GRASP WITH PATH-RELINKING FOR THE MULTI-TAP SINGLE-FINGER KEYPAD LAYOUT PROBLEM

Ricardo M. de A. Silva
Centro de Informática
Universidade Federal de Pernambuco, C.P. 50.740-560, Recife, PE, Brazil
rmas@cin.ufpe.br

Geraldo R. Mateus
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais, C.P. 31270-010, Belo Horizonte, MG, Brazil.
mateus@dcc.ufmg.br

Mauricio G. C. Resende
Internet and Network Systems Research
AT&T Labs Research, 180 Park Avenue, Room C241, Florham Park, NJ 07932 USA.
mgcr@research.att.com

Geber L. Ramalho
Centro de Informática
Universidade Federal de Pernambuco, C.P. 50.740-560, Recife, PE, Brazil
glr@cin.ufpe.br

RESUMO

Este artigo adapta a heurística GRASP with path-relinking, introduzido por Mateus et al. (2011) para o problema da atribuição quadrática generalizada (AQG), a um novo problema denominado multi-tap single-finger keypad layout problem (MTSFKLP) O MTSFKLP consiste em atribuir símbolos de uma linguagem para teclas de um teclado tal que a soma das áreas das teclas não exceda a área disponível da tecla. Dentre todas as atribuições possíveis, buscamos aquela que minimiza a soma dos produtos das frequências de transição entre cada par de caracteres e as distâncias entre as teclas nas quais os caracteres est ao atribuídos.

PALAVRAS CHAVE. grasp, path-relinking, keypad layout problem.

Área Principal: Metaheuristics

ABSTRACT

This paper adapts the GRASP with path-relinking heuristic introduced by Mateus et al. (2011) for the generalized quadratic assignment problem (GQAP), for a novel problem denominated multi-tap single-finger keypad layout problem (MTSFKLP). The MTSFKLP consists in assigning symbols (characters) of a language to keys of a keypad such that the total area of the symbols assigned to a key does not exceed the available area of the key. Among all feasible assignments, we seek one that minimizes the sum of products of frequency of the transition between each pair of consecutive characters and distances between keys to which the characters pairs are assigned.

KEYWORDS. grasp, path-relinking, keypad layout problem.

Main Area: Metaheuristics

1. Introduction

There exists several variants about keyboard layout problem (KLP) in the literature (DellAmico et al., 2009; Mittal and Sengupta, 2009; Lesher et al., 1998; Eggers et al., 2003; Cardinal and Langerman, 2005; Zhai et al., 2002; MacKenzie and Zhang, 1999; Li et al., 2006; Kreifeldt et al., 1989; Zhai et al., 2000). In this paper, we consider a novel problem denominated multi-tap single-finger keypad layout problem (MTSFKLP). The MTSFKLP consists in assigning symbols (characters) of a language to keys of a keypad such that the total area of the symbols assigned to a key does not exceed the available area of the key. Among all feasible assignments, we seek one that minimizes the sum of products of frequency of the transition between each pair of consecutive characters and distances between keys to which the characters pairs are assigned. More formally, let $N = \{1, \ldots, n\}$ denote the set of characters and $M = \{1, \ldots, m\}$ the set of keys. Furthermore, denote by $A_{n\times n} = (a_{ii'})$ the frequency of the transition between each pair of consecutive characters $i \in N$ and $i' \in N$, such that $a_{ii'} \in \Re^+$ if $i \neq i'$, otherwise $a_{ii'} = 0$; and denote by $B_{m\times m} = (b_{jj'})$ the distance between keys $j \in M$ and $j' \in M$, such that $b_{jj'} \in \Re^+$ if $j \neq j'$, otherwise $b_{jj'} = 0$. Let $q_i \in \Re^+$ be the area demanded by character $i \in N$, and $Q_j \in \Re^+$, the total available area of key $j \in M$. The MTSFKLP can be modelled as a generalized quadratic assignment problem (GQAP). It consists in finding $X_{n\times m} = (x_{ij})$, with $x_{ij} = \{0, 1\}$, where character $i \in N$ is assigned to key $j \in M$ if and only if $x_{ij} = 1$, such that the constraints

$$\sum_{j\in M} x_{ij} = 1, \forall i \in N, \tag{1}$$

$$\sum_{i\in N} q_i x_{ij} \leq Q_j, \forall j \in M, \tag{2}$$

$$x_{ij} \in \{0, 1\}, \forall i \in N, \forall j \in M$$

are satisfied and the objective function

$$\sum_{i\in N}\sum_{j\in M}\sum_{i'\in N, i'\neq i}\sum_{j'\in M} a_{ii'}b_{jj'}x_{ij}x_{i'j'}$$

is minimized. Constraints (1) guarantee that each character is assigned to exactly one key, while constraints (2) ensure that key capacities are not violated.

2. GRASP with path-relinking

A GRASP (Feo and Resende, 1989; 1995; Resende and Ribeiro, 2010) is a multi-start heuristic where at each iteration a greedy randomized solution is constructed to be used as a starting solution for local search. Local search repeatedly substitutes the current solution by a better solution in the neighborhood of the current solution. Each such replacement is called a *move*. If there is no better solution in the neighborhood, the current solution is declared a local minimum and the search stops. The best local minimum found over all GRASP iterations is output as the solution. One way to incorporate memory into GRASP is with path-relinking (Glover., 1996; Ribeiro and Resende, 2012). In GRASP with path-relinking (Laguna and Martí, 1999; Resende and Ribeiro, 2005), an elite set of diverse good-quality solutions is maintained to be used during each GRASP iteration. After a solution is produced with greedy randomized construction and local search, that solution is combined with a randomly selected solution from the elite set using the path-relinking operator. The combined solution is a candidate for inclusion in the elite set and is added to the elite set if it meets certain quality and diversity criteria.

Mateus et al. (2011) introduced a GRASP with path-relinking for the GQAP. Algorithm 1 shows pseudo-code for this algorithm, adapted for the MTSFKLP as a GQAP problem. The algorithm takes as input the set N of characters, the set M of keys, the flow matrix A, the distance matrix B, the area q_i demanded by character $i \in N$, and the key capacities $Q_j,\ j \in M$, and outputs

a feasible solution p^* specifying the key of each character in the best solution found. After initializing the elite set P as empty, the GRASP with path-relinking iterations are computed until a stopping criterion is satisfied. This criterion could be, for example. a maximum number of iterations, a target solution quality, or a maximum number of iterations without improvement. During each iteration, a greedy randomized solution is generated and local search is applied using it as a starting point, resulting in a solution p. If the greedy randomized solution is infeasible, a feasible solution is randomly selected from the elite set and local search is applied on this solution. Path-relinking is applied between p and some elite solution q only if the elite set has at least a minimum number of elements. Otherwise, solution p is simply added to the elite set if it is sufficiently different from the solutions already in the elite set. To more precisely define the term *sufficiently different*, let the *symmetric difference* $\Delta(x, y)$ between two solutions x and y be defined as the number of moves needed to transform x into y or vice-verse. For a given level of difference δ, we say x is sufficiently different from all elite solutions in P if $\Delta(x, p) > \delta$ for all $p \in P$, which we indicate by the notation $x \not\approx P$. If the elite set is not yet full, the solution r resulting from path-relinking is added to the elite set if $r \not\approx P$. Otherwise, if r is not of worse quality than any elite solution and $r \not\approx P$, then it will be added to the elite set in place of some elite solution. Among all elite solutions having cost no better than that of r, the one most similar to r, i.e. with smallest symmetric difference with respect to r, is selected to be removed from the elite set. At the end, the best elite set solution is output as the solution of the GRASP with path-relinking heuristic.

Data : N, M, A, B, q_i, Q_j.
Result : Feasible solution p^*.
$P \leftarrow \emptyset$;
while *stopping criterion not satisfied* **do**
  $p \leftarrow$ `GreedyRandomized`$(\cdot)$;
  if *elite set P has enough elements* **then**
    if *p is not feasible* **then**
      Randomly select a new solution $p \in P$;
    end
    $p \leftarrow$ `LocalSearch`(p);
    Randomly select a solution $q \in P$
    $r \leftarrow$ `PathRelinking`(p, q);
    if *elite set P is full* **then**
      if $c(r) \leq \max\{c(s) \mid s \in P\}$ *and* $r \not\approx P$ **then**
        replace the element most similar to r among all elements with cost worse than r;
      end
      else if $r \not\approx P$ **then**
        $P \leftarrow P \cup \{r\}$;
      end
    else if *p is feasible and* $p \not\approx P$ **then**
      $P \leftarrow P \cup \{p\}$;
    end
end
return $p^* = \mathbf{argmin}\{c(s) \mid s \in P\}$;

Algorithm 1: A GRASP with path-relinking heuristic for the MTSFKL problem.

2.1. Greedy randomized construction.

The construction procedure builds a solution one assignment at time. Suppose a partial solution is on hand, i.e. a number of assignments have already been made. To make the next assignment, the procedure needs to select a new character and a key. Keys are made available, one at time. The procedure randomly determines whether to use a new key or a previously chosen key, favoring a new key when the previously chosen keys have insufficient or barely sufficient available capacity. If the procedure determines that a previously chosen key is to be selected, it then determines which characters can be assigned to that key with the maximum available capacity and randomly selects one of these characters to be assigned. Of the keys that can accommodate this character, one is selected at random and the assignment is made. On the other hand, if there is no previously chosen key with sufficient capacity or if the available capacity is barely sufficient, a new key is selected at random from the set of yet unchosen keys.

The above procedure is not guaranteed to produce a feasible solution. The greedy randomized construction procedure, shown in Algorithm 2, addresses this difficulty by repeatedly applying the steps described above. The main loop in lines 1 to 21 is repeated a maximum number of times or until all characters are assigned, i.e. when $F = \emptyset$. In each iteration of the procedure, the working sets are initialized in line 2 and the threshold probability is set to 1 in line 3. The purpose of the threshold is to control whether a new key should be selected. Since it is initially set to 1, then in the first iteration of the until loop in lines 4 to 19, the procedure always selects a new (first) key. At each iteration of the until loop, the threshold is updated in line 17 such that it will be more likely that a new key is selected when there are few characters that can be assigned to keys in the current set R. The until loop consists of two stages. With probability equal to the threshold, the first stage (lines 5 to 9) selects a new key in line 6, updates the sets L and CL in line 7, and in line 8 determines the set T of characters that can be assigned to some selected key. In the second stage (lines 10 to 18), the procedure randomly selects a character from set T in line 11, updates the sets T, F, and CF in line 12, creates the key set R in line 13, randomly selects a key from that set in line 14, makes the assignment of the character to the key in line 15, determines the set T of characters that can be assigned to some selected key in line 16, and updates the threshold probability in line 17. The until loop is repeated until both sets T and L are empty in line 19. The while loop ends either with a valid assignment in line 25 (indicated by $F = \emptyset$) or with no solution found determined in line 23.

2.2. Approximate local search.

The construction procedure of Subsection 2.1 produces a feasible solution p that is not guaranteed to be locally optimal. A local search procedure is applied starting at p to find an approximate local minimum. The local search procedure makes use of two neighborhood structures which we call *1-move* and *2-move*. A solution in the 1-move neighborhood of p is obtained by changing one character-to-key assignment in p. Likewise, a solution in the 2-move neighborhood of p is obtained by simultaneously changing two character-to-key assignments in p.

One way to carry out a local search in these neighborhoods is to evaluate moves in the 1-move neighborhood and move to the first improving solution. If no 1-move improving solution exists, 2-move neighborhood solutions are evaluated and a move is made to the first improving solution. Another way to carry out the local search is to evaluate all 1-move and 2-move neighborhood solutions and move to the best improving solution. In both variants, the search is repeated until no improving solution in the neighborhoods exists. We propose a tradeoff approach here. Instead of evaluating all of the 1-move and 2-move neighborhood solutions, we sample these neighborhoods and populate a candidate list with improving solutions. One of the solutions from the candidate list is randomly selected and a move is made to that solution. The search is repeated until no improving solution is sampled. Because solutions are sampled, not all neighbors may be evaluated. Consequently, the best solution found may not be a local minimum. Mateus et al. (2011) call this solution an *approximate local minimum*.

Data : $\bar{t} =$ *maximum number of tries*
Result : Solution $x \in \mathcal{X}$
while $k < \bar{t}$ **and** $F \neq \emptyset$ **do**
  $F \leftarrow N$; $CF \leftarrow \emptyset$; $L \leftarrow M$; $CL \leftarrow \emptyset$; $T \leftarrow \emptyset$;
  Set `threshold` $\leftarrow 1$;
  repeat
    if $L \neq \emptyset$ **and** `random`$([0,1]) \leq$ `threshold` **then**
      Randomly select a key $l \in L$;
      Update sets $L \leftarrow L \setminus \{l\}$ and $CL \leftarrow CL \cup \{l\}$;
      Set $T \subseteq F$ to be all characters with area demands less than or equal to the maximum slack in CL;
    end
    if $T \neq \emptyset$ **then**
      Randomly select a character $f \in T$;
      Update sets $T \leftarrow T \setminus \{f\}$; $F \leftarrow F \setminus \{f\}$; and $CF \leftarrow CF \cup \{f\}$;
      Create set $R \subseteq CL$ to be all keys having slack greater than or equal to area demand of character f;
      Randomly select a key $l \in R$;
      Assign character f to key l;
      Set $T \subseteq F$ to be all characters with area demands less than or equal to the maximum slack in CL;
      Set `threshold` $\leftarrow 1 - \mid T \mid / \mid F \mid$;
    end
  until $T = \emptyset$ **and** $L = \emptyset$;
  $k \leftarrow k + 1$;
end
if $F \neq \emptyset$ **then**
  Solution not found;
else
  return assignment $x \in \mathcal{X}$;
end

Algorithm 2: Pseudo-code for `GreedyRandomized`: Greedy randomized construction procedure.

Pseudo-code for the approximate local search is shown in Algorithm 3. The procedure takes as input the starting solution π and two parameters, $MaxCLS$ and $MaxItr$, which control the sampling. The repeat until loop in lines 1 to 13 is repeated until an approximate local minimum is produced. In line 2, the sampling counter $count$ and the candidate list CLS are initialized. At each iteration of the inner loop in lines 3 to 9, the 1-move and 2-move neighborhoods of π are sampled without replacement by procedure $\texttt{Move}(\pi)$ in line 4. If this neighbor is an improving feasible solution, it is inserted into CLS in line 6. This is repeated until either the candidate list is full or a maximum number of neighbors have been sampled. In lines 10 to 12, if the candidate list is not empty, an assignment $\pi \in CLS$ is randomly chosen. If the set CLS is empty after the sampling process, the procedure terminates returning π as an approximate local minimum in line 14. Otherwise, the procedure moves to a solution in CLS, repeating the outer loop.

Data : $\pi, MaxCLS, MaxItr$
Result : Approximate local minimum π
1 **repeat**
2 $\quad count \leftarrow 0$; $CLS \leftarrow \emptyset$;
3 $\quad$ **repeat**
4 $\quad\quad \pi' \leftarrow \texttt{Move}(\pi)$;
5 $\quad\quad$ **if** π' *is feasible* **and** $cost(\pi') < cost(\pi)$ **then**
6 $\quad\quad\quad CLS \leftarrow CLS \cup \{\pi'\}$;
7 $\quad\quad$ **end**
8 $\quad\quad count \leftarrow count + 1$;
9 $\quad$ **until** $|CLS| \geq MaxCLS$ **or** $count \geq MaxItr$;
10 $\quad$ **if** $CLS \neq \emptyset$ **then**
11 $\quad\quad$ Randomly select a solution $\pi \in CLS$;
12 $\quad$ **end**
13 **until** $CLS = \emptyset$;
14 **return** π;

Algorithm 3: Pseudo-code for `ApproxLocalSearch`: Approximate local search procedure.

2.3. Path-relinking

Motived by the fact that a single move from a solution x in the direction of a target solution x_t does not guarantee the feasibility of the new constructed solution, a new variant of path-relinking is proposed in Mateus et al. (2011).

Suppose that among the differences between x and x_t is the key assigned to character f. In other words, while the key assigned to f in x_t is l, the key assigned to f in x is l', with $l \neq l'$. In this case, is not necessarily feasible to perform a move in x that assigns f to l. If the capacity Q_l is not violated, then the new solution is feasible. Otherwise, a repair procedure must be applied to try to make it feasible.

In this repair procedure, a character set F is created with all not yet fixed characters assigned to key l for which capacity is violated. Next, the set $T \subseteq F$ is constructed with all characters in F having area demands less than or equal to the maximum available capacity of keys in M. After a character from T is randomly selected, set R consists of keys in M that can accommodate it. A key is selected from set R and the character is assigned to it. This process is repeated until the capacity of key l has a nonnegative slack.

The path-relinking process is a sequence of steps from x_s to x_t. In each step a move is performed from the current solution x with or without repair. Next, a character i is randomly selected from a set composed of all not yet fixed characters corrected in the step. A character is *corrected* when its key becomes the same as the one assigned to it in the target solution x_t. After character i is fixed, the next step begins. This process continues until the target solution x_t is reached or when no feasible solution is obtained from x.

Algorithm 4 shows pseudo-code for the path-relinking procedure. The algorithm takes as input π_s and π_t, the starting and target solutions, respectively, and outputs the best solution π^* in path from π_s to π_t. Initially, the best solution in the path is set as π^* in line 1 and its corresponding objective function is assigned to f^* in line 2. In line 3, the current solution π' is initialized with π_s, and the working sets Fix and $nonFix$ are respectively initialized empty and with N. The while loop in lines 5 to 35 is repeated until all characters in π' are assigned to the same keys assigned to them in π_t, i.e. the set $\varphi(\pi', \pi_t) = \{i \in N \mid \pi'(i) \neq \pi_t(i)\}$ is empty, where $\pi'(i)$ and $\pi_t(i)$ are the keys assigned to character i in solutions π' and π_t, respectively.

After the set $\mathcal{B}$ of best solutions is set to empty in line 6, each while loop iteration consists of two stages. The first stage in lines 7 to 20 implements the path-relinking step. It creates set $\mathcal{B}$ with the best feasible solutions constructed from the current solution π'. In line 6, $\mathcal{B}$ is initialized as empty. Each character $v \in \varphi(\pi', \pi_t)$ is analyzed in lines 7 to 20 to create the set $\mathcal{B}$ with the best feasible solutions constructed from the current solution π'. Procedure `makeFeasible` is applied in line 8 to character v to attempt to create a new solution $\bar{\pi}$ from π'. The application of `makeFeasible` to character v can either result in a feasible or infeasible solution. In case `makeFeasible` returns a feasible solution $\bar{\pi} \notin \mathcal{B}$, $\bar{\pi}$ is added to $\mathcal{B}$ if $\mathcal{B}$ is not yet full. Otherwise, if $\mathcal{B}$ is full and solution $\bar{\pi} \notin \mathcal{B}$ is not worse than any elite solution, then $\bar{\pi}$ is added to $\mathcal{B}$ replacing some other elite solution.

In the second stage (lines 21 to 34), the procedure first randomly selects a solution π from set $\mathcal{B}$ in line 22. Then, in line 25, it selects at random a character $i \in I = \varphi(\pi', \pi_t) \setminus (\varphi(\pi', \pi_t) \cap \varphi(\pi, \pi_t))$, where I is defined in line 24 as the set containing all unfixed characters whose keys were corrected in the previous path-relinking step. A character is corrected when its key becomes the one assigned to it in the target solution. After fixing character $i \in I$, sets Fix and $nonFix$ are updated in line 26. Finally, the next path-relinking step solution π' is set as π in line 27 and, if $f(\pi') < f^*$, then the best cost f^* and best solution π^* are updated in lines 29 and 30, respectively. However, if in some path-relinking step no feasible solution is obtained from π', the while loop is interrupted, returning the current solution π^* as the result in line 33. If the target solution is reached, then π^* is returned in line 36.

This path-relinking is different from the standard variant because given solutions x_s and x_t, their commons elements are not kept fixed a priori, such that a small portion of the solution space spanned by the remaining elements is explored. The new variant fixes one character at time at each step.

3. Experimental results

In this section, we present results on computational experiments with the GRASP with path-relinking (GRASP-PR) for MTSFKP. Due to the fact that this KLP variant is original, there are not algorithms available in the literature for comparison of results. Therefore, we will present just the results of our algorithm. First, we describe our test environment. Next, we apply our implementation on four real language benchmarks: English, French, Spanish, and Italian.

3.1. Test environment.

All experiments with GRASP-PR were done on a Dell PE1950 computer with dual quad core 2.66 GHz Intel Xeon processors and 16 Gb of memory, running Red Hat Linux nesh version

Data : Starting solution π_s, target solution π_t, and candidate size factor η
Result : Best solution π^* in path from π_s to π_t

$\pi^* \leftarrow \text{argmin}\{f(\pi_s), f(\pi_t)\}$;
$f^* \leftarrow f(\pi^*)$;
$\pi' \leftarrow \pi_s$; $Fix \leftarrow \emptyset$; $nonFix \leftarrow N$;
Compute difference $\varphi(\pi', \pi_t)$ between solution π' and π_t;
while $\varphi(\pi', \pi_t) \neq \emptyset$ **do**
  $\mathcal{B} \leftarrow \emptyset$;
  for $\forall v \in \varphi(\pi', \pi_t)$ **do**
    Move the character v in π' to the same key l assigned to v in π_t;
    $\bar{\pi} \leftarrow$ `makeFeasible`(π', v);
    if $\bar{\pi}$ *is feasible* **then**
      if $|\mathcal{B}| \geq \eta \cdot |\varphi(\pi', \pi_t)|$ **then**
        if $c(\bar{\pi}) \leq \max\{c(\pi) \mid \pi \in \mathcal{B}\}$ **and** $\bar{\pi} \notin \mathcal{B}$ **then**
          replace the element most similar to $\bar{\pi}$ among all elements with cost worst than $\bar{\pi}$;
        end

      else if $\bar{\pi} \notin \mathcal{B}$ **then**
        $\mathcal{B} \leftarrow \mathcal{B} \cup \{\bar{\pi}\}$;
      end
    end
  end
  if $\mathcal{B} \neq \emptyset$ **then**
    Randomly select a solution $\pi \in \mathcal{B}$;
    Compute difference $\varphi(\pi, \pi_t)$ between solution π and π_t;
    Set $I = \varphi(\pi', \pi_t) \setminus (\varphi(\pi', \pi_t) \cap \varphi(\pi, \pi_t))$;
    Randomly select a character $i \in I$;
    $Fix \leftarrow Fix \cup \{i\}$; $nonFix \leftarrow nonFix \setminus \{i\}$;
    $\pi' \leftarrow \pi$;
    if $f(\pi') < f^*$ **then**
      $f^* \leftarrow f(\pi')$;
      $\pi^* \leftarrow \pi'$;
    end
  else
    return assignment π^*;
  end
end
return assignment π^*;

Algorithm 4: Pseudo-code for `PathRelinking`: Path-relinking procedure.

5.1.19.6 (CentOS release 5.2, kernel 2.6.18-53.1.21.el5). The GRASP-PR heuristic was implemented in Java and compiled into bytecode with `javac` version 1.6.0_05. The random-number generator is an implementation of the Mersenne Twister algorithm (Matsumoto and Nishimura., 1998) from the COLT[1] library. The stopping criteria used in the experiments were a target solution equals to zero, associated with a maximum number (100,000) of iterations without improvement.

We adopt the same procedure reported by DellAmico et al. (2009) as follows. A list of the most frequent words of each language has been taken from the following sources. For English and Spanish frequency lists web site `http://www.wiktionary.org`; for French a list of words extracted from the CDROM of Monde Diplomatique (1987-1997) by prof. Jean Véronis (`http://www.up.univ-mrs.fr/~veronis`); for Italian the word's list has 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 has 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.

For each of the real language benchmarks, we report in Figure 1 the layout and the corresponding solution value of the best assignment found by the GRASP with path-relinking for MTSFKLP during all runs: 4194.0 (English), 86.0 (French), 1290.0 (Spanish) and 0.0 (Italian). We can observe that the GRASP-PR heuristic was able to find the optimal solution for Italian language. The running times spent to find these solutions were 1.395, 0.655, 0.438, and 0.183 seconds, respectively.

4. Conclusions

In this paper, we revisted the GRASP with path-relinking heuristic for the generalized quadratic assignment problem (GQAP) of Mateus et al. (2011) and applied the heuristic to solve the multi-tap single-finger keypad layout problem (MTSFKLP) as a generalized quadratic assignment problem (GQAP). We illustrate the solution method using four real language benchmarks: English, French, Spanish, and Italian. The promising results shown here illustrate the potential of GRASP-PR for MTSFKLP.

Acknowledgment

The research of R.M.A Silva was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq), the Foundation for Support of Research of the State of Minas Gerais, Brazil (FAPEMIG), Coordination for the Improvement of Higher Education Personnel, Brazil (CAPES), Foundation for the Support of Development of the Federal University of Pernambuco, Brazil (FADE), the Office for Research and Graduate Studies of the Federal University of Pernambuco (PROPESQ), and the Foundation for Support of Science and Technology of the State of Pernambuco (FACEPE).

References

Jean Cardinal and Stefan Langerman. Designing small keyboards is hard. *Theoretical computer science*, 332(1):405–415, 2005.

Mauro DellAmico, José Carlos Díaz Díaz, Manuel Iori, and Roberto Montanari. The single-finger keyboard layout problem. *Computers & Operations Research*, 36(11):3002–3012, 2009.

Jan Eggers, Dominique Feillet, Steffen Kehl, Marc Oliver Wagner, and Bernard Yannou. Optimization of the keyboard arrangement problem using an ant colony algorithm. *Eur. J. of Operational Research*, 148(3):672–686, 2003.

[1]COLT is a open source library for high performance scientific and technical computing in Java. See `http://acs.lbl.gov/~hoschek/colt/`.

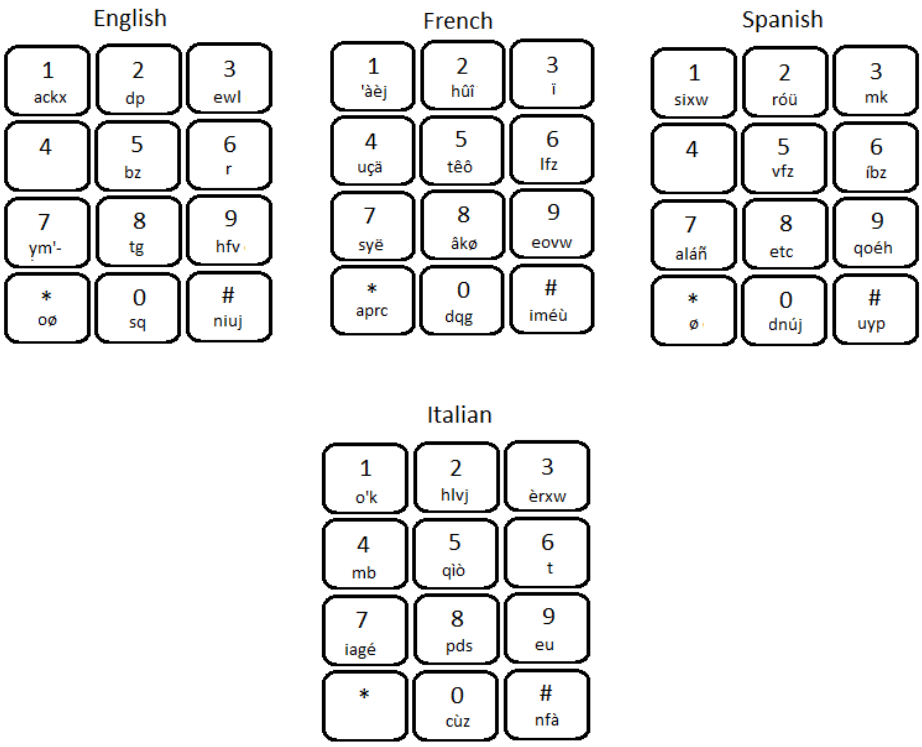

Figure 1: Best layouts for the real language benchmarks with the following values for the objective function: 4194.0 (English), 86.0 (French), 1290.0 (Spanish) and 0.0 (Italian). The running times (in seconds) spent to find these solutions were 1.395, 0.655, 0.438, and 0.183, respectively.

T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *J. of Global Optim.*, 6:109–133, 1995.

F. Glover. Tabu search and adaptive memory programing – Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, DIMACS Series in Discrete Matehematics and Theoretical Computer Science, pages 1–75. Kluwer, 1996.

JG Kreifeldt, SL Levine, and C Iyengar. Reduced keyboard designs using disambiguation. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 33, pages 441–444. SAGE Publications, 1989.

M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

Gregory W Lesher, Bryan J Moulton, and D Jeffery Higginbotham. Optimal character arrangements for ambiguous keyboards. *Rehabilitation Engineering, IEEE Transactions on*, 6(4):415–423, 1998.

Yanzhi Li, Lijuan Chen, and Ravindra S Goonetilleke. A heuristic-based approach to optimize

keyboard design for single-finger keying applications. *International journal of industrial ergonomics*, 36(8):695–704, 2006.

I Scott MacKenzie and Shawn X Zhang. The design and evaluation of a high-performance soft keyboard. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 25–31. ACM, 1999.

G. R. Mateus, M. G. C. Resende, and R. M. A. Silva. GRASP with path-relinking for the generalized quadratic assignment problem. *J. of Heuristics*, 17:527–565, 2011.

M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8: 3–30, 1998.

Arpit Mittal and Arijit Sengupta. Improvised layout of keypad entry system for mobile phones. *Computer Standards & Interfaces*, 31(4):693–698, 2009.

M. G. C. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedures: Advances and applications. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, pages 293–319. Springer, 2nd edition, 2010.

M.G.C. Resende and C.C. Ribeiro. GRASP with path-relink.: Recent advances and applic. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as Real Prob. Solvers*, pages 29–63. Springer, 2005.

C. C. Ribeiro and M. G. C. Resende. Path-relinking intensification methods for stochastic local search algorithms. *J. of Heuristics*, 18:193–214, 2012.

Shumin Zhai, Michael Hunter, and Barton A Smith. The metropolis keyboard-an exploration of quantitative techniques for virtual keyboard design. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 119–128. ACM, 2000.

Shumin Zhai, Michael Hunter, and Barton A Smith. Performance optimization of virtual keyboards. *Human–Computer Interaction*, 17(2-3):229–269, 2002.