location of both φ and φ'. In this case $z(\varphi'')$ can be evaluated in $O(1)$ using

$$\Delta(\varphi', h, l) = z(\varphi'') - z(\varphi') = \Delta(\varphi, h, l) + a_{ih}(b_{jl} - b_{\varphi(i)l} + b_{\varphi(i)\varphi(h)} - b_{\varphi(h)j}). \tag{5}$$

If, instead, location l is a new contour location that appears when i is moved (i.e., l is adjacent to j, or equivalently, l is in the contour of i in solution φ'), then (4) must be used. But the number of possible new contour locations is a constant, so the time required to compute the solution value for all possible movements of symbol h to a new location is $O(n)$ and the computation of the entire neighborhood can be done in $O(n^2)$.

Neighborhood $\mathcal{N}_2$ (pairwise-exchange): Set of solutions obtained by swapping the assignment of two symbols r and s.

The neighborhood size is again $O(n^2)$ and can evaluated from scratch in $O(n^3)$ since the difference between the new solution φ' and the starting solution φ is:

$$\Delta(\varphi, r, s) = z(\varphi') - z(\varphi) = \sum_{\substack{k=1 \\ k \neq r,s}}^{n} (a_{kr} - a_{ks})(b_{\varphi(s)\varphi(k)} - b_{\varphi(r)\varphi(k)}). \tag{6}$$

Neighborhood $\mathcal{N}_k$ (k-exchange): Set of solutions obtained by permuting in all possible ways the assignments of k symbols.

This is generalization of the pairwise-exchange neighborhood which is obtained by setting $k = 2$. The size of the neighborhood is $O(n^k)$.

4.2 Speed-ups

The techniques used to speed up the search refer to avoid visiting equivalent solutions, and to avoid visiting some previously generated solutions.

4.2.1 Equivalent solutions

An important difference between QAP and SK-QAP, that has large impact on the solution algorithms, is the following. Two different QAP solutions are determined by two different permutations. They can have the same solution value, but the two solution structures are different. For SK-QAP, instead, two different assignments of the n symbols may result into the same layout. Consider, e.g., the six symbol assignments depicted in Figure 5. Solution (a), called "original", is transformed into equivalent assignments (b), (c) and (d), by applying in sequence a translation, a vertical reflectional symmetry and a 90° counter-clockwise rotation. (A vertical reflectional symmetry produces the mirroring image along a vertical axis.) The last solution (d) is called "canonical form", and will be discussed later in this section.

Property 1 *Consider the largest subset of the solutions space that does not contain two solutions to be obtained one from the other through a translation, a rotation, or a symmetry operation. This subset contains an optimal solution of SK-QAP.*

Another simple, but useful property concerns very "sparse"solutions.

Property 2 *A solution of SK-QAP is not optimal if there is an empty line (row or column) of the square lattice with symbols assigned on both sides of the line.*

In our algorithm we implement the restrictions of Properties 1 and 2 as follows. A solution is first shrinked applying Property 2 to remove empty rows and columns, then it is transformed by the operators of Property 1 into an equivalent solution in which:

(i) there is at least one symbol in the upper row and one symbol in the leftmost column of the square lattice;

(ii) the smallest symbol in alphabetical order on a border location is on the upper-leftmost position.

We call this solution a *canonical form*. In the example of Figure 5 we first apply a translation to satisfy condition (i) above, thus obtaining solution (b) of Figure 5. We next apply a vertical reflectional symmetry and a 90° counter-clockwise rotation to obtain the assignment (d). After the transformations there are symbols in the upper row (A and B) and in the leftmost column (D), and the smallest symbol assigned to a border location (A) is in a top border location, in the leftmost position. In order to save time, the transformation into canonical form is applied only to a solution which is candidate as starting point of our local search procedure described in Section 6.

4.2.2 Hashing

To speed up the search we introduce a *long term memory* (see below the discussion on the tabu search algorithms) that uses a sort of hashing technique to store solutions already visited and optimized. A candidate solution in canonical form is coded into a single long integer as follows. We consider the four borders of the solution (top, right, bottom and left) and we select one symbol for each side: (i) the rightmost symbol of the top border; (ii) the lowest symbol of the right border; (iii) the leftmost symbol of the bottom border and (iv) the highest symbol of the left border. In the example of Figure 5-(d) the selected symbols are B, G, F and D. For each symbol i, among the four selected one, we sum up the frequency a_{ik} between the symbol and each adjacent symbol k. The resulting value is used as a representative of the current solution and it is stored in a four-dimensional matrix in which each dimension is associated with a border and has one value for each symbol. A solution in canonical form having an hashing value already stored in an entry of the matrix is no longer considered for continuing the search.

5 Metaheuristic Algorithms for QAP

In this section we firstly resume some metaheuristic approaches which have been applied with success to QAP. Then we describe the implementations we have developed for SK-QAP.

Simulated Annealing (SA) is one of the first metaheuristic approaches, going back to the seminal papers by Kirkpatrick, Gelatt and Vecchi [27] and Černý [11].

The algorithm gets inspiration from a method used by physicians to obtain a state of minimum energy of a multi-particle physical system. SA starts with a high *temperature* value T, and performs a search in the solution space by randomly selecting neighboring solutions φ' of the current solution φ, and accepting them accordingly to the probability function

$$P(\varphi' \text{ accepted}) = \begin{cases} 1 & \text{if } z(\varphi') < z(\varphi), \\ e^{-(z(\varphi')-z(\varphi))/(k_B T)} & \text{if } z(\varphi') \geq z(\varphi), \end{cases} \tag{7}$$

where k_B is the Boltzmann constant.

After a number of iterations, depending on the so called *cooling scheme*, the temperature is reduced, thus inducing a decrease in the probability of accepting a worsening solution. For a detailed discussion on SA the reader is referred to the books by Aarts and Korst [1] and Aarts and Lenstra[2]. Burkard and Rendl [8] applied for the first time SA to the QAP using the pairwise exchange neighborhood $\mathcal{N}_2$ and a cooling scheme that halves the temperature every $2n$ iterations. Other approaches are due, e.g., to Wilhelm and Ward [44] and Connolly [13].

The *Tabu Search* (TS) method introduced by Glover [19, 20] is based on a simple yet effective concept to avoid being trapped in a local minimum: When no solution of the current neighborhood is improving, then select a worsening solution that was not visited in the past. It is impractical to store all the visited solutions and to check if each neighboring solution is identical to a previous one. Therefore Glover proposed to store in a *tabu list* only some *attribute* of each solution, or the *move* performed to transform the current solution into the neighboring one. The solutions that have the same attributes of a solution in the tabu list, or that can be generated with a move stored in the tabu list, are not considered for continuing the search. For a comprehensive introduction to tabu search algorithms we refer the reader to Glover, Taillard, and de Werra [22] and to Glover and Laguna [21].

Several other ingredients have been proposed to design effective TS algorithms. Among others some important ones are: (a) the strategy to update the tabu list length; (b) an *aspiration criterion*, i.e., a condition that in some cases cancels a tabu status; (c) a *long-term* memory, generally based on the frequency of the application of particular moves, which is used for a better guiding of the search. A TS algorithm for QAP, based on the pairwise exchange neighborhood, was presented by Skorin-Kapov [38]. Taillard [43] proposed the *robust tabu search*, that is another TS implementation based on the pairwise exchange neighborhood, but makes use of the efficient neighborhood evaluation method proposed by Frieze, Yadegar, El-Horbaty and Parkinson [17]. The tabu list consists of a matrix TL where the

facilities are associated with the rows and the locations with the columns. When an exchange assigns facility i to location j and facility i' to location j' such that $\varphi(i) = j'$ and $\varphi(i') = j$, then the algorithm stores in TL_{ij} the number of the iteration in which the swap was performed. At the following iterations a swap is declared tabu if both facilities are assigned to locations they had occupied in the last tabu tenure iterations. The tabu tenure is randomly chosen between prefixed minimum and maximum values, and is frequently updated.

Battiti and Tecchiolli [4] introduced the *Reactive Tabu Search* (RTS), which involves a sophisticated mechanism for adapting the tabu tenure and an original diversification scheme. The neighborhood and the tabu status are the same as in Skorin-Kapov [38] and Taillard [43]. The algorithm *reacts* during the evolution of the search by increasing the tabu tenure when a solution is repeated along the search, and decreasing it if no repetition occurs for a certain number of iterations. Hashing functions, binary trees and bucket lists are used to store the solutions and to check if a neighbor solution was already visited. If a solution is repeated more than once, the algorithm performs a diversification phase based on a *random walk*, i.e., on the execution of a number of random swaps which are also stored in the tabu list to avoid an immediate return to the region of the walk's starting solution. The numerical results show that RTS is competitive with robust tabu search in terms of number of iterations performed to reach the best solution. In [5] the same authors compared their RTS with an implementation of the simulated annealing by Burkard and Rendl [8]. They showed that if short computing times are allowed, then SA beats RTS, but the latter needs less CPU time than SA to reach average results which are as close as 1% to the best known solutions.

The *Variable Neighborhood Search* (*VNS*) introduced by Mladenović and Hansen [33] is a strategy that involves the use of complex neighborhoods. The basic idea is to use several neighborhood structures and to explore them in a systematic way, by increasing complexity.

Concerning QAP, an application of *VNS* was proposed by Taillard and Gambardella [42]. The neighborhoods used are the k-exchange neighborhoods with k smaller than or equal to a parameter $k_{\max}$. The exploration strategy consists of a number, say n_I, of iterations using one neighborhood at a time in a cyclic manner, as follows:

1. randomly generate a solution φ^* and set $k := 1$;
2. **for** $i := 1$ **to** n_I **do**
3. randomly select $\varphi' \in \mathcal{N}_k(\varphi^*)$;

4. apply a fast improving procedure to φ';
5. **if** $(z(\varphi') < z(\varphi^*))$ **then** set $\varphi^* := \varphi'$, $k := 1$;
6. **else** set $k := k \bmod k_{\max} + 1$
7. **endfor**

The *Ant Colony Optimization* (ACO) is a metaheuristic approach which takes inspiration from the behavior of an ant colony in search for food. In the seminal paper by Colorni, Dorigo and Maniezzo [12], an *ant* is a simple computation agent, that iteratively constructs a solution basing its decisions on the partial solution it has constructed so far, and some information on the solutions constructed by other ants. Concerning QAP, the attractiveness of assigning a facility i to a location j depends on the so called *pheromone trail*, a value stored in a global array (τ_{ij}) accessible to all ants and computed using the objective function value of the previous solutions using the $i \to j$ assignment.

The *hybrid* ACO method is an ACO algorithm that optimizes each solution through a local search method. Maniezzo and Colorni [32] proposed a hybrid ACO algorithm based on the 2-exchange neighborhood. Gambardella, Taillard and Dorigo [18] proposed a hybrid method which works with complete solutions instead of partial ones. Taillard [41] introduced the so called *Fast ANT* ($FANT$) method which is inspired by the hybrid ACO method above, but differs from it in two main respects : (i) it does not use a population, but constructs one solution at a time; (ii) when a new solution φ is generated, it reinforces each value $\tau_{i\varphi(i)}$, but at the same time it uses a parameter R to give a strong reinforcement to the values $\tau_{i\varphi^*(i)}$, where φ^* is the best solution so far. This memory updating provides a fast convergence toward the best solutions, but it also restricts the search to a small area of the search space. The algorithm is completed by a diversification method which resets matrix (τ_{ij}) if the current solution is identical to the best solution found so far. Computational experiments show that ACO methods are competitive heuristics for real life QAP instances where there are few good solutions, clustered together. For instances which have many good solutions distributed "uniformly" in the search space, they are outperformed by the other heuristics.

6 Metaheuristic algorithms for SK-QAP

In this section we present in detail the modifications that we have done to some of the metaheuristic algorithms for QAP, in order to solve SK-QAP.

Local Search

We start by describing a simple Local Search algorithm, called *LS_Refine*, that we use to drive to a local optimum a solution obtained by a more sophisticated metaheuristic algorithm. *LS_Refine* starts by finding a local optimum using neighborhood $\mathcal{N}_\mathbf{1}$, then tries to further improve the solution by looking for a local optimum with respect to $\mathcal{N}_\mathbf{2}$. If the search with $\mathcal{N}_\mathbf{2}$ improves the solution, then the procedure is reapplied from the beginning, otherwise *LS_Refine* returns the current solution. The exploration of the neighborhoods is performed with two techniques. The first one is a *best-improvement* method which explores the entire neighborhood and selects the solution with best objective function value. The corresponding algorithm has been called *LS_Refine*^{B}. The second algorithm, called *LS_Refine*^{F}, uses a *first-improvement* method. It explores $\mathcal{N}_\mathbf{1}$ by considering a symbol at a time, and selects the best location for it. The exploration of $\mathcal{N}_\mathbf{1}$ terminates as soon as the relocation of a symbol improves the objective function value. The search with neighborhood $\mathcal{N}_\mathbf{2}$ terminates as soon as the exchange of assignment between two symbols produces an improving solution.

Due to the first-improvement technique, *LS_Refine*^{F} has different behavior with different ordering of the symbols. Given a symbol i, its total frequency $f_i = \sum_{k=1}^{n} a_{ik}$ is an indicator of its importance in the objective function. Therefore, we implemented the search of both $\mathcal{N}_\mathbf{1}$ and $\mathcal{N}_\mathbf{2}$ ordering the symbols by decreasing values f_i.

Before using any of the two versions of *LS_Refine* to optimize a given solution we transform it into canonical form (see Section 4.2.1), we compute the hashing value (see Section 4.2.2) and we check the long term memory. If a solution with the same hashing value was already encountered, we do not apply the local search procedure to the current solution.

Simulated Annealing

We implemented two Simulated Annealing algorithms, called, respectively, *SA2* and *SA12*. The first one uses neighborhood $\mathcal{N}_\mathbf{2}$, while the second is based on the combined neighborhood $\mathcal{N}_\mathbf{1} \cup \mathcal{N}_\mathbf{2}$. This large neighborhood can be used in SA since each solution is randomly selected and no exploration of the entire neighborhood is required. We use a static cooling scheme which starts from an initial temperature T_0 and reduces the current temperature T every ΔT iterations by means of a linear reduction factor α. In our experiments we set T_0 to the objective function value of the starting solution and $\alpha \in \{0.95, 0.98\}$. The value ΔT is initially set to n and

then it is increased by n each time the temperature is updated. Moreover we apply *LS_Refine* to the current solution before any change of the temperature.

Tabu Search

Our Tabu Search algorithms *TS2* and *TS12* use, respectively, neighborhood $\mathcal{N}_\mathbf{1}$ and neighborhoods $\mathcal{N}_\mathbf{1}$ and $\mathcal{N}_\mathbf{2}$ in sequence. Two tabu lists TL_1 and TL_2 are associated, respectively, with $\mathcal{N}_\mathbf{1}$ and $\mathcal{N}_\mathbf{2}$. TL_1 is an array that stores in $TL(i)$ the last iteration in which symbol i was moved. Tabu list TL_2 is an upper triangular matrix which stores in element $TL_2(i,h)$ $(i < h)$ the last iteration in which symbol i and symbol h have been interchanged. The two tabu list lengths, namely l_x with $x = 1, 2$, are initialized at $\lfloor(\min_x + \max_x)/2\rfloor$. The lengths are dynamically updated, accordingly to the status of the search. Let φ' be a neighborhing solution of the current solution φ, obtained with neighborhood $\mathcal{N}_\mathbf{x}$ $(x = 1, 2)$. If $z(\varphi') < z(\varphi)$ then (improving move) we set $l_x = \max(l_x - 1, \min_x)$, otherwise (worsening move) we set $l_x = \min(l_x + 1, \max_x)$. We also adopted a method to escape from stagnating search phases. If the algorithm does not improve the best solution found so far for Δ consecutive iterations, then we clean the two tabu lists and we perturb the solution by randomly removing three symbols and reassigning them with a greedy approach that selects the location minimizing the variation of the objective function value. *LS_Refine* is used to optimize the resulting solution.

ON the basis of preliminary computational experiments we set $\min_1 = 5, \max_1 = 25$, $\min_2 = 15, \max_2 = 200$ and $\Delta = 400$.

Variable Neighborhood Search

We started from the implementation by Taillard and Gambardella [42] which uses the k-exchange neighborhoods. After some preliminary computational experiments we set the minimum value of k to 3 and the maximum to n (i.e., we consider a possible relocation of all symbols). Due to the size of the neighborhoods we set to 1 the value of n_I, i.e., we try only one solution for each value of k. Our procedure *LS_Refine* is used to optimize each neighboring solution.

FANT

We implemented the Taillard [41] *FANT* algorithm using our procedure *LS_Refine* to optimize the solution generated by the ants. The only parameter of the method

is the value R of the reinforcement given to the pheromone corresponding to the best solution so far. We have computationally evaluated the seven values $R = 4, 5, \ldots, 10$.

7 Benchmark Instances

We have generated two sets of benchmark instances. The first one consists of four instances associated with the frequencies of the symbol transitions of some important languages currently in use, whereas the second set has been randomly generated using statistical distributions similar to those of the instances in the first set. We considered the English, French, Italian and Spanish languages.

A list of the most frequent worlds of each language has been taken from the following sources: For English and Spanish the frequency lists were taken from the web site `http://www.wiktionary.org`; for French, a list of words extracted from the CD-ROM of Monde Diplomatique (1987-1997) by prof. Jean Véronis (`http://www.up.univ-mrs.fr/~veronis`); for Italian, words list have been taken from the linguistic laboratory of the Scuola Normale Superiore of Pisa (`http://alphalinguistica.sns.it`). These lists give us the frequency of appearance of each word in the corresponding language. The first 10,000 words were selected from each list. Using a parsing method the frequency of the transition between each pair of consecutive symbols has been obtained. Punctuation have been omitted from our count, while space at the beginning and at the end of each word was included, as well as the apostrophe and symbol '-' (minus), for the English language. Table 1 gives, for each language and for each symbol i, the total frequency $f_i = \sum_{k=1}^{n} a_{ik}$. We use 'ø' to denote the symbol 'space'

The second set consists of randomly generated instances. For each language, we considered its statistical properties by comparing the frequency distribution obtained with 10,000 words with the distribution associated with the 1,500 words with largest frequency. The statistical Pearson correlation test of the two distributions resulted to be greater than 0.96 for all languages. Therefore each of the 1,500 samples and the corresponding 10,000 words list share the same frequency profile. We used the short lists to generate five random benchmarks for each language, by giving to each pair of symbols a random frequency proportional to its weight in the original list. The resulting frequency distributions are very close to the original one. The two sets of instances are available in our web site `www.or.unimore.it`.

8 Computational Results

All the metaheuristic algorithms of section 6 have been coded in Delphi language and run on a PC with an INTEL Pentium 4 running at 3.0 GHz under the Windows XP operating system. We tested the algorithms on the 4 real benchmark instances and on the 20 random ones. Each algorithm was given a time limit of 120 CPU seconds for each parameter setting. In order to use integer arithmetic we multiplied the (fractional) distance values b_{jl} by 1000 and we truncated the resulting values, thus obtaining a precision of 10^{-3}.

In Tables 2 and 3 we report, for each instance and for each algorithm, the absolute gap between the solution value provided by the algorithm and the value of the best solution obtained by all algorithms. In these tables the algorithms use procedure *LS_Refine*^{F} (implementing a first-improvement search) for finding the local optima. The last row of each table gives the number of instances in which the algorithm finds the best solution. The first two columns of each table give, respectively, the number of the instance and the first letter of the corresponding language, namely E,F,I and S for English, French, Italian and Spanish. The first four rows refer to the real instances (first set), while the remaining rows present the results obtained for the four groups of 5 random instances (second set). The number of symbols varies from $n = 29$ for English to $n = 38$ for French. Concerning the size of the grid some preliminary experiments showed us that fourteen rows and columns are enough to accommodate any reasonable solution, indeed the best solutions we found are contained in a 8×8 grid. Further note that the size of the grid does not affect significantly the overall computing time, since it has a direct impact only on the implementation of Property 2.

Table 2 shows the results obtained with the simulated annealing and tabu search algorithms, in the two versions which use neighborhood $\mathcal{N}_\mathbf{2}$ or $\mathcal{N}_\mathbf{1} \cup \mathcal{N}_\mathbf{2}$, as described in Section 6 (namely algorithms *SA2*/*SA12* and *TS2*/*TS12*). The last column reports on the variable neighborhood search algorithm *VNS*. For each *SA* algorithm two values of the reduction factor α of the cooling scheme have been tested. Algorithm *SA2* with parameter α set to 0.95 dominates all other variants of the simulated annealing approach, with respect to the number of best solution found. The easiest instances appears to be the Spanish ones, while the hardest are the French. The two variants of the tabu search have similar and very good performances, since solve at the best all instances but the French instances # 2 and # 11. Worth is noting that no one of the algorithm tested in Table 2 finds the best solution for these instances. Also the performances of the *VNS* are quite satisfactory, since it finds the best solution for all instances but three.

Table 3 reports on the *FANT* algorithm with seven settings for the reinforcement parameter R. In general the performances are quite good, but the behavior changes with the parameter value. With $R = 8$ the algorithm is able to find the best solution for all instances. The Italian and Spanish instances appear to be easy for *FANT*, since the instances of both benchmark sets are solved at the best with any R value. The French instances are the harder to solve and the gap with respect to the best solution can be as large as 10^7.

The results obtained with the best-improvement technique for the local search are summarized in Table 4. We report only the best choice for each algorithm. The performances of all algorithms are worst with the best-improvement rather than with the first-improvement. This is mainly due to the fact that the choice of the next solution takes much more time, since the evaluation of the objective function is a computational expensive task. Therefore, the number of visited solution within the 120 seconds time limit is much smaller, while the improvement in the quality of the single movement is not enough to determine an overall benefit.

Finally we report in Figure 6, for each of the real language benchmarks, the layout and the corresponding solution value of the best assignment found during all runs (we draw only the minimal grid containing each solution instead of the entire 14×14 grid). One can observe some general characteristics of these solutions:

- for each language the space is very close to the center of the layout;
- no vowel (with the exception of the ones with accents) is placed on the border of the layout;
- all the locations in the corners of the keyboards are left empty (the overall shape is more "circular" than rectangular);
- although theoretically possible, no empty location exists in the interior of the layout.

9 Conclusions

In this paper we have considered the problem of designing new layouts for keyboards of small devices (like PDA and Smartphones) with the objective of minimizing the typing speed of an average message in a given language. The problem has been addressed by several researchers in the Ergonomics domain, but it has been given very limited attention in the Operations Research literature. We started

the paper by resuming the most relevant problems in keyboard design, then we concentrated on keyboards that have to be used with a single finger. We have shown that the corresponding optimization problem is a generalization of the well known Quadratic Assignment Problem (QAP). We have recalled some of the most effective metaheuristic algorithms for QAP and we have proposed extensions of some of them to the keyboard design problem. Extensive computational experiments have been performed with instances derived from word lists of English, French, Italian and Spanish languages. These benchmark instances and the best solutions that we obtained are published on the web.

The theoretical high quality of the new layouts requires now an assessment directly involving users. From this study, that has to be performed in collaboration with researchers in Ergonomics, we will get back either a confirmation of the quality of the new keyboards, or some hits to modify the objective function in order to consider in a broader way the human factors involved in the typewriting task.

References

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. Wiley, Chichester, UK, 1989.

[2] E. Aarts and J.K. Lenstra. *Local Search in Combinatorial Optimization*. Wiley, Chichester, UK, 1997.

[3] W.P. Adams, M. Guignard, P.M. Hahn, and W.L. Hightower. A level-2 reformulation-linearization technique bound for the quadratic assignment problem. *European Journal of Operational Research*, 180:983–996, 2007.

[4] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA J. Comput.*, 6:126–140, 1994.

[5] R. Battiti and G. Tecchiolli. Simulated annealing and tabu search in the long run: A comparison on qap tasks. *Computers and Mathematics with Applications*, 28:1–8, 1994.

[6] R. Burkard, M. Dell'Amico, and S. Martello. *Assignment problems*. SIAM, 2008. (To appear).

[7] R.E. Burkard and J. Offermann. Entwurf von Schreibmaschinentastaturen mittels quadratischer Zuordnungsprobleme. *Z. Oper. Res. (B)*, 21:B121–B132, 1977. (In German).

[8] R.E. Burkard and F. Rendl. A thermodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operational Research*, 17:169–174, 1984.

[9] S.K. Card, T.P. Moran, and A. Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 1983.

[10] J. Cardinal and S. Langerman. Designing small keyboards is hard. *Theor. Comput. Sci.*, 332:405–415, 2005.

[11] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *J. Optim. Theory Appl.*, 45:41–51, 1985.

[12] A. Colorni, M. Dorigo, and V. Maniezzo. The ant system: optimization by a colony of cooperating agents. *IEEE Trans. Syst. Man Cybernetics*, 26:29–41, 1996.

[13] D.T. Connolly. An improved annealing scheme for the QAP. *European Journal of Operational Research*, 46:93–100, 1990.

[14] Z. Drezner, P.M. Hahn, and E. Taillard. Recent advances for quadratic assignment problem with special emphasis on instances that are difficult for metaheuristic methods. *Ann. Oper. Res.*, 139:65–94, 2005.

[15] J. Eggers, D. Feillet, S. Kehl, M.O. Wagner, and B. Yannou. Optimization of the keyboard arrangement problem using an ant colony algorithm. *European Journal of Operational Research*, 148:672–686, 2003.

[16] P.M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *J. of Experim. Psychology*, 47:381–391, 1954.

[17] A.M. Frieze, J. Yadegar, S. El-Horbaty, and D. Parkinson. Algorithms for assignment problems on an array processor. *Parallel Comput.*, 11:151–162, 1989.

[18] L.M. Gambardella, E.D. Taillard, and M. Dorigo. Ant colonies for the QAP. *Journal of the Operational Research Society*, 50:167–176, 1999.

[19] F. Glover. Tabu search - part i. *ORSA J. Comput.*, 1:190–206, 1989.

[20] F. Glover. Tabu search - part ii. *ORSA J. Comput.*, 2:4–32, 1990.

[21] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.

[22] F. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. *Ann. Oper. Res.*, 41:12–37, 1993.

[23] W. Hargreaves, D. Rempel, N. Halpern, R. Markison, K. Kroemer, and J. Litewka. Toward a more humane keyboard. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 365–368, New York, NY, USA, 1992. ACM.

[24] C.H. Heider. A n-step, 2-variable search algorithm for the component placement problem. *Naval Res. Log. Quart.*, 20:699–724, 1973.

[25] ISO/IEC 9995-8. Information systems keyboard layouts for text and office systems part 8: Allocation of letters to keys of a numeric keypad. *International Organization for Standardization*, 1994.

[26] T. James, C. Rego, and F. Glover. Multi-start tabu search and diversification strategies for the quadratic assignment problem. *IEEE Trans. on Systems, Man and Cybernetics*, 2008. (To appear).

[27] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[28] Y. Li, L. Chen, and R. S. Goonetilleke. A heuristic-based approach to optimize keyboard design for single-finger keying applications. *Industrial Ergonomics*, 36:695–704, 2006.

[29] I.S. MacKenzie, A. Sellen, and W.A.S. Buxton. A comparison of input devices in element pointing and dragging tasks. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 161–166, New York, NY, USA, 1991. ACM.

[30] I.S. MacKenzie and R.W. Soukoreff. Text entry fo mobile computing: models and methods, theory and practice. *Human-Computer Interaction*, 17:147–198, 2002.

[31] S. MacKenzie. Fitts' law as a research and design tool in human-computer interaction. *Human-Computer Interaction*, 7:91–139, 1992.

[32] V. Maniezzo and A. Colorni. The ant system applied to the quadratic assignment problem. *IEEE Trans. Knowl. Data Engin.*, 11:769–778, 1999.

[33] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100, 1997.

[34] D.A. Norman and D. Fisher. Why alphabetic keyboards are not easy to use: Keyboard layout doesn't much matter. *Human Factors*, 24:509–519, 1982.

[35] C.E. Nugent, T.E. Vollmann, and J. Ruml. An experimental comparison of techniques for the assignment of facilities to locations. *Operations Research*, 16:150–173, 1968.

[36] M.A. Pollatschek, N. Gershoni, and Y.T. Radday. Optimization of the typewriter keyboard by simulation. *Angewandte Informatik*, 17:438–439, 1976.

[37] M.S. Sanders and E.J. McCormick. *Human Factors in Engineering and Design*. McGraw-Hill, 1993.

[38] J. Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA J. Comput.*, 2:33–45, 1990.

[39] K. Sörensen. Multi-objective optimization of mobile phone keymaps for typing messages using a word list. *European Journal of Operational Research*, 179:838–846, 2007.

[40] N.G. Swanson, T.L. Galinsky, L.L. Cole, C.S. Pan, and S.L. Sauter. The impact of keyboard design on comfort and productivity in a text-entry task. *Applied Ergonomics*, 28:9–16, 1993.

[41] E. Taillard. FANT: Fast ant system. Technical Report 46-98, IDSIA, 1998.

[42] E. Taillard and L.M. Gambardella. Adaptive memories for the quadratic assignment problem. Technical Report I-87-97, IDSIA, Lugano, 1999.

[43] E.D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Comput.*, 17:443–455, 1991.

[44] M.R. Wilhelm and T.L. Ward. Solving quadratic assignment problems by simulated annealing. *IEEE Trans.*, 19:107–119, 1987.

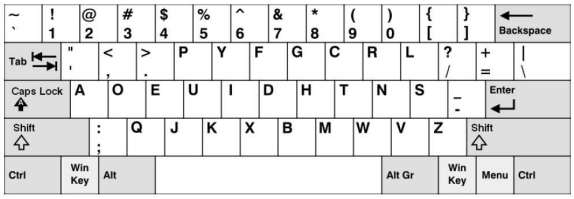
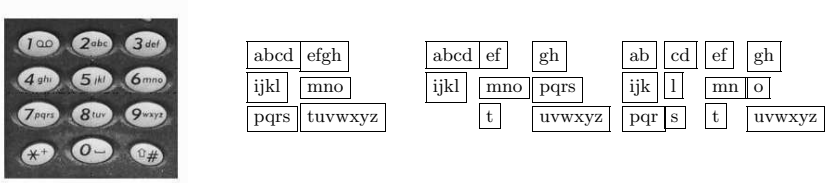


Figure 1: Dvorak layout



(a) standard ISO (b) optimal keyboards with 6,8 and 12 keys (from [10])

Figure 2: ISO and optimized contiguous keyboards

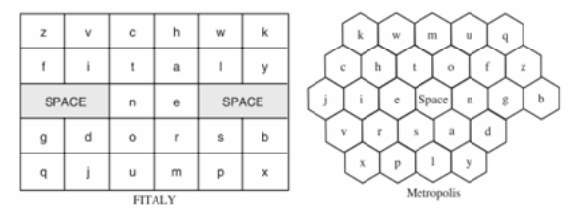


Figure 3: FITALY and Metropolis layouts

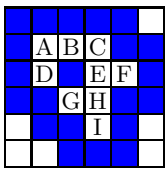


Figure 4: Contour of a nine symbol assignment

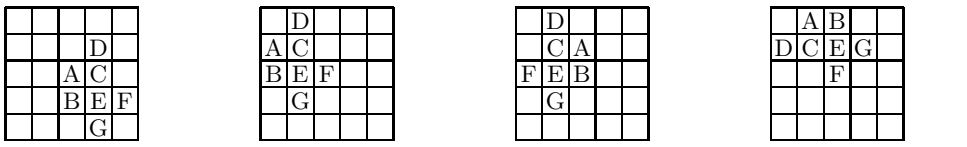


Figure 5: Equivalent solutions: (d) is in canonical form

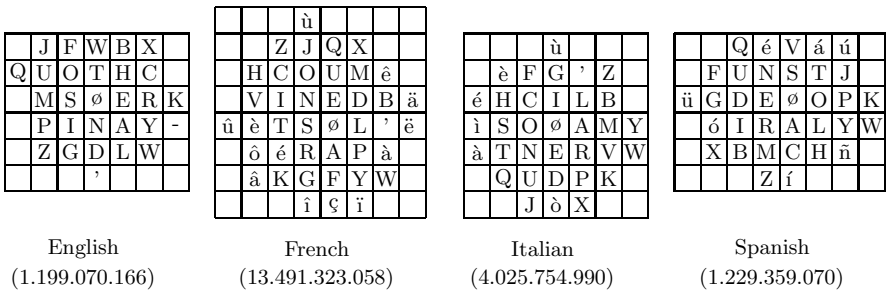


Figure 6: Best layouts for the real language benchmarks

Table 1: Total frequency of each symbol for the real language benchmarks

	English		French		Italian		Spanish	
	sym	f_i	sym	f_i	sym	f_i	sym	f_i
1	ø	1619668	ø	16897072	ø	5251966	ø	1852000
2	E	839158	E	11454806	E	2795124	E	1034122
3	T	640258	S	6214006	A	2699702	A	750196
4	O	531022	A	5747416	I	2466318	O	657204
5	A	513640	I	5694660	O	2261184	S	572004
6	N	469234	N	5636260	N	1732272	N	479322
7	I	469042	T	5388668	R	1448588	R	372966
8	H	455478	R	4951116	L	1403278	U	372214
9	S	392940	U	4505204	T	1314752	L	322082
10	R	391962	L	4334160	S	1109626	T	312322
11	D	294176	O	4208856	C	999684	D	299150
12	L	223712	D	3457398	U	784950	I	288410
13	U	185180	C	2524706	D	775526	M	210008
14	F	172990	P	2248476	P	684122	C	203402
15	M	169262	M	1990376	M	619602	P	169190
16	W	163458	é	1792198	V	370394	Q	141008
17	C	162484	’	1113636	G	358204	Y	95284
18	Y	131696	V	969024	H	308052	H	92014
19	G	126286	Q	845404	F	225670	B	86372
20	P	107224	G	774074	B	173678	V	74270
21	B	98694	F	720336	’	170618	G	69138
22	V	65426	B	562136	Z	162730	é	52890
23	K	47658	H	458702	Q	130262	á	50344
24	X	10082	à	387776	è	82692	í	50306
25	J	7938	X	359796	à	56240	ó	31726
26	’	6422	è	251904	ù	33164	J	30150
27	Q	6412	J	215136	ò	20188	F	26886
28	Z	2130	Y	186548	é	18332	Z	15448
29	-	498	ê	116026	ì	16964	ú	12076
30			ô	41520	X	4784	ñ	9688
31			Z	39700	K	4642	X	3702
32			K	38998	Y	4238	K	854
33				32760	J	3604	W	168
34			ù	26636	W	3372	ü	44
35			î	25786				
36			û	17366				
37			W	16038				
38			â	14822				
39			ï	11086				
40			ë	5686				
41			ä	252				

Table 2: *SA*, *TS* and *VNS*: first-improvement

N.	$\mathcal{L}$	*SA2*		*SA12*		*TS2*	*TS12*	*VNS*
		$\alpha = 0.95$	$\alpha = 0.98$	$\alpha = 0.95$	$\alpha = 0.98$			
1	E	0	0	0	0	0	0	0
2	F	6.461.578	18.937.729	19.299.247	580.433	5.080.177	11.857.697	5.080.177
3	I	0	0	0	0	0	0	0
4	S	0	0	0	171.439	0	0	0
5		0	0	0	0	0	0	0
6		0	10.730	0	115.629	0	0	0
7	E	0	0	0	1.290.638	0	0	0
8		0	0	0	0	0	0	0
9		0	0	0	0	0	0	0
10		0	142.729	1.297.299	131359	0	0	0
11		243.404	243.404	243.404	521.211	243.404	243.404	243.404
12	F	0	368.964	0	991.021	0	0	0
13		0	0	1.341.765	312.749	0	0	0
14		0	33.895	0	748.529	0	0	659.504
15		0	92.783	0	92.144	0	0	0
16		0	1.183.330	0	221.130	0	0	0
17	I	0	1.162.935	0	887.635	0	0	0
18		0	0	0	0	0	0	0
19		0	0	0	14.806	0	0	0
20		0	0	0	0	0	0	0
21		0	0	0	0	0	0	0
22	S	0	0	0	0	0	0	0
23		0	52.141	0	0	0	0	0
24		0	0	0	0	0	0	0
#Best		22	14	20	11	22	22	21

Table 3: *FANT*: first-improvement

		R						
N	$\mathcal{L}$	4	5	6	7	8	9	10
1	E	0	0	0	0	0	0	0
2	F	0	9.645.961	9.645.961	18.937.729	0	5.467.701	0
3	I	0	0	0	0	0	0	0
4	S	0	0	0	0	0	0	0
5		0	0	0	0	0	0	0
6		0	0	0	0	0	0	115.629
7	E	0	0	0	0	0	0	0
8		0	0	0	0	0	0	0
9		0	0	0	0	0	0	0
10		0	0	0	0	0	0	0
11		324	0	0	0	0	243.404	243.404
12	F	0	0	0	0	0	1.001.120	0
13		0	0	0	0	0	0	0
14		284.888	0	0	284.888	0	0	33.895
15		0	0	0	0	0	0	0
16		0	0	0	0	0	0	0
17	I	0	0	0	0	0	0	0
18		0	0	0	0	0	0	0
19		0	0	0	0	0	0	0
20		0	0	0	0	0	0	0
21		0	0	0	0	0	0	0
22	S	0	0	0	0	0	0	0
23		0	0	0	0	0	0	0
24		0	0	0	0	0	0	0
#Best		22	22	23	22	24	21	21

Table 4: Best-improvement technique

N.	$\mathcal{L}$	$SA2_{\alpha=0.95}$	$TS12$	VNS	$FANT_{R=7}$
1	E	0	0	0	0
2	F	5.080.177	9.645.961	5.080.177	9.645.961
3	I	0	0	0	0
4	S	572.337	0	130.371	0
5		0	0	0	0
6		0	0	0	0
7	E	0	0	0	0
8		0	0	0	0
9		0	0	0	0
10		0	0	1.282.032	1.028.313
11		243.404	243.404	243.404	1.152.708
12	F	0	306.501	0	0
13		0	1.341.765	0	0
14		861.163	0	805.748	0
15		0	0	0	0
16		0	0	0	0
17	I	0	0	0	0
18		0	0	0	0
19		0	0	903.610	0
20		0	0	0	0
21		0	0	0	0
22	S	0	0	0	0
23		0	0	0	0
24		0	0	74.364	0
#Best		20	20	17	21